Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

DECLARATION

I understand that this is an individual assessment and that collaboration is not permitted. I have not received any assistance with my work for this assessment. Where I have used the published work of others, I have indicated this with appropriate citation.

I have not and will not share any part of my work on this assessment, directly or indirectly, with any other student.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write."

I understand that by returning this declaration with my work, I am agreeing with the above statement. ☑

Name: Karan Dua

Date: 24-04-2023

Introduction

The primary objective of this assignment is to **perform hyper-parameter optimisation** for the given Convolutional Neural Network and measure the performance of various optimisation techniques on the given CNN model. It is utmost important to optimise and select hyper-parameters for different optimisation techniques as hyper-parameters have **significant impact on overall accuracy of the model**. It also influences how model generalises on the unseen data thereby **mitigating any impacts of over-fitting or under-fitting** of data in the model.

For this assignment, we have been provided with CNN model which is to be trained on MNIST dataset. To understand the performance of various optimisation algorithms, the model is compiled with two **optimizers, SGD with constant step size and Adam**. The model uses **categorical-cross-entropy** function to determine the loss and the goal of optimizer is to **adjust its parameters to minimize the loss**.

I have used 2 different approaches, **Global Random Search and Nelder-Mead technique**, to perform hyper-parameter optimisation and selection for this model and evaluated their performance. These approaches are **gradient-free** approaches to optimise hyper-parameters. I have used **Grid-Search as baseline** to evaluate the performance of each of these 2 Gradient-free approaches.

Therefore, the assignment emphasizes the importance of carefully selecting hyper-parameters for different optimization techniques on CNN model using MNIST dataset.

Methodology

This section elucidates the details of implementation for these 3 optimisation algorithms for CNN model. I have used gradient-free approaches like Global Random Search and Nelder-Mead to optimise the hyper-parameters of the model. I have compared the performance of both of these algorithms against Grid Search optimisation technique which is the baseline technique for our evaluation process. I have built CNN model using two different optimisers to evaluate the performance these algorithms. These optimisers are Adam and SGD with constant step size. **SGD optimizer that I have used is constant step size SGD optimizer as I am using only Learning Rate hyperparameter for to optimize the model in this assignment.** Both of these optimisers have different set of hyper-parameters that need to be tuned in order to minimize the overall loss of the model.

Finally, I have compared the results of these 3 different gradient-free optimization techniques, Global Random Search and Nelder-Mead, and Grid Search, on CNN model built using two different optimizers, Adam and SGD with constant step size. I have also tried to highlight the trade-offs between efficiency and effectiveness of these optimization techniques. This will help us to understand the significance of selecting the right optimizer and tuning its hyperparameters to achieve the best performance for a given problem.

Implementation 1: Grid Search

Grid search is a hyperparameter optimisation algorithm which generates **a pre-defined search space** and then searches for the optimized combination of hyper-parameter in that search space by **exhaustively evaluating all possible permutations and combinations of the hyper-parameters** in the pre-defined search space.

The **primary reason for choosing Grid Search as Baseline Model** is that the Grid Search **evaluates model on entire pre-defined search space, ensuring that no hyperparameter combination is missed**. For each combination of the hyperparameter in search space, the Grid Search algorithm trains and evaluated the model's performance against the metric, such as **loss or accuracy**. The **hyper-parameter combination for which we get best performance** is then chosen.

First of all, I have defined the search space for hyper-parameters of the two models. For model that uses **SGD optimizer with constant step size**, I have provided two hyper-parameters: **Learning Rate (in range [0.001,0.01,0.1]) and Batch Size (in range [32,64,128])**.

For model that uses **Adam optimizer**, I have provided four hyper-parameters: **Learning Rate (in range [0.001,0.01,0.1]), Beta 1 (in range [0.25,0.5,0.9]), Beta 2 (in range [0.9,0.99,0.999]), and Batch Size (in range [32,64,128])**. These values are **chosen on the basis of the assignments that I have completed as part of this course**.

To implement Grid Search, I have used **GridSearch** function from **keras_tuner** library. I have passed the **create model function as argument** that takes these hyper-parameters as input and compiles the model using 2 different optimizers (Adam and SGD with constant step size). I have set the **objective** argument as **val_loss** so that the grid search will be searching for the hyperparameters that results in the lowest validation loss.

For **training the model**, the Grid Search algorithm was run for 10 epochs using **search** function. In each of these epochs, grid search **finds the best hyper-parameter combination for which validation loss was minimum** in that epoch. Also, I have stored the hyperparameters, training loss and validation loss for each of these epochs to generate plots.

For **testing the model**, I have **re-used the hyper-parameters from training epochs**, for which the validation loss was minimum, to build the CNN model and then tested the model using test data which was held out separately from the training and validation data. I have **calculated test accuracy and test loss** of the model for each of these combinations and stored it in list to generate plots later. Finally, the **hyper-parameters combination for which the test loss was minimum was chosen as the ideal parameter for CNN model**.

Furthermore, I have **generated 2 plots** for each of two CNN models (Adam and SGD with constant step size optimizer). The first plot **displays the iteration wise for training loss, validation loss and test loss for the model**. The second plot highlights the **iteration wise best loss for the model for training, validation and test data**. This graph **is easier to interpret** as it only updates loss in current iteration if it was minimum then loss from previous iteration. Therefore, it is an **ideal graph to validate and select the hyper-parameters** for the CNN Model.

Finally, I have generated **another plot that compares the performance of Grid Search on CNN Model using Adam optimizer and SGD with constant step size optimizer**.

Implementation 2: Global Random Search

Global Random Search is a hyper-parameter optimisation algorithm which is based **on randomized optimisation of the hyperparameters**. This algorithm works by **randomly generating samples** of the hyperparameters from the given search space and the evaluating the model's performance against given metric. It is based on **the idea that optimized hyperparameters can be potentially found out by randomly exploring the search space**.

There **are two primary reasons** for choosing this optimisation technique. Firstly, this algorithm is **not resource intensive** as it randomly samples data from search **space instead of exploring all hyperparameter combinations in the search space,** thus making it more efficient than Grid Search. Secondly, **Global Random Search might be able to find hyperparameters that Grid Search might miss** as it as it **does not rely on any prior assumptions about the search space**. Thus, it makes it an interesting algorithm to be evaluated against the baseline Grid Search model.

First of all, I have defined the search space for hyper-parameters of the two models. For model that uses **SGD with constant step size optimizer**, I have provided two hyper-parameters: **Learning Rate (in range 1e-4 to 1e-2, using log sampling) and Batch Size (in range [32, 64, 96, 128])**.

For model that uses **Adam optimizer**, I have provided four hyper-parameters: **Learning Rate (in range 1e-4 to 1e-2, using log sampling), Beta 1 (in range [0.25,0.5,0.9]), Beta 2 (in range [0.9,0.99,0.999]), and Batch Size (in range [32, 64, 96, 128]).** These values are **chosen on the basis of the assignments that I have completed as part of this course**.

To implement Global Random Search, I have used **RandomSearch** function from **kerastuner** library. I have passed the **create model function as argument** that takes these hyper-parameters as input and compiles the model using 2 different optimizers (Adam and SGD with constant step size). I have set the **objective** argument as **val_loss** so that the grid search will be searching for the hyperparameters that results in the lowest validation loss. I have set

max_trials argument to **20** to specify the maximum number of hyperparameter combinations to be tested for optimisation in each epoch.

For **training the model**, the Global Random Search algorithm was run for 10 epochs using **search** function. In each of these epochs, the Global Random Search **finds the best hyper-parameter, among number of combinations of hyperparameters specified in max_trials argument, for which validation loss was minimum** in that epoch. Also, I have stored the hyperparameters, training loss and validation loss for each of these epochs to generate plots.

For **testing the model**, I have **re-used the hyper-parameters from training epochs**, for which the validation loss was minimum, to build the CNN model and then tested the model using test data which was held out separately from the training and validation data. I have **calculated test accuracy and test loss** of the model for each of these combinations and stored it in list to generate plots later. Finally, the **hyper-parameters combination for which the test loss was minimum was chosen as the ideal parameter for CNN model**.

Like Grid Search, I have **generated 2 plots** for each of two CNN models (Adam and SGD with constant step size optimizer) for Global Random Search Optimisation.

Finally, I have generated **another plot that compares the performance of Global Random Search on CNN Model using Adam optimizer and SGD with constant step size optimizer**.

Implementation 3: Nelder-Mead Method

Nelder-Mead is a gradient-free optimisation algorithm which is based on **iteratively refining a set of hyperparameter that define a simplex**. The simplex, which is **heuristic of the search space**, is a considered as a **geometric shape containing a set of points in a multi-dimensional space**. In each iteration, the Nelder Mead algorithm **evaluates the objective function using the hyperparameters from this heuristic search space** to find the minimum or maximum of the function by iteratively refining the search space.

There **are two primary reasons** for choosing this optimisation technique. Firstly, this algorithm is **not resource intensive** as the Nelder Mead algorithm iteratively refines the search space to converge to an optimal solution, unlike Grid Search which explores the entire search space. Secondly, Nelder Mead method can optimize a continuous search space therefore, it does not require a predefined grid of values like Grid Search. This enables Nelder Mead method to handle a large number of hyperparameters. Furthermore, it also helps it optimize complex models without needing to specify a large number of grid points, which is opposite to Grid Search. Thus, it makes it an interesting algorithm to be evaluated against the baseline Grid Search model.

First of all, I have defined the initial set of hyper-parameters of the two models. For model that uses **SGD with constant step size optimizer**, I have provided two hyper-parameters: **Learning Rate to 0.001 and Batch Size to 32**.

For model that uses **Adam optimizer**, I have provided four hyper-parameters: **Learning to 0.001, Beta 1 to 0.9, Beta 2 to 0.999, and Batch Size to 32.** This is only the initial guess for hyperparameter values that Nelder Mead uses to create simplex of search space.

To implement Nelder Mead method, I have used **minimize** function from **scipy.optimize** library. I have passed the **tune model function as argument** that takes these hyper-parameters as input and compiles the model using 2 different optimizers (Adam and SGD with constant step size). I have set the **method** argument as **Nelder-Mead** so that minimizes the function using Nelder-Mead method. I have set **maxiter** argument to **20** to specify the maximum number of epochs.

For **training the model**, first the model was built using the parameters selected from the simplex by Nelder-Mead method. I have stored the training loss and validation loss for each of these epochs to generate plots.

For **testing the model**, I have **used the trained model** and tested it using test data which was held out separately from the training and validation data. I have **calculated test accuracy and test loss** of the model and stored it in list to generate plots later. Finally, the **hyper-parameters combination for which the test loss was minimum was chosen as the ideal parameter for CNN model**.

Like Grid Search, I have **generated 2 plots** for each of two CNN models (Adam and SGD with constant step size optimizer) for Global Random Search Optimisation.

Finally, I have generated **another plot that compares the performance of Nelder-Mead method on CNN Model using Adam optimizer and SGD with constant step size optimizer**.

