Deep Learning Assignment - 4
Report

Question 1.

Link of code (Link)

a) **Given Paper** : **“ Neural Architecture Search without Training “**
   Dataset details : **CIFAR 10**

Reference code address :

1) https://github.com/BayesWatch/nas-without-training.git

Cloning the Git codes in google colab :

1) https://github.com/google-research/nasbench
2) https://github.com/facebookresearch/nds

and place the json files in naswot-codebase/nds_data/ Download the NASbench101 data

https://github.com/google-research/nasbench

Note : The tensorflow version which was used in the reference code is the older version i.e. **tensorflow_version 1.x** but right now it is outdated for the latest version of the google colab which is using the Python version 3.9 .

So for the above reason I was not able to reproduce the result of the paper completely ,but I did as much as possible taking beside the version of the tensorflow.

As we are instructed to write down our comprehension of the algorithm and codeflow in the report.

b) Code flow with the different Algorithm used in the Implementation:

- We are loading the NASBench library and the dataset for the implementation with the tensorflow version 1.x
- Importing the different standard libraries for the implementation.
- Establish a set of potential neural architectures that may be searched for using directed acyclic graphs (DAGs), where nodes stand for operations (such convolution and pooling) and edges for connections between operations.
- The weights of the edges in the DAGs, or the architectural weights, should be randomly initialized.
- The correctness of each candidate design should be assessed by generating a surrogate loss function, which is a differentiable approximation of the actual validation loss.
- Identify the gradients of the surrogate loss function with respect to the architectural weights.
- The accuracy of the candidate architectures can be increased by updating the architectural weights using the gradients.
- Steps 3-5 should be repeated indefinitely or until a convergence requirement is satisfied.
- Based on its correctness on a held-out validation set, choose the optimum architecture.
- To create the final model, completely train the chosen architecture on the training set.

Here is the code flow for implementing the DARTS-WT method:

- Create a list of input and output nodes, a set of primitive operations (such convolution and pooling), and the search space of potential neural networks.
- Utilize the basic operations and the architectural weights to implement the surrogate loss function.
- Create a method to calculate the surrogate loss function's gradients in relation to the architectural weights.
- Implement an optimizer to update the architectural weights using the calculated gradients, such as stochastic gradient descent.
- Iterate the following stages until convergence after randomly initializing the architectural weights:

1) Select a sample of potential architectural designs from the search space.
2) Use the surrogate loss function to calculate each candidate architecture's accuracy.
3) Determine the surrogate loss function's gradients in relation to the architectural weights.
4) Update the architecture weights using the gradients and the optimizer.

- Based on its correctness on a held-out validation set, choose the optimum architecture.
- To create the final model, completely train the chosen architecture on the training set.

c) We are choosing the Pets dataset for this task :

By selecting an appropriate search space of neural architectures and constructing a surrogate loss function that simulates the actual validation loss, the DARTS-WT approach may be used to analyze any image classification dataset. The approach can assist in automating the search for neural architectures and lower the computational expense associated with training several candidate structures.

d) Improvements for the above algorithm :

- The DARTS-WT approach can be improved by using a more complex search strategy that can more thoroughly explore the search space of potential neural architectures. More specifically, we may produce and assess a wide range of candidate structures by combining evolutionary search with random search.

- Architectures from the search space are randomly selected for analysis, and their performance is assessed against a validation set. We may explore a greater area of the search space by sampling a large number of random structures, and perhaps find high-performing architectures that a more concentrated search approach might miss.

- In evolutionary search, new designs are iteratively created by merging or modifying old architectures, and the best performing architectures are chosen for the following generation. We may direct the evolutionary search towards more promising areas of the search space by integrating a fitness function that assesses the performance of each design on a validation set.

- We can begin by randomly selecting a large number of potential candidate architectures and assessing each one's performance on the validation set before combining these search strategies. Then, utilizing genetic operators like crossover and mutation, we may repeatedly create new designs using the best-performing structures as a starting population for the evolutionary search. We may assess how well new designs perform on the validation set and choose the top architectures for the next generation.

- Comparing the performance of the hybrid DARTS-WT method with the original DARTS-WT method on the Pets dataset allows us to assess the efficacy of this hybrid search methodology. We may assess the test and validation sets' classification accuracy as well as other pertinent metrics.

Mathematical Support :

The mathematical combination of random search and evolutionary search can increase the efficacy and efficiency of the neural architecture search process by investigating a greater area of the search space and directing the search towards more promising locations. We can find high-performing designs with fewer evaluations and lower the computational cost of the search process by combining a varied collection of candidate architectures and choosing the best-performing structures using a fitness function.