I have also **stored time of execution** for each of these 3 implementations of the optimisation techniques. I have used this to **compare their performance and evaluate how resource intensive** these 3 techniques are. Finally, I have **plotted another graph that compares the performance of the best models from these 3 techniques** to identify one technique which is best suited for our optimization problem. This will also help us in identifying the ideal set of optimized hyperparameter for our model. The primary objective of the current process is to select an appropriate algorithm and optimization technique by considering the task requirements and associated trade-offs.

Evaluation and Critical Discussion

Implementation 1: Grid Search with SGD with constant step size optimizer – The Baseline

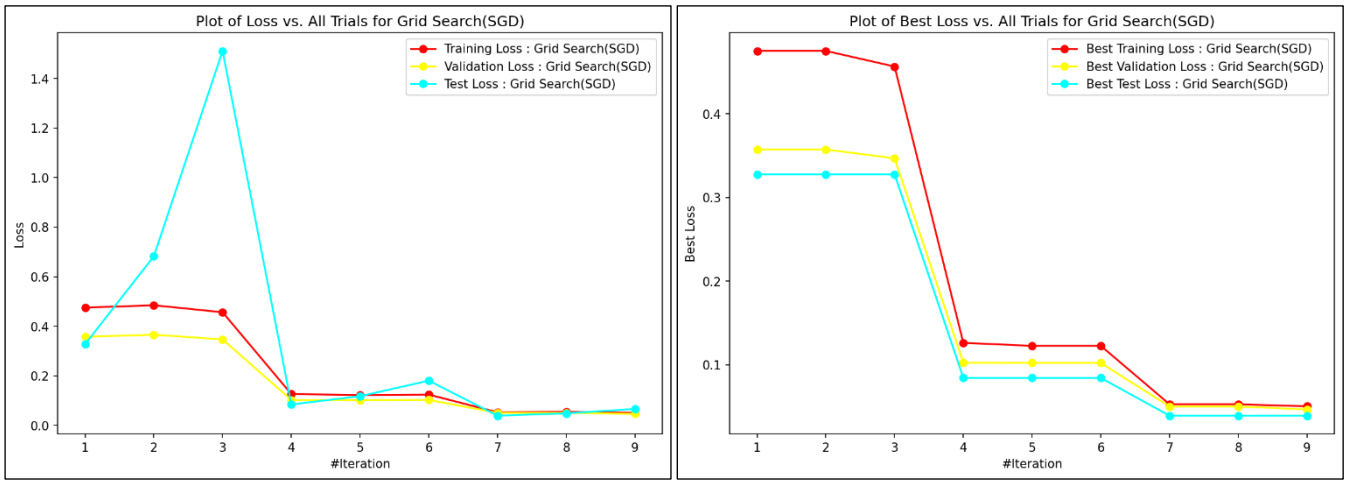

Figure 1: Plot of Loss vs Iterations for Grid Search with SGD with constant step size Optimizer for CNN model

From **Figure 1** I can clearly analyse that **test loss is less than both training and validation loss** of the model. This means that model **has generalized well and is not overfitting or underfitting**. Furthermore, I can also identify that **test loss was minimum for hyperparameters used in 7th iteration** for the model and it has not changed since then. Thus, the optimized hyperparameters from 7th iteration can be used with this CNN model using SGD with constant step size optimizer as these parameters clearly mitigate the overfitting problems in the model while minimizing the loss at the same time. Based on this analysis the optimized hyperparameters are:

	Trial	Learning Rate	Batch Size	Test Accuracy	Test Loss
0	7	0.1	32	0.9865	0.039173

Based on above result, the **optimized Learning Rate is 0.1 and Batch Size is 32** for this model according to grid search optimisation. Using these parameters, I was able to achieve overall **accuracy of 98.5% and loss was 0.039173** on test data. This model took **379 seconds** to execute completely.

Implementation 1: Grid Search with Adam optimizer – The Baseline

From **Figure 2 below** I can clearly analyse that **test loss is less than both training and validation loss** of the model. This means that model **has generalized well and is not overfitting or underfitting**. Furthermore, I can also identify that **test loss was minimum for hyperparameters used in 5th iteration** for the model and it has not changed since then. Thus, the optimized hyperparameters from 5th iteration can be used with this CNN model using Adam optimizer as these parameters clearly mitigate the overfitting problems in the model while minimizing the loss at the same time.

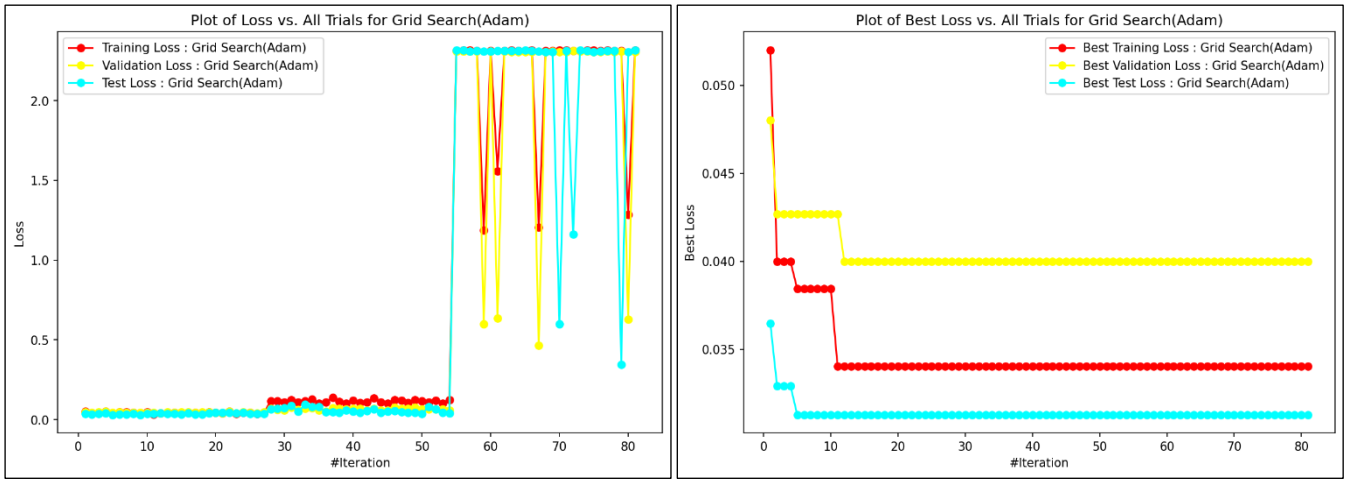

Figure 2: Plot of Loss vs Iterations for Grid Search with Adam Optimizer for CNN model

Based on this analysis the optimized hyperparameters are:

	Trial	Learning Rate	Batch Size	Beta 1	Beta 2	Test Accuracy	Test Loss
0	5	0.001	32	0.5	0.99	0.9896	0.03128

Based on above result, the **optimized Learning Rate is 0.001, Beta 1 is 0.5, Beta 2 is 0.99 and Batch Size is 32** for this model according to grid search optimisation. Using these parameters, I was able to achieve overall **accuracy of 98.96% and loss was 0.03128** on test data. This model took **3465 seconds** to execute completely.

Comparison of Grid Search (with SGD with constant step size Optimizer) and Grid Search (with Adam Optimizer)

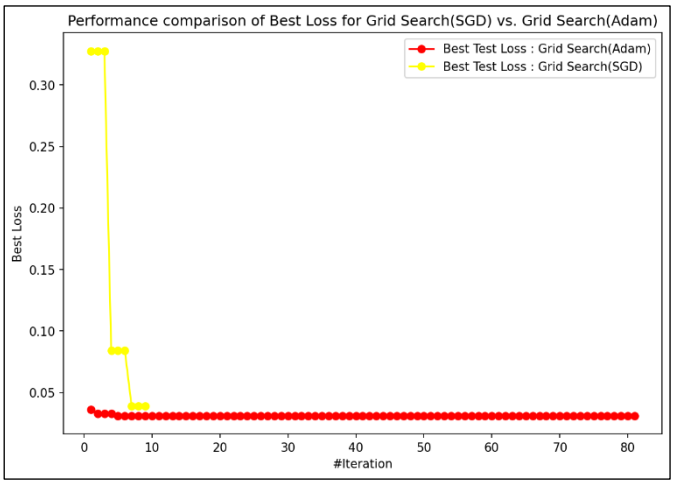

Figure 3: Plot comparing performance of Grid Search with different optimisers

From **Figure 3** I can clearly analyse that **Grid Search works best with Adam optimizer** for our CNN model as the loss on test data was less compared to that with SGD with constant step size optimizer. I can also analyse that **Grid Search with Adam optimizer explored 81 parameters as compared to 9 parameters explored in SGD with constant step size optimizer**. The primary reason for this is that in **Adam, we need to tune more hyperparameter** (Learning rate, Batch size, Beta 1, Beta 2) than SGD with constant step size (Learning rate, Batch size). But in this plot, I can analyse that **for all hyperparameter combinations, Adam optimizer always performed better than SGD with constant step size optimizer as its loss was lower for all combinations**. Hence**, Grid Search with Adam Optimizer will be used as baseline model** to compare the performance with other optimisation techniques.

Implementation 2: Global Random Search with SGD with constant step size optimizer

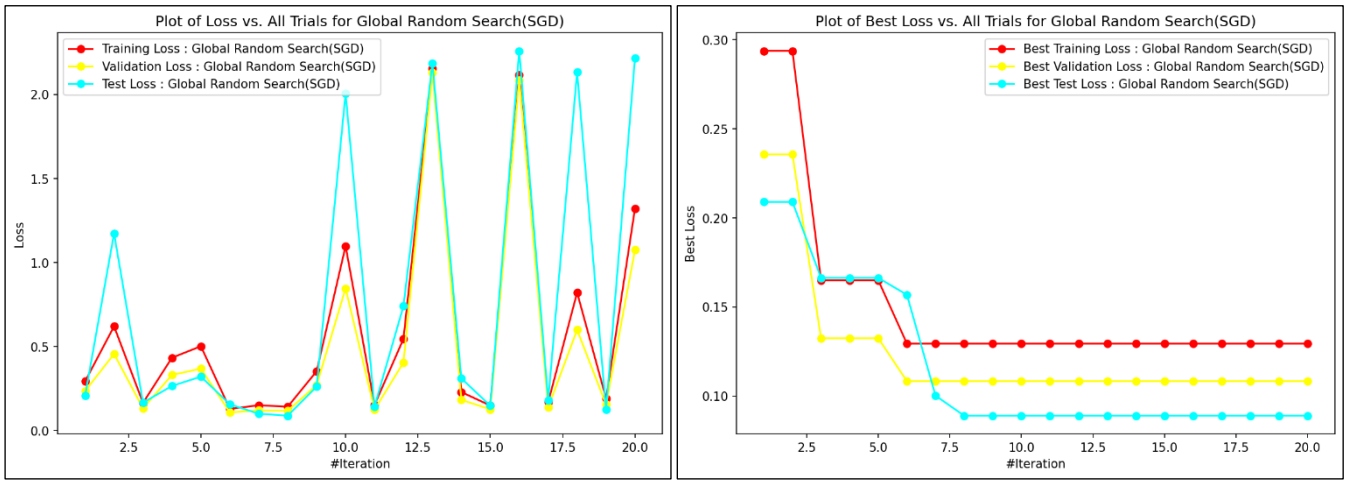

Figure 4: Plot of Loss vs Iterations for Global Random Search with SGD with constant step size Optimizer for CNN model

From **Figure 4** I can clearly analyse that after first few iterations, **test loss is less than both training and validation loss** of the model. This means that model **has generalized well and is not overfitting or underfitting**. Furthermore, I can also identify that **test loss was minimum for hyperparameters used in 8th iteration** for the model and it has not changed since then. Thus, the optimized hyperparameters from 8th iteration can be used with this CNN model using SGD with constant step size optimizer as these parameters clearly mitigate the overfitting problems in the model while minimizing the loss at the same time. Based on this analysis the optimized hyperparameters are:

	Trial	Learning Rate	Batch Size	Test Accuracy	Test Loss
0	8	0.008547	32	0.9731	0.089044

Based on above result, the **optimized Learning Rate is 0.08547 and Batch Size is 32** for this model according to global random search optimisation. Using these parameters, I was able to achieve overall **accuracy of 97.31% and loss was 0.089044** on test data. This model took **909 seconds** to execute completely.

Implementation 2: Global Random Search with Adam optimizer

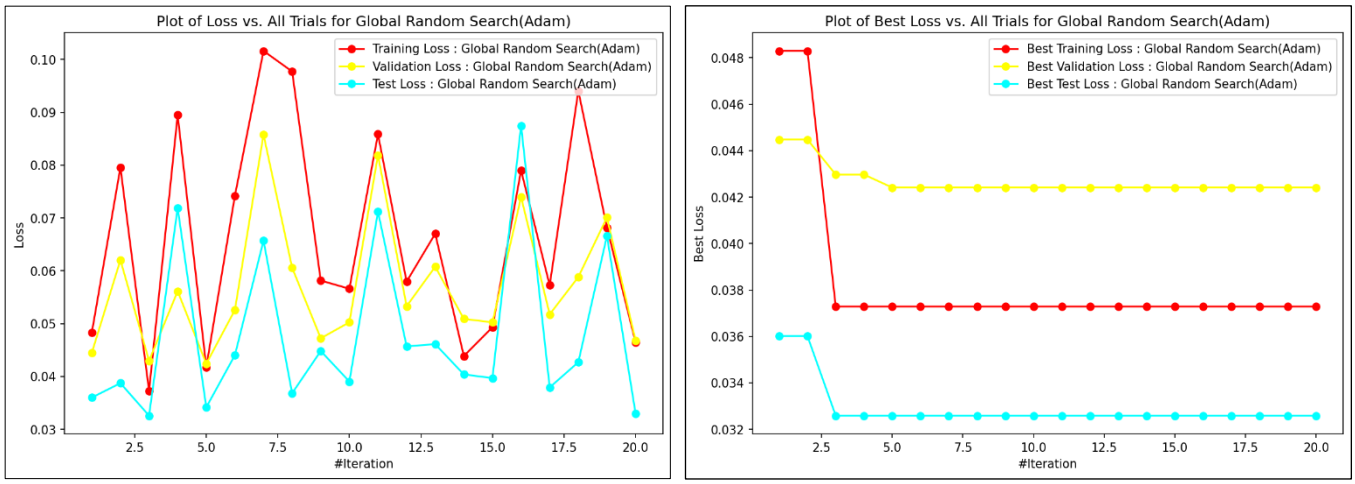

Figure 5: Plot of Loss vs Iterations for Global Random Search with Adam Optimizer for CNN model

From **Figure 5** I can clearly analyse that **test loss is less than both training and validation loss** of the model for all iterations. This means that model **has generalized well and is not overfitting or underfitting**. Furthermore, I can also identify that **test loss was minimum for hyperparameters used in 3rd iteration** for the model and it has not changed since then. Thus, the optimized hyperparameters from 3rd iteration can be used with this CNN model using Adam optimizer as these parameters clearly mitigate the overfitting problems in the model while minimizing the loss at the same time. Based on this analysis the optimized hyperparameters are:

	Trial	Learning Rate	Batch Size	Beta 1	Beta 2	Test Accuracy	Test Loss
0	3	0.000989	32	0.9	0.99	0.9888	0.032592

Based on above result, the **optimized Learning Rate is 0.000989, Beta 1 is 0.9, Beta 2 is 0.99 and Batch Size is 32** for this model according to global random search optimisation. Using these parameters, I was able to achieve overall **accuracy of 98.88% and loss was 0.032592** on test data. This model took **867 seconds** to execute completely.

Comparison of Global Random Search (with SGD with constant step size Optimizer) and Global Random Search (with Adam Optimizer)

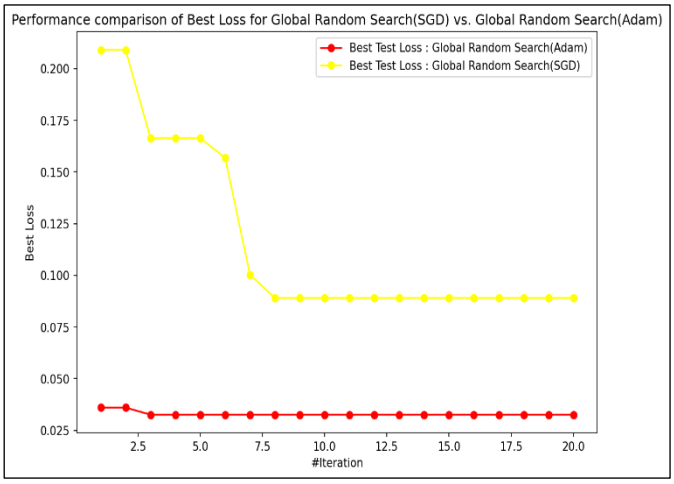

Figure 6: Plot comparing performance of Global Random Search with different optimisers

From **Figure 6** I can clearly analyse that **Global Random Search works best with Adam optimizer** for our CNN model as the loss on test data was less compared to that with SGD with constant step size optimizer. Furthermore, I can also analyse that that **for all iterations and for all random hyperparameter combinations explored by Global Random Search in these iterations, Adam optimizer always performed better than SGD with constant step size optimizer as its loss was lower for all iterations**. Hence**, Global Random Search with Adam Optimizer will be used** to compare the performance with other optimisation techniques.

Implementation 3: Nelder-Mead method with SGD with constant step size optimizer

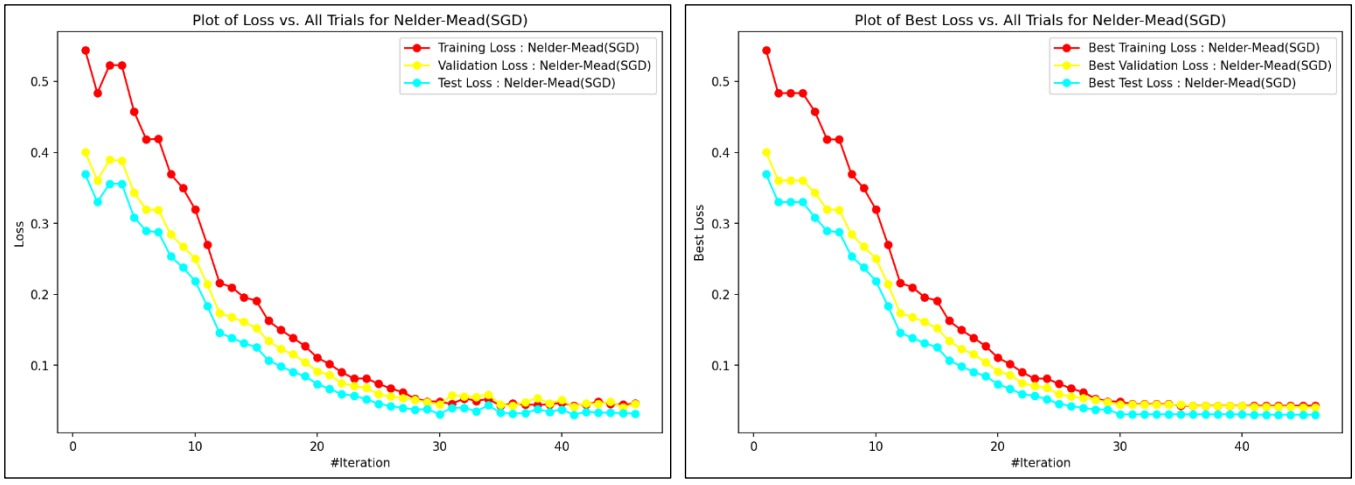

Figure 7: Plot of Loss vs Iterations for Nelder-Mead method with SGD with constant step size Optimizer for CNN model

From **Figure 7** I can clearly analyse **test loss is less than both training and validation loss** of the model for all the iterations. This means that model **has generalized well and is not overfitting or underfitting**. Furthermore, below table highlights the optimized parameters for SGD with constant step size optimizer for which the test loss was minimum.

	Learning Rate	Batch Size	Test Accuracy	Test Loss
0	0.1	16	0.9891	0.030602

Based on above result, the **optimized Learning Rate is 0.1 and Batch Size is 16** for this model according to Nelder-Mead optimisation. Using these parameters, I was able to achieve overall **accuracy of 98.91% and loss was 0.030602** on test data. This model took **2814 seconds** to execute completely.

Implementation 3: Nelder-Mead method with Adam optimizer

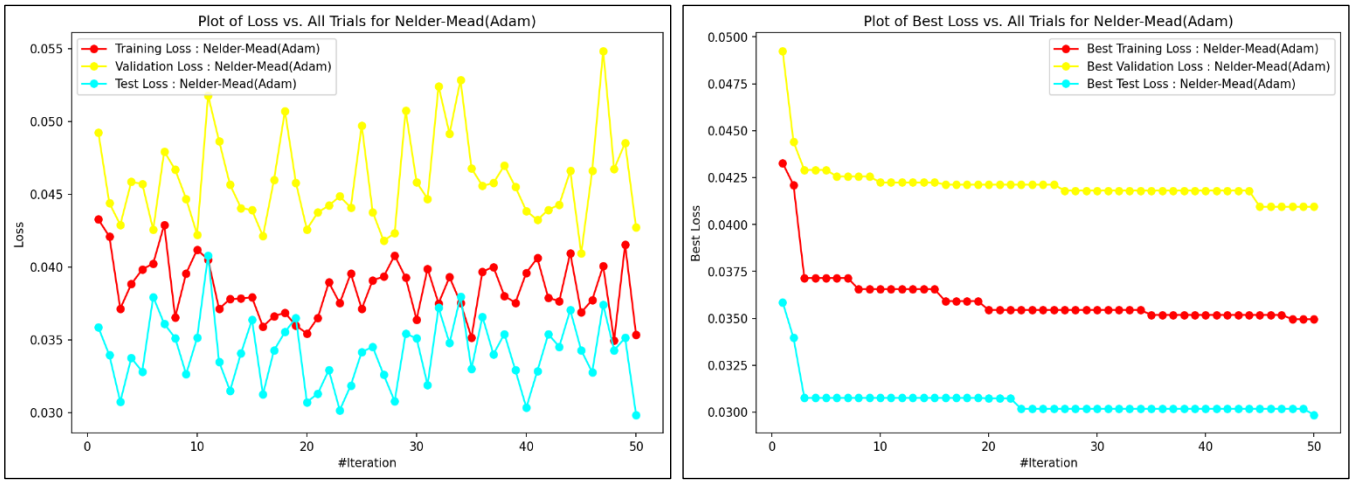

Figure 8: Plot of Loss vs Iterations for Nelder-Mead method with Adam Optimizer for CNN model

From **Figure 8** I can clearly analyse that **test loss is less than both training and validation loss** of the model for all iterations. This means that model **has generalized well and is not overfitting or underfitting**. Furthermore, below table highlights the optimized parameters for Adam optimizer for which the test loss was minimum

	Learning Rate	Batch Size	Beta 1	Beta 2	Test Accuracy	Test Loss
0	0.001035	33	0.911447	0.982217	0.9899	0.029859

Based on above result, the **optimized Learning Rate is 0.001035, Beta 1 is 0.911447, Beta 2 is 0.982217 and Batch Size is 33** for this model according to Nelder-Mead optimisation. Using these parameters, I was able to achieve overall **accuracy of 98.99% and loss was 0.029859** on test data. This model took **1735 seconds** to execute completely.

Comparison of Nelder-Mead Method (with SGD with constant step size Optimizer) and Nelder-Mead Method (with Adam Optimizer)

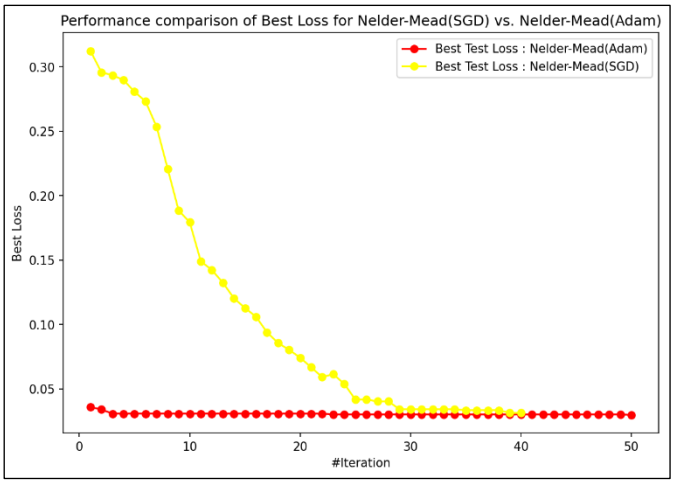

Figure 9: Plot comparing performance of Nelder-Mead method with different optimisers

From **Figure 9** I can clearly analyse that **Nelder-Mead method works best with Adam optimizer** for our CNN model as the loss on test data was less compared to that with SGD with constant step size optimizer. I can also analyse that **Nelder-Mead with Adam optimizer explored 50 parameters combinations as compared to 40 parameters combinations explored in SGD with constant step size optimizer**. The primary reason for this is that in **Adam, we need to tune more hyperparameter** (Learning rate, Batch size, Beta 1, Beta 2) than SGD with constant step size (Learning rate, Batch size). But in this plot, I can analyse that **for all hyperparameter combinations explored from simplex heuristic by Nelder-Mead method, Adam optimizer always performed better than SGD with constant step size optimizer as its loss was lower for all combinations**. Hence**, Nelder-Mead method with Adam Optimizer will be used** to compare the performance with other optimisation techniques.

Based on above results I can **conclude** that all three hyperparameter optimisation techniques, i.e., **Grid Search, Global Random Search and Nelder-Mead method, works optimally with Adam optimizer**. Now I will further compare their performance to analyse which technique is best suited for our CNN Model.

Time Comparison of Grid Search (with Adam optimizer), Global Random Search (with Adam Optimizer), and Nelder-Mead Method (with Adam Optimizer)

Algorithm	Time Taken to run training Iterations (in seconds)
Grid Search (Adam)-Baseline	3465.105232
Global Random Search (Adam)	867.4592354
Nelder-Mead (Adam)	1735.187932

Table 1: Execution time comparison for 3 optimisation techniques with Adam optimizer

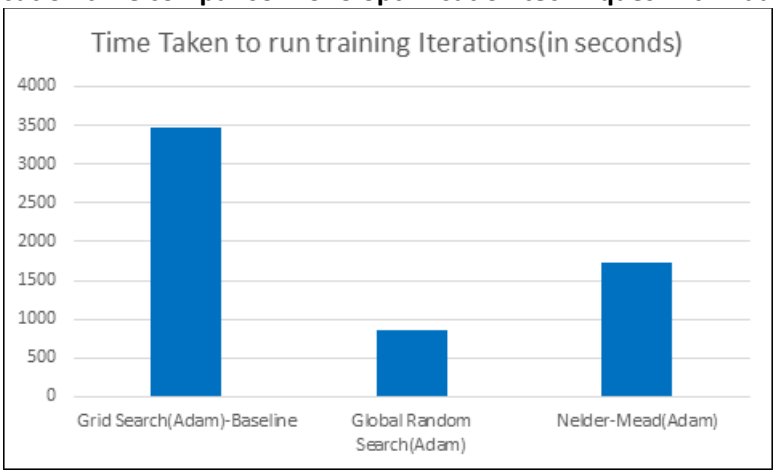

Figure 9: Plot of Time of execution for 3 optimisation techniques with Adam optimizer

From **Figure 9** I can clearly analyse that **Global Random Search takes least amount of time** to execute, therefore it is least computationally expensive as compared to other techniques. Thus**, if time/resource is the main criteria to choose the hyperparameter optimisation technique then Global Random Search is ideal technique** as it very fast as compared to others. However, **its accuracy (98.88%) is also lowest among the 3 techniques (Nelder-Mead: 98.99% and Grid Search: 98.96%)**.

Test Loss Comparison of Grid Search (with Adam optimizer), Global Random Search (with Adam Optimizer), and Nelder-Mead Method (with Adam Optimizer)

From **Figure 10 below** I can clearly analyse that **Nelder-Mead method was able to minimize the** loss to the least value. Therefore, **if minimum loss is the main criteria to choose the hyperparameter optimization technique then Nelder-Mead method is ideal technique**. Furthermore, **its accuracy (98.99%) is also highest among the 3 techniques (Global Random Search: 98.88% and Grid Search: 98.96%)**.

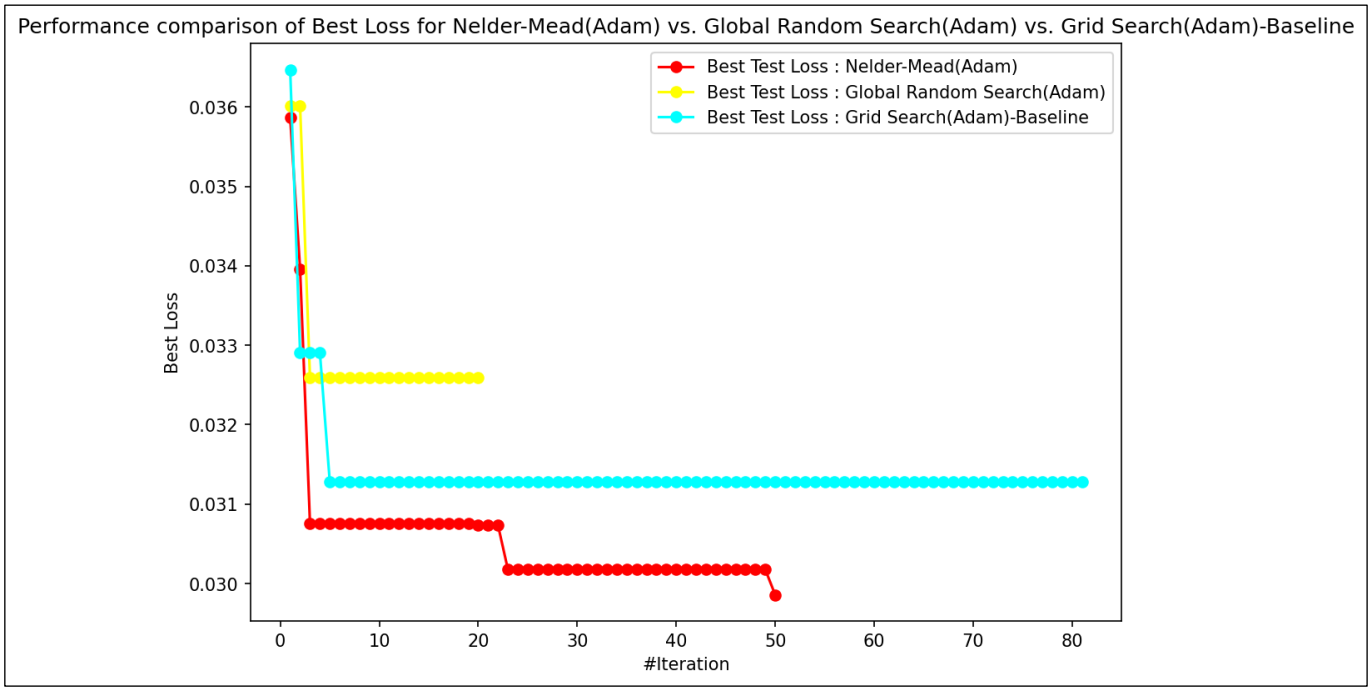

Figure 10: Plot of Test Loss for 3 optimisation techniques with Adam optimizer

Conclusion

Based on the results obtained from above evaluation and critical discussion, I can clearly **conclude that Nelder-Mead method is the ideal hyperparameter tuning technique for CNN model and it works most efficiently with Adam Optimizer**. It is **better than the baseline Grid Search technique** which explores all possible combinations of the hyper-parameters. The **primary reason** for this behavior is that **Nelder-Mead method can handle non-convex and noisy objective functions**, thus it has performed best for optimizing the hyperparameters for our CNN model. Furthermore, Nelder-Mead method **can optimize a continuous search space** therefore, it is **not dependent a predefined grid** of values like Grid Search. This enables Nelder-Mead method to **explore a large search space of hyperparameters** as compared to Grid Search. Furthermore, **Nelder-Mead method takes less time to execute and is computationally less expensive than Grid Search** for optimizing our CNN model.

Furthermore, **Global Random Search has also performed well** in this experiment. It is **computationally least expensive**. Although, its performance was not better than the Baseline Grid Search model as it was not able to minimize the loss to minima but **the difference in performance level is not substantial enough given the performance difference is huge** in terms of time and utilization of resources. Thus, **Global Random Search is an appropriate alternative to Nelder-Mead method** if the availability of the resources is constrained.

Grid Search is an ideal choice for hyperparameter optimisation **if and only if small number of hyperparameters** are present resulting in a smaller search space of parameters to be optimized. The Grid search technique becomes **extensively computationally expensive as the number of parameters to be optimized increases** due to increase in the combinations if parameters that this algorithm has to explore.

Lastly, all these 3 optimisation techniques performed best with Adam optimiser when compared to SGD with constant step size optimizer. The difference in performance was also significant and there was not much difference in resource utilization during the optimization process. Therefore, we should prefer Adam optimizer over SGD with constant step size optimizer for our CNN model.

Appendix: Code for Grid Search SGD

```
import numpy as np
import keras
import tensorflow as tf
from tensorflow.keras import layers
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from tensorflow.keras.optimizers import SGD, Adam
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import train_test_split
from kerastuner import GridSearch
import matplotlib.pyplot as plot
from keras.utils import np_utils
import pandas as pd

from IPython.display import display
import time

num_classes = 10
input_shape = (28, 28, 1)

(x_train, y_train), (x_test, y_test) = mnist.load_data()

n=30000
x_train = x_train[1:n]
y_train=y_train[1:n]

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

x_train, x_validation, y_train, y_validation = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

def create_Grid_Search_SGD_model(parameters):

    selectedLearningRate = parameters.Choice('learning_rate',values=[0.001,0.01,0.1])
    selectedBatchSize = parameters.Choice('batch_size',values= [32,64,128])

    model = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation="softmax"),
        ]
    )

    optimizer = SGD(learning_rate = selectedLearningRate)

    model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])

    return model

gridSearch_SGD_optimiser = GridSearch(
                create_Grid_Search_SGD_model,
                objective = 'val_loss',
```

```
            seed = 42,
            directory = 'GridSearch_SGD',
            project_name = 'mnist_data'
        )

start_time = time.time()
gridSearch_SGD_optimiser.search(x_train, y_train, validation_data = (x_validation, y_validation), epochs = 10)
end_time = time.time()
print(f'\n\nTotal time to execute Grid Search(SGD) is : {end_time - start_time} seconds')

gridSearch_SGD_param_values = gridSearch_SGD_optimiser.oracle.trials.values()

trainingLoss_data = []
validationLoss_data = []
bestTrainingLoss_data = []
bestValidationLoss_data = []

currentBestTrainingLoss = currentBestValidationLoss = float("inf")


parameters = []
counter = 1

for parameter in gridSearch_SGD_param_values:
  trainingLoss = parameter.metrics.get_history('loss')[-1].value[0]
  validationLoss = parameter.metrics.get_history('val_loss')[-1].value[0]
  trainingAccuracy = parameter.metrics.get_history('accuracy')[-1].value[0]
  validationAccuracy = parameter.metrics.get_history('val_accuracy')[-1].value[0]
  trainingLoss_data.append(trainingLoss)
  validationLoss_data.append(validationLoss)

  parameter_dictionary = {
    "Trial": counter,
    "Learning Rate": parameter.hyperparameters.get("learning_rate"),
    "Batch Size": parameter.hyperparameters.get("batch_size"),
    "Training Accuracy": trainingAccuracy,
    "Validation Accuracy": validationAccuracy,
    "Training Loss": trainingLoss,
    "Validation Loss": validationLoss
  }

  if validationLoss < currentBestValidationLoss:
    currentBestValidationLoss = validationLoss

  if trainingLoss < currentBestTrainingLoss:
    currentBestTrainingLoss = trainingLoss

  bestTrainingLoss_data.append(currentBestTrainingLoss)
  bestValidationLoss_data.append(currentBestValidationLoss)

  parameters.append(parameter_dictionary)
  counter += 1

gridSearch_SGD_Parameters_df = pd.DataFrame(parameters)

display(gridSearch_SGD_Parameters_df)

testLoss_data = []
bestTestLoss_data = []
currentBestTestLoss = float("inf")
optimisedParameters = {}

counter = 1
```

```
for parameter in gridSearch_SGD_param_values:

    model = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation="softmax"),
        ]
    )

    optimizer = SGD(learning_rate = parameter.hyperparameters.get('learning_rate'))

    model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])

    model.fit(x_train, y_train, validation_data=(x_validation, y_validation), batch_size = parameter.hyperparameters.get('batch_size'),
epochs = 10)
    testLoss, testAccuracy = model.evaluate(x_test, y_test, verbose=0)
    testLoss_data.append(testLoss)

    if testLoss < currentBestTestLoss:
        currentBestTestLoss = testLoss
        optimisedParameters = {
                            "Trial": counter,
                            "Learning Rate": parameter.hyperparameters.get("learning_rate"),
                            "Batch Size": parameter.hyperparameters.get("batch_size"),
                            "Test Accuracy": testAccuracy,
                            "Test Loss": testLoss
                        }
    bestTestLoss_data.append(currentBestTestLoss)

    counter += 1

gridSearch_SGD_Test_Parameters_df = pd.DataFrame([optimisedParameters])
display(gridSearch_SGD_Test_Parameters_df)

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(gridSearch_SGD_param_values) + 1), trainingLoss_data, label="Training Loss : Grid Search(SGD)", marker='o',
color="Red")
plot.plot(range(1, len(gridSearch_SGD_param_values) + 1), validationLoss_data, label="Validation Loss : Grid Search(SGD)", marker='o',
color="Yellow")
plot.plot(range(1, len(gridSearch_SGD_param_values) + 1), testLoss_data, label="Test Loss : Grid Search(SGD)", marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Loss")
plot.title('Plot of Loss vs. All Trials for Grid Search(SGD)')
plot.legend()
plot.show()

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(gridSearch_SGD_param_values) + 1), bestTrainingLoss_data, label="Best Training Loss : Grid Search(SGD)",
marker='o', color="Red")
plot.plot(range(1, len(gridSearch_SGD_param_values) + 1), bestValidationLoss_data, label="Best Validation Loss : Grid Search(SGD)",
marker='o', color="Yellow")
plot.plot(range(1, len(gridSearch_SGD_param_values) + 1), bestTestLoss_data, label="Best Test Loss : Grid Search(SGD)", marker='o',
color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Best Loss")
plot.title('Plot of Best Loss vs. All Trials for Grid Search(SGD)')
```

```
plot.legend()
plot.show()

bestTest_GS_SGD_DF = pd.DataFrame(bestTestLoss_data, columns=['BestTestLoss_GS_SGD'])
bestTest_GS_SGD_DF.to_csv('GridSearch_SGD_BestTestLoss.csv', index=False)
```

Appendix: Code for Grid Search Adam

```
get_ipython().system('pip install keras-tuner --upgrade')
import numpy as np
import keras
import tensorflow as tf
from tensorflow.keras import layers
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from tensorflow.keras.optimizers import SGD, Adam
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import train_test_split
from keras_tuner import GridSearch
import matplotlib.pyplot as plot
from keras.utils import np_utils
import pandas as pd

from IPython.display import display
import time
from tqdm import tqdm

num_classes = 10
input_shape = (28, 28, 1)

(x_train, y_train), (x_test, y_test) = mnist.load_data()

n=30000
x_train = x_train[1:n]
y_train=y_train[1:n]

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

x_train, x_validation, y_train, y_validation = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

def create_Grid_Search_Adam_model(parameters):

    selectedLearningRate = parameters.Choice('learning_rate',values=[0.001,0.01,0.1])
    selectedBatchSize = parameters.Choice('batch_size',values= [32,64,128])
    selectedBeta1 = parameters.Choice('beta_1',values= [0.25,0.5,0.9])
    selectedBeta2 = parameters.Choice('beta_2',values= [0.9,0.99,0.999])

    model = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation="softmax"),
        ]
    )

    optimizer = Adam(learning_rate=selectedLearningRate, beta_1=selectedBeta1, beta_2=selectedBeta2)

    model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])

    return model

gridSearch_Adam_optimiser = GridSearch(
                create_Grid_Search_Adam_model,
                objective = 'val_loss',
```

```
            seed = 42,
            directory = 'GridSearch_Adam',
            project_name = 'mnist_data'
        )

gridSearch_Adam_param_values = gridSearch_Adam_optimiser.oracle.trials.values()

trainingLoss_data = []
validationLoss_data = []
bestTrainingLoss_data = []
bestValidationLoss_data = []

currentBestTrainingLoss = currentBestValidationLoss = float("inf")


parameters = []
counter = 1

for parameter in gridSearch_Adam_param_values:
    trainingLoss = parameter.metrics.get_history('loss')[-1].value[0]
    validationLoss = parameter.metrics.get_history('val_loss')[-1].value[0]
    trainingAccuracy = parameter.metrics.get_history('accuracy')[-1].value[0]
    validationAccuracy = parameter.metrics.get_history('val_accuracy')[-1].value[0]
    trainingLoss_data.append(trainingLoss)
    validationLoss_data.append(validationLoss)

    parameter_dictionary = {
        "Trial": counter,
        "Learning Rate": parameter.hyperparameters.get("learning_rate"),
        "Batch Size": parameter.hyperparameters.get("batch_size"),
        "Beta 1": parameter.hyperparameters.get("beta_1"),
        "Beta 2": parameter.hyperparameters.get("beta_2"),
        "Training Accuracy": trainingAccuracy,
        "Validation Accuracy": validationAccuracy,
        "Training Loss": trainingLoss,
        "Validation Loss": validationLoss
    }

    if validationLoss < currentBestValidationLoss:
        currentBestValidationLoss = validationLoss

    if trainingLoss < currentBestTrainingLoss:
        currentBestTrainingLoss = trainingLoss

    bestTrainingLoss_data.append(currentBestTrainingLoss)
    bestValidationLoss_data.append(currentBestValidationLoss)

    parameters.append(parameter_dictionary)
    counter += 1

gridSearch_Adam_Parameters_df = pd.DataFrame(parameters)

start_time = time.time()
gridSearch_Adam_optimiser.search(x_train, y_train, validation_data = (x_validation, y_validation), epochs = 10)
end_time = time.time()
print(f'\n\nTotal time to execute Grid Search(Adam) is : {end_time - start_time} seconds')

gridSearch_Adam_param_values = gridSearch_Adam_optimiser.oracle.trials.values()

trainingLoss_data = []
validationLoss_data = []
bestTrainingLoss_data = []
bestValidationLoss_data = []

currentBestTrainingLoss = currentBestValidationLoss = float("inf")


parameters = []
```

```
counter = 1

for parameter in gridSearch_Adam_param_values:
    trainingLoss = parameter.metrics.get_history('loss')[-1].value[0]
    validationLoss = parameter.metrics.get_history('val_loss')[-1].value[0]
    trainingAccuracy = parameter.metrics.get_history('accuracy')[-1].value[0]
    validationAccuracy = parameter.metrics.get_history('val_accuracy')[-1].value[0]
    trainingLoss_data.append(trainingLoss)
    validationLoss_data.append(validationLoss)

    parameter_dictionary = {
        "Trial": counter,
        "Learning Rate": parameter.hyperparameters.get("learning_rate"),
        "Batch Size": parameter.hyperparameters.get("batch_size"),
        "Beta 1": parameter.hyperparameters.get("beta_1"),
        "Beta 2": parameter.hyperparameters.get("beta_2"),
        "Training Accuracy": trainingAccuracy,
        "Validation Accuracy": validationAccuracy,
        "Training Loss": trainingLoss,
        "Validation Loss": validationLoss
    }

    if validationLoss < currentBestValidationLoss:
        currentBestValidationLoss = validationLoss

    if trainingLoss < currentBestTrainingLoss:
        currentBestTrainingLoss = trainingLoss

    bestTrainingLoss_data.append(currentBestTrainingLoss)
    bestValidationLoss_data.append(currentBestValidationLoss)

    parameters.append(parameter_dictionary)
    counter += 1

gridSearch_Adam_Parameters_df = pd.DataFrame(parameters)

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', None)
display(gridSearch_Adam_Parameters_df)

testLoss_data = []
bestTestLoss_data = []
currentBestTestLoss = float("inf")
optimisedParameters = {}

counter = 1

for parameter in tqdm(gridSearch_Adam_param_values):

    model = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation="softmax"),
        ]
    )

    optimizer = Adam(learning_rate=parameter.hyperparameters.get('learning_rate'), beta_1=parameter.hyperparameters.get('beta_1'),
beta_2=parameter.hyperparameters.get('beta_2'))

    model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

```
    model.fit(x_train, y_train, validation_data=(x_validation, y_validation), batch_size = parameter.hyperparameters.get('batch_size'), epochs = 10, verbose=0)
    testLoss, testAccuracy = model.evaluate(x_test, y_test, verbose=0)
    testLoss_data.append(testLoss)

    if testLoss < currentBestTestLoss:
        currentBestTestLoss = testLoss
        optimisedParameters = {
                            "Trial": counter,
                            "Learning Rate": parameter.hyperparameters.get("learning_rate"),
                            "Batch Size": parameter.hyperparameters.get("batch_size"),
                            "Beta 1": parameter.hyperparameters.get("beta_1"),
                            "Beta 2": parameter.hyperparameters.get("beta_2"),
                            "Test Accuracy": testAccuracy,
                            "Test Loss": testLoss
                        }
    bestTestLoss_data.append(currentBestTestLoss)

    counter += 1

gridSearch_Adam_Test_Parameters_df = pd.DataFrame([optimisedParameters])
display(gridSearch_Adam_Test_Parameters_df)

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(gridSearch_Adam_param_values) + 1), trainingLoss_data, label="Training Loss : Grid Search(Adam)", marker='o', color="Red")
plot.plot(range(1, len(gridSearch_Adam_param_values) + 1), validationLoss_data, label="Validation Loss : Grid Search(Adam)", marker='o', color="Yellow")
plot.plot(range(1, len(gridSearch_Adam_param_values) + 1), testLoss_data, label="Test Loss : Grid Search(Adam)", marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Loss")
plot.title('Plot of Loss vs. All Trials for Grid Search(Adam)')
plot.legend()
plot.show()

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(gridSearch_Adam_param_values) + 1), bestTrainingLoss_data, label="Best Training Loss : Grid Search(Adam)", marker='o', color="Red")
plot.plot(range(1, len(gridSearch_Adam_param_values) + 1), bestValidationLoss_data, label="Best Validation Loss : Grid Search(Adam)", marker='o', color="Yellow")
plot.plot(range(1, len(gridSearch_Adam_param_values) + 1), bestTestLoss_data, label="Best Test Loss : Grid Search(Adam)", marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Best Loss")
plot.title('Plot of Best Loss vs. All Trials for Grid Search(Adam)')
plot.legend()
plot.show()

bestTest_GS_Adam_DF = pd.DataFrame(bestTestLoss_data, columns=['BestTestLoss_GS_Adam'])
bestTest_GS_Adam_DF.to_csv('GridSearch_Adam_BestTestLoss.csv', index=False)
```

Appendix: Code for Grid Search Comparison

```
import pandas as pd
from IPython.display import display
import matplotlib.pyplot as plot

GridSearch_Adam_BestTestLoss_df = pd.read_csv('GridSearch_Adam_BestTestLoss.csv')
GridSearch_SGD_BestTestLoss_df = pd.read_csv('GridSearch_SGD_BestTestLoss.csv')

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', None)
display(GridSearch_Adam_BestTestLoss_df)

display(GridSearch_SGD_BestTestLoss_df)

bestTestLoss_GS_Adam = GridSearch_Adam_BestTestLoss_df['BestTestLoss_GS_Adam']
bestTestLoss_GS_SGD = GridSearch_SGD_BestTestLoss_df['BestTestLoss_GS_SGD']

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(bestTestLoss_GS_Adam) + 1), bestTestLoss_GS_Adam, label="Best Test Loss : Grid Search(Adam)", marker='o', color="Red")
plot.plot(range(1, len(bestTestLoss_GS_SGD) + 1), bestTestLoss_GS_SGD, label="Best Test Loss : Grid Search(SGD)", marker='o', color="Yellow")
plot.xlabel("#Iteration")
plot.ylabel("Best Loss")
plot.title('Performance comparison of Best Loss for Grid Search(SGD) vs. Grid Search(Adam)')
plot.legend()
plot.show()
```

Appendix: Code for Global Random Search SGD

```
import numpy as np
import keras
import tensorflow as tf
from tensorflow.keras import layers
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from tensorflow.keras.optimizers import SGD, Adam
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import train_test_split
from kerastuner import RandomSearch
import matplotlib.pyplot as plot
from keras.utils import np_utils
import pandas as pd

from IPython.display import display
import time
from tqdm import tqdm

num_classes = 10
input_shape = (28, 28, 1)

(x_train, y_train), (x_test, y_test) = mnist.load_data()

n=30000
x_train = x_train[1:n]
y_train=y_train[1:n]

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

x_train, x_validation, y_train, y_validation = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

def create_GLobal_Random_SGD_model(parameters):

    selectedLearningRate = parameters.Float('learning_rate', 1e-4, 1e-2, sampling='log')
    selectedBatchSize = parameters.Choice('batch_size',values= [32,64,96,128])

    model = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation="softmax"),
        ]
    )

    optimizer = SGD(learning_rate = selectedLearningRate)

    model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])

    return model

globalRandomSearch_SGD_optimiser = RandomSearch(
                create_GLobal_Random_SGD_model,
```

```
            objective = 'val_loss',
            seed = 42,
            max_trials=20,
            directory = 'GlobalRandomSearch_SGD',
            project_name = 'mnist_data'
        )

start_time = time.time()
globalRandomSearch_SGD_optimiser.search(x_train, y_train, validation_data = (x_validation, y_validation), epochs = 10)
end_time = time.time()
print(f'\n\nTotal time to execute Global Random Search(SGD) is : {end_time - start_time} seconds')

globalRandomSearch_SGD_param_values = globalRandomSearch_SGD_optimiser.oracle.trials.values()

trainingLoss_data = []
validationLoss_data = []
bestTrainingLoss_data = []
bestValidationLoss_data = []

currentBestTrainingLoss = currentBestValidationLoss = float("inf")

parameters = []
counter = 1

for parameter in globalRandomSearch_SGD_param_values:
    trainingLoss = parameter.metrics.get_history('loss')[-1].value[0]
    validationLoss = parameter.metrics.get_history('val_loss')[-1].value[0]
    trainingAccuracy = parameter.metrics.get_history('accuracy')[-1].value[0]
    validationAccuracy = parameter.metrics.get_history('val_accuracy')[-1].value[0]
    trainingLoss_data.append(trainingLoss)
    validationLoss_data.append(validationLoss)

    parameter_dictionary = {
        "Trial": counter,
        "Learning Rate": parameter.hyperparameters.get("learning_rate"),
        "Batch Size": parameter.hyperparameters.get("batch_size"),
        "Training Accuracy": trainingAccuracy,
        "Validation Accuracy": validationAccuracy,
        "Training Loss": trainingLoss,
        "Validation Loss": validationLoss
    }

    if validationLoss < currentBestValidationLoss:
        currentBestValidationLoss = validationLoss

    if trainingLoss < currentBestTrainingLoss:
        currentBestTrainingLoss = trainingLoss

    bestTrainingLoss_data.append(currentBestTrainingLoss)
    bestValidationLoss_data.append(currentBestValidationLoss)

    parameters.append(parameter_dictionary)
    counter += 1

globalRandomSearch_SGD_Parameters_df = pd.DataFrame(parameters)

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', None)
display(globalRandomSearch_SGD_Parameters_df)

testLoss_data = []
bestTestLoss_data = []
```

```
currentBestTestLoss = float("inf")
optimisedParameters = {}

counter = 1

for parameter in tqdm(globalRandomSearch_SGD_param_values):

  model = keras.Sequential(
    [
      keras.Input(shape=(28, 28, 1)),
      layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
      layers.MaxPooling2D(pool_size=(2, 2)),
      layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
      layers.MaxPooling2D(pool_size=(2, 2)),
      layers.Flatten(),
      layers.Dropout(0.5),
      layers.Dense(num_classes, activation="softmax"),
    ]
  )

  optimizer = SGD(learning_rate = parameter.hyperparameters.get('learning_rate'))

  model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])

  model.fit(x_train, y_train, validation_data=(x_validation, y_validation), batch_size = parameter.hyperparameters.get('batch_size'),
verbose=0, epochs = 10)
  testLoss, testAccuracy = model.evaluate(x_test, y_test, verbose=0)
  testLoss_data.append(testLoss)

  if testLoss < currentBestTestLoss:
    currentBestTestLoss = testLoss
    optimisedParameters = {
                "Trial": counter,
                "Learning Rate": parameter.hyperparameters.get("learning_rate"),
                "Batch Size": parameter.hyperparameters.get("batch_size"),
                "Test Accuracy": testAccuracy,
                "Test Loss": testLoss
            }
  bestTestLoss_data.append(currentBestTestLoss)

  counter += 1

globalRandomSearch_SGD_Test_Parameters_df = pd.DataFrame([optimisedParameters])
display(globalRandomSearch_SGD_Test_Parameters_df)

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(globalRandomSearch_SGD_param_values) + 1), trainingLoss_data, label="Training Loss : Global Random
Search(SGD)", marker='o', color="Red")
plot.plot(range(1, len(globalRandomSearch_SGD_param_values) + 1), validationLoss_data, label="Validation Loss : Global Random
Search(SGD)", marker='o', color="Yellow")
plot.plot(range(1, len(globalRandomSearch_SGD_param_values) + 1), testLoss_data, label="Test Loss : Global Random Search(SGD)",
marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Loss")
plot.title('Plot of Loss vs. All Trials for Global Random Search(SGD)')
plot.legend()
plot.show()

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(globalRandomSearch_SGD_param_values) + 1), bestTrainingLoss_data, label="Best Training Loss : Global Random
Search(SGD)", marker='o', color="Red")
plot.plot(range(1, len(globalRandomSearch_SGD_param_values) + 1), bestValidationLoss_data, label="Best Validation Loss : Global
Random Search(SGD)", marker='o', color="Yellow")
```

```
plot.plot(range(1, len(globalRandomSearch_SGD_param_values) + 1), bestTestLoss_data, label="Best Test Loss : Global Random Search(SGD)", marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Best Loss")
plot.title('Plot of Best Loss vs. All Trials for Global Random Search(SGD)')
plot.legend()
plot.show()

bestTest_GRS_SGD_DF = pd.DataFrame(bestTestLoss_data, columns=['BestTestLoss_GRS_SGD'])
bestTest_GRS_SGD_DF.to_csv('GlobalRandomSearch_SGD_BestTestLoss.csv', index=False)
```

Appendix: Code for Global Random Search Adam

```
import numpy as np
import keras
import tensorflow as tf
from tensorflow.keras import layers
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from tensorflow.keras.optimizers import SGD, Adam
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import train_test_split
from kerastuner import RandomSearch
import matplotlib.pyplot as plot
from keras.utils import np_utils
import pandas as pd

from IPython.display import display
import time
from tqdm import tqdm

num_classes = 10
input_shape = (28, 28, 1)

(x_train, y_train), (x_test, y_test) = mnist.load_data()

n=30000
x_train = x_train[1:n]
y_train=y_train[1:n]

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

x_train, x_validation, y_train, y_validation = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

def create_Global_Random_Adam_model(parameters):

    selectedLearningRate = parameters.Float('learning_rate', 1e-4, 1e-2, sampling='log')
    selectedBatchSize = parameters.Choice('batch_size',values= [32,64,96,128])
    selectedBeta1 = parameters.Choice('beta_1',values= [0.25,0.5,0.9])
    selectedBeta2 = parameters.Choice('beta_2',values= [0.9,0.99,0.999])

    model = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation="softmax"),
        ]
    )

    optimizer = Adam(learning_rate=selectedLearningRate, beta_1=selectedBeta1, beta_2=selectedBeta2)

    model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])

    return model
```

```
globalRandomSearch_Adam_optimiser = RandomSearch(
                create_Global_Random_Adam_model,
                objective = 'val_loss',
                seed = 42,
                max_trials=20,
                directory = 'GlobalRandomSearch_Adam',
                project_name = 'mnist_data'
            )

start_time = time.time()
globalRandomSearch_Adam_optimiser.search(x_train, y_train, validation_data = (x_validation, y_validation), epochs = 10)
end_time = time.time()
print(f'\n\nTotal time to execute Global Random Search(Adam) is : {end_time - start_time} seconds')

globalRandomSearch_Adam_param_values = globalRandomSearch_Adam_optimiser.oracle.trials.values()

trainingLoss_data = []
validationLoss_data = []
bestTrainingLoss_data = []
bestValidationLoss_data = []

currentBestTrainingLoss = currentBestValidationLoss = float("inf")

parameters = []
counter = 1

for parameter in globalRandomSearch_Adam_param_values:
   trainingLoss = parameter.metrics.get_history('loss')[-1].value[0]
   validationLoss = parameter.metrics.get_history('val_loss')[-1].value[0]
   trainingAccuracy = parameter.metrics.get_history('accuracy')[-1].value[0]
   validationAccuracy = parameter.metrics.get_history('val_accuracy')[-1].value[0]
   trainingLoss_data.append(trainingLoss)
   validationLoss_data.append(validationLoss)

   parameter_dictionary = {
      "Trial": counter,
      "Learning Rate": parameter.hyperparameters.get("learning_rate"),
      "Batch Size": parameter.hyperparameters.get("batch_size"),
      "Beta 1": parameter.hyperparameters.get("beta_1"),
      "Beta 2": parameter.hyperparameters.get("beta_2"),
      "Training Accuracy": trainingAccuracy,
      "Validation Accuracy": validationAccuracy,
      "Training Loss": trainingLoss,
      "Validation Loss": validationLoss
   }

   if validationLoss < currentBestValidationLoss:
      currentBestValidationLoss = validationLoss

   if trainingLoss < currentBestTrainingLoss:
      currentBestTrainingLoss = trainingLoss

   bestTrainingLoss_data.append(currentBestTrainingLoss)
   bestValidationLoss_data.append(currentBestValidationLoss)

   parameters.append(parameter_dictionary)
   counter += 1

globalRandomSearch_Adam_Parameters_df = pd.DataFrame(parameters)

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', None)
```

```
display(globalRandomSearch_Adam_Parameters_df)

testLoss_data = []
bestTestLoss_data = []
currentBestTestLoss = float("inf")
optimisedParameters = {}

counter = 1

for parameter in tqdm(globalRandomSearch_Adam_param_values):

  model = keras.Sequential(
    [
      keras.Input(shape=(28, 28, 1)),
      layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
      layers.MaxPooling2D(pool_size=(2, 2)),
      layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
      layers.MaxPooling2D(pool_size=(2, 2)),
      layers.Flatten(),
      layers.Dropout(0.5),
      layers.Dense(num_classes, activation="softmax"),
    ]
  )

  optimizer = Adam(learning_rate=parameter.hyperparameters.get('learning_rate'), beta_1=parameter.hyperparameters.get('beta_1'),
beta_2=parameter.hyperparameters.get('beta_2'))

  model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])

  model.fit(x_train, y_train, validation_data=(x_validation, y_validation), batch_size = parameter.hyperparameters.get('batch_size'),
epochs = 10, verbose=0)
  testLoss, testAccuracy = model.evaluate(x_test, y_test, verbose=0)
  testLoss_data.append(testLoss)

  if testLoss < currentBestTestLoss:
    currentBestTestLoss = testLoss
    optimisedParameters = {
                "Trial": counter,
                "Learning Rate": parameter.hyperparameters.get("learning_rate"),
                "Batch Size": parameter.hyperparameters.get("batch_size"),
                "Beta 1": parameter.hyperparameters.get("beta_1"),
                "Beta 2": parameter.hyperparameters.get("beta_2"),
                "Test Accuracy": testAccuracy,
                "Test Loss": testLoss
            }
  bestTestLoss_data.append(currentBestTestLoss)

  counter += 1

globalRandomSearch_Adam_Test_Parameters_df = pd.DataFrame([optimisedParameters])
display(globalRandomSearch_Adam_Test_Parameters_df)

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(globalRandomSearch_Adam_param_values) + 1), trainingLoss_data, label="Training Loss : Global Random
Search(Adam)", marker='o', color="Red")
plot.plot(range(1, len(globalRandomSearch_Adam_param_values) + 1), validationLoss_data, label="Validation Loss : Global Random
Search(Adam)", marker='o', color="Yellow")
plot.plot(range(1, len(globalRandomSearch_Adam_param_values) + 1), testLoss_data, label="Test Loss : Global Random Search(Adam)",
marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Loss")
plot.title('Plot of Loss vs. All Trials for Global Random Search(Adam)')
plot.legend()
```

```
plot.show()

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(globalRandomSearch_Adam_param_values) + 1), bestTrainingLoss_data, label="Best Training Loss : Global Random Search(Adam)", marker='o', color="Red")
plot.plot(range(1, len(globalRandomSearch_Adam_param_values) + 1), bestValidationLoss_data, label="Best Validation Loss : Global Random Search(Adam)", marker='o', color="Yellow")
plot.plot(range(1, len(globalRandomSearch_Adam_param_values) + 1), bestTestLoss_data, label="Best Test Loss : Global Random Search(Adam)", marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Best Loss")
plot.title('Plot of Best Loss vs. All Trials for Global Random Search(Adam)')
plot.legend()
plot.show()

bestTest_GRS_Adam_DF = pd.DataFrame(bestTestLoss_data, columns=['BestTestLoss_GRS_Adam'])
bestTest_GRS_Adam_DF.to_csv('GlobalRandomSearch_Adam_BestTestLoss.csv', index=False)
```

Appendix: Code for Global Random Search Comparison

```
import pandas as pd
from IPython.display import display
import matplotlib.pyplot as plot

GlobalRandomSearch_Adam_BestTestLoss_df = pd.read_csv('GlobalRandomSearch_Adam_BestTestLoss.csv')
GlobalRandomSearch_SGD_BestTestLoss_df = pd.read_csv('GlobalRandomSearch_SGD_BestTestLoss.csv')

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', None)
display(GlobalRandomSearch_Adam_BestTestLoss_df)

display(GlobalRandomSearch_SGD_BestTestLoss_df)

bestTestLoss_GRS_Adam = GlobalRandomSearch_Adam_BestTestLoss_df['BestTestLoss_GRS_Adam']
bestTestLoss_GRS_SGD = GlobalRandomSearch_SGD_BestTestLoss_df['BestTestLoss_GRS_SGD']

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(bestTestLoss_GRS_Adam) + 1), bestTestLoss_GRS_Adam, label="Best Test Loss : Global Random Search(Adam)",
marker='o', color="Red")
plot.plot(range(1, len(bestTestLoss_GRS_SGD) + 1), bestTestLoss_GRS_SGD, label="Best Test Loss : Global Random Search(SGD)",
marker='o', color="Yellow")
plot.xlabel("#Iteration")
plot.ylabel("Best Loss")
plot.title('Performance comparison of Best Loss for Global Random Search(SGD) vs. Global Random Search(Adam)')
plot.legend()
plot.show()
```

Appendix: Code for Nelder-Mead SGD

```
import numpy as np
import keras
import tensorflow as tf
from tensorflow.keras import layers
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from tensorflow.keras.optimizers import SGD, Adam
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import train_test_split
from scipy.optimize import minimize
import matplotlib.pyplot as plot
from keras.utils import np_utils
import pandas as pd

from IPython.display import display
import time
from tqdm import tqdm

num_classes = 10
input_shape = (28, 28, 1)

(x_train, y_train), (x_test, y_test) = mnist.load_data()

n=30000
x_train = x_train[1:n]
y_train=y_train[1:n]

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

x_train, x_validation, y_train, y_validation = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

def create_NelderMead_SGD_model(parameters):

    selectedLearningRate = parameters[0]
    selectedBatchSize = int(parameters[1])

    model = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation="softmax"),
        ]
    )

    optimizer = SGD(learning_rate = selectedLearningRate)

    model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])

    return model

trainingLoss_data = []
validationLoss_data = []
bestTrainingLoss_data = []
bestValidationLoss_data = []

currentBestTrainingLoss = currentBestValidationLoss = float("inf")
```

```
testLoss_data = []
bestTestLoss_data = []
currentBestTestLoss = float("inf")
optimisedParameters = {}

def tune_NelderMead_SGD_model(parameters):
   global currentBestValidationLoss
   global currentBestTrainingLoss
   global currentBestTestLoss
   global optimisedParameters

   parameters[0] = np.clip(parameters[0], 1e-4, 1e-1)
   parameters[1]  = np.clip(parameters[1], 16, 128)

   model = create_NelderMead_SGD_model(parameters)
   history = model.fit(x_train, y_train, epochs=10, batch_size=int(parameters[1]), validation_data=(x_validation, y_validation), verbose=0)
   validationLoss = history.history['val_loss'][-1]
   trainingLoss = history.history['loss'][-1]

   trainingLoss_data.append(trainingLoss)
   validationLoss_data.append(validationLoss)

   if validationLoss < currentBestValidationLoss:
      currentBestValidationLoss = validationLoss

   if trainingLoss < currentBestTrainingLoss:
      currentBestTrainingLoss = trainingLoss

   bestTrainingLoss_data.append(currentBestTrainingLoss)
   bestValidationLoss_data.append(currentBestValidationLoss)

   testLoss, testAccuracy = model.evaluate(x_test, y_test, batch_size=int(parameters[1]), verbose=0)

   testLoss_data.append(testLoss)

   if testLoss < currentBestTestLoss:
      currentBestTestLoss = testLoss
      optimisedParameters = {
                        "Learning Rate": parameters[0],
                        "Batch Size": int(parameters[1]),
                        "Test Accuracy": testAccuracy,
                        "Test Loss": testLoss
                     }
   bestTestLoss_data.append(currentBestTestLoss)
   return validationLoss

initial_parameters = [0.001, 32]
start_time = time.time()
n_iterations = 20
with tqdm(total=n_iterations-1) as pbar:
   def tqdm_callback(x):
      pbar.update(1)

   minimize(tune_NelderMead_SGD_model, x0=initial_parameters, options={'maxiter': n_iterations}, callback=tqdm_callback,
method='Nelder-Mead')
end_time = time.time()
print(f'\n\nTotal time to execute Nelder-Mead(SGD) is : {end_time - start_time} seconds')

NelderMead_Test_Parameters_df = pd.DataFrame([optimisedParameters])
display(NelderMead_Test_Parameters_df)

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(trainingLoss_data) + 1), trainingLoss_data, label="Training Loss : Nelder-Mead(SGD)", marker='o', color="Red")
plot.plot(range(1, len(validationLoss_data) + 1), validationLoss_data, label="Validation Loss : Nelder-Mead(SGD)", marker='o',
color="Yellow")
plot.plot(range(1, len(testLoss_data) + 1), testLoss_data, label="Test Loss : Nelder-Mead(SGD)", marker='o', color="Cyan")
plot.xlabel("#Iteration")
```

```
plot.ylabel("Loss")
plot.title('Plot of Loss vs. All Trials for Nelder-Mead(SGD)')
plot.legend()
plot.show()

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(bestTrainingLoss_data) + 1), bestTrainingLoss_data, label="Best Training Loss : Nelder-Mead(SGD)", marker='o', color="Red")
plot.plot(range(1, len(bestValidationLoss_data) + 1), bestValidationLoss_data, label="Best Validation Loss : Nelder-Mead(SGD)", marker='o', color="Yellow")
plot.plot(range(1, len(bestTestLoss_data) + 1), bestTestLoss_data, label="Best Test Loss : Nelder-Mead(SGD)", marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Best Loss")
plot.title('Plot of Best Loss vs. All Trials for Nelder-Mead(SGD)')
plot.legend()
plot.show()

bestTest_NM_SGD_DF = pd.DataFrame(bestTestLoss_data, columns=['BestTestLoss_NM_SGD'])
bestTest_NM_SGD_DF.to_csv('NM_SGD_BestTestLoss.csv', index=False)
```

Appendix: Code for Nelder-Mead Adam

```
import numpy as np
import keras
import tensorflow as tf
from tensorflow.keras import layers
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from tensorflow.keras.optimizers import SGD, Adam
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import train_test_split
from scipy.optimize import minimize
import matplotlib.pyplot as plot
from keras.utils import np_utils
import pandas as pd

from IPython.display import display
import time
from tqdm import tqdm

num_classes = 10
input_shape = (28, 28, 1)

(x_train, y_train), (x_test, y_test) = mnist.load_data()

n=30000
x_train = x_train[1:n]
y_train=y_train[1:n]

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

x_train, x_validation, y_train, y_validation = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

def create_NelderMead_Adam_model(parameters):

    selectedLearningRate = parameters[0]
    selectedBatchSize = int(parameters[1])
    selectedBeta1 = parameters[2]
    selectedBeta2 = parameters[3]
    model = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation="softmax"),
        ]
    )

    optimizer = Adam(learning_rate=selectedLearningRate, beta_1=selectedBeta1, beta_2=selectedBeta2)

    model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])

    return model

trainingLoss_data = []
validationLoss_data = []
bestTrainingLoss_data = []
bestValidationLoss_data = []
```

```
currentBestTrainingLoss = currentBestValidationLoss = float("inf")

testLoss_data = []
bestTestLoss_data = []
currentBestTestLoss = float("inf")
optimisedParameters = {}

def tune_NelderMead_Adam_model(parameters):
    global currentBestValidationLoss
    global currentBestTrainingLoss
    global currentBestTestLoss
    global optimisedParameters

    parameters[0] = np.clip(parameters[0], 1e-4, 1e-1)
    parameters[1] = np.clip(parameters[1], 16, 128)
    parameters[2] = np.clip(parameters[2], 0.25, 0.99)
    parameters[3] = np.clip(parameters[3], 0.9, 0.999)

    model = create_NelderMead_Adam_model(parameters)
    history = model.fit(x_train, y_train, epochs=10, batch_size=int(parameters[1]), validation_data=(x_validation, y_validation), verbose=0)
    validationLoss = history.history['val_loss'][-1]
    trainingLoss = history.history['loss'][-1]

    trainingLoss_data.append(trainingLoss)
    validationLoss_data.append(validationLoss)

    if validationLoss < currentBestValidationLoss:
        currentBestValidationLoss = validationLoss

    if trainingLoss < currentBestTrainingLoss:
        currentBestTrainingLoss = trainingLoss

    bestTrainingLoss_data.append(currentBestTrainingLoss)
    bestValidationLoss_data.append(currentBestValidationLoss)

    testLoss, testAccuracy = model.evaluate(x_test, y_test, batch_size=int(parameters[1]), verbose=0)

    testLoss_data.append(testLoss)

    if testLoss < currentBestTestLoss:
        currentBestTestLoss = testLoss
        optimisedParameters = {
                            "Learning Rate": parameters[0],
                            "Batch Size": int(parameters[1]),
                            "Beta 1": parameters[2],
                            "Beta 2": parameters[3],
                            "Test Accuracy": testAccuracy,
                            "Test Loss": testLoss
                        }
    bestTestLoss_data.append(currentBestTestLoss)
    return validationLoss

initial_parameters = [0.001, 32, 0.9, 0.999]
start_time = time.time()
n_iterations = 20
with tqdm(total=n_iterations-1) as pbar:
    def tqdm_callback(x):
        pbar.update(1)

    minimize(tune_NelderMead_Adam_model, x0=initial_parameters, options={'maxiter': n_iterations}, callback=tqdm_callback,
method='Nelder-Mead')
end_time = time.time()
print(f'\n\nTotal time to execute Nelder-Mead(Adam) is : {end_time - start_time} seconds')

NelderMead_Test_Parameters_df = pd.DataFrame([optimisedParameters])
display(NelderMead_Test_Parameters_df)

plot.figure(figsize=(9, 6), dpi=150)
```

```
plot.plot(range(1, len(trainingLoss_data) + 1), trainingLoss_data, label="Training Loss : Nelder-Mead(Adam)", marker='o', color="Red")
plot.plot(range(1, len(validationLoss_data) + 1), validationLoss_data, label="Validation Loss : Nelder-Mead(Adam)", marker='o', color="Yellow")
plot.plot(range(1, len(testLoss_data) + 1), testLoss_data, label="Test Loss : Nelder-Mead(Adam)", marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Loss")
plot.title('Plot of Loss vs. All Trials for Nelder-Mead(Adam)')
plot.legend()
plot.show()

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(bestTrainingLoss_data) + 1), bestTrainingLoss_data, label="Best Training Loss : Nelder-Mead(Adam)", marker='o', color="Red")
plot.plot(range(1, len(bestValidationLoss_data) + 1), bestValidationLoss_data, label="Best Validation Loss : Nelder-Mead(Adam)", marker='o', color="Yellow")
plot.plot(range(1, len(bestTestLoss_data) + 1), bestTestLoss_data, label="Best Test Loss : Nelder-Mead(Adam)", marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Best Loss")
plot.title('Plot of Best Loss vs. All Trials for Nelder-Mead(Adam)')
plot.legend()
plot.show()

bestTest_NM_Adam_DF = pd.DataFrame(bestTestLoss_data, columns=['BestTestLoss_NM_Adam'])
bestTest_NM_Adam_DF.to_csv('NM_Adam_BestTestLoss.csv', index=False)
```

Appendix: Code for Nelder-Mead Comparison

```
import pandas as pd
from IPython.display import display
import matplotlib.pyplot as plot

NM_Adam_BestTestLoss_df = pd.read_csv('NM_Adam_BestTestLoss.csv')
NM_SGD_BestTestLoss_df = pd.read_csv('NM_SGD_BestTestLoss.csv')

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', None)
display(NM_Adam_BestTestLoss_df)

display(NM_SGD_BestTestLoss_df)

bestTestLoss_NM_Adam = NM_Adam_BestTestLoss_df['BestTestLoss_NM_Adam']
bestTestLoss_NM_SGD = NM_SGD_BestTestLoss_df['BestTestLoss_NM_SGD']

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(bestTestLoss_NM_Adam) + 1), bestTestLoss_NM_Adam, label="Best Test Loss : Nelder-Mead(Adam)", marker='o', color="Red")
plot.plot(range(1, len(bestTestLoss_NM_SGD) + 1), bestTestLoss_NM_SGD, label="Best Test Loss : Nelder-Mead(SGD)", marker='o', color="Yellow")
plot.xlabel("#Iteration")
plot.ylabel("Best Loss")
plot.title('Performance comparison of Best Loss for Nelder-Mead(SGD) vs. Nelder-Mead(Adam)')
plot.legend()
plot.show()
```

Appendix: Code for All Comparison

```
import pandas as pd
from IPython.display import display
import matplotlib.pyplot as plot

NM_Adam_BestTestLoss_df = pd.read_csv('NM_Adam_BestTestLoss.csv')
GS_Adam_BestTestLoss_df = pd.read_csv('GridSearch_Adam_BestTestLoss.csv')
GRS_Adam_BestTestLoss_df = pd.read_csv('GlobalRandomSearch_Adam_BestTestLoss.csv')

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', None)
display(NM_Adam_BestTestLoss_df)

display(GS_Adam_BestTestLoss_df)

display(GRS_Adam_BestTestLoss_df)

bestTestLoss_NM_Adam = NM_Adam_BestTestLoss_df['BestTestLoss_NM_Adam']
bestTestLoss_GS_Adam = GS_Adam_BestTestLoss_df['BestTestLoss_GS_Adam']
bestTestLoss_GRS_Adam = GRS_Adam_BestTestLoss_df['BestTestLoss_GRS_Adam']

plot.figure(figsize=(9, 6), dpi=150)
plot.plot(range(1, len(bestTestLoss_NM_Adam) + 1), bestTestLoss_NM_Adam, label="Best Test Loss : Nelder-Mead(Adam)", marker='o', color="Red")
plot.plot(range(1, len(bestTestLoss_GRS_Adam) + 1), bestTestLoss_GRS_Adam, label="Best Test Loss : Global Random Search(Adam)", marker='o', color="Yellow")
plot.plot(range(1, len(bestTestLoss_GS_Adam) + 1), bestTestLoss_GS_Adam, label="Best Test Loss : Grid Search(Adam)-Baseline", marker='o', color="Cyan")
plot.xlabel("#Iteration")
plot.ylabel("Best Loss")
plot.title('Performance comparison of Best Loss for Nelder-Mead(Adam) vs. Global Random Search(Adam) vs. Grid Search(Adam)-Baseline')
plot.legend()
plot.show()
```