CSL7540 - Course Project

IIT Jodhpur

October 2021

1 Important Instructions

1. You may work in teams of maximum three or by yourself. If you are short of partners, our recommendation is that this assignment is quite straightforward and a partner should not be required.
2. You cannot use built-in libraries/implementations for search or scheduling algorithms.
3. You are allowed to use the Python programming language.
4. Please do not search the Web for solutions to the problem. Your submission will be checked for plagiarism with the codes available on Web as well as the codes submitted by other teams. Any team found guilty will be awarded a suitable penalty as per IIT rules.

2 What to submit

In google classroom submit a zip file named in the format `<RollNo>.zip`. If there are two members in your team it should be called `<RollNo1_RollNo2>.zip` and so on.

Your zip file should contain following:

- `<RollNo1_RollNo2>.ipynb`: Google Colab notebook containing your code. You can copy this notebook and use the basic code provided here: https://colab.research.google.com/drive/1FhB8iE0tEXodqL3ZW9y-mA0T_2H1pn3E?usp=sharing
- `<RollNo1_RollNo2>.pdf`: 2 page write-up containing details about the search strategy you followed. Details about heuristics (if designed any) etc. It should also describe the best architecture found by your algorithm and its test accuracy and parameter count. The report should follow this IEEE format: https://www.overleaf.com/latex/templates/ieee-conference-template/grfzhhncsfqn

Note:

1. At both the places one submission per group is required.

2. Fill your group details in the following google sheet: https://docs.google.com/spreadsheets/d/1QIj4CaVqc7lvUx2jIGtX6qQsuN2eUUOikN8nbCRepdM/edit?usp=sharing

You can follow any published research papers on Neural Architecture Search (NAS) and your implementation can include concepts used in published papers **as long as you include some modification in the paper's approach and explain such modifications in your report.**

3 Problem Statement

Your task is to create a search algorithm to search in a Neural Architecture Search (NAS) space for the best performing Convolutional Neural Network (CNN) architecture on the `fashion-mnist` data-set. Best performing is defined as having a test accuracy **above 75%** and the lowest parameter count model your algorithm can find. You will encounter a trade-off between test accuracy and parameter count. Your search algorithm should balance this trade-off and you should be able to justify your choices regarding this balancing.

Since the space of all possible neural networks is very large, there are some constraints on the CNN architecture that your search function should return.

3.1 CNN Architecture Constraints

Your search algorithm should search for models with the following basic structure. Sequential model (no skip connections) having the following layers,

- Any number of normal(NC) CNN layers
  - `torch.nn.Conv2d` or `tf.keras.layers.Conv2D`
  - A normal CNN layer has the following constraints,
    - `stride=1`
    - `padding=same`
    - `1 <= kernel size < 8`
    - Any of the following activation functions,
      - relu
      - sigmoid
      - tanh
      - swish
      - gelu
- **Exactly** 2 reduction(RC) CNN layers
  - `torch.nn.Conv2d` or `tf.keras.layers.Conv2D`
  - A reduction layer has the **same** constraints as the normal layer except the following,
    - `stride=2`
    - `padding=valid`

- **Exactly** the following structure as the **final layers**,
  - First, a Global Average Pooling 2D layer.
    - `torch.nn.AvgPool2d(kernel_size=layer_input_image_size)` or `tf.keras.layers.GlobalAveragePooling2D`
  - Then, a Fully Connected/Dense layer with 64 units (such that output is shape `(batch_size, 10)`).
    - This layer can have any of the allowed activations in NC layer but all final layers must use same activation.
    - `torch.nn.Linear(64)` or `tf.keras.layers.Dense(64)`
  - Then, a Fully Connected/Dense layer with 10 units (such that output is shape `(batch_size, 10)`)
    - This layer can have any of the allowed activations in NC layer but all final layers must use same activation.
    - `torch.nn.Linear(10)` or `tf.keras.layers.Dense(10)`

The normal(NC) and reduction(RC) layers can be in **any order** as long as the constraints above are maintained.

The final layers of you model should be exactly as mentioned in these constraints.

All of the above constraints **apply only to your final output**.

Once your algorithm finds the best architecture, it should represent the architecture as a genome string (explained below) which will be fed to the training function given below to get the final accuracy.

To make your work simpler, you are provided with a *genome* based search space along with neural network model creation, training and testing code. You can use the provided code to measure the *test accuracy*, *train accuracy* and *parameter count* of any genome you choose.

3.2 Genome String Format

A model genome is a string having `;` separated parts where each part describes a specific layer.

For ex., this cell genome has 1 normal convolution layer followed by 2 reduction convolution layers and `relu` activation in the final layer.

```
NC 10 2 sigmoid;RC 10 3 relu;RC 10 3 relu;FL relu;
```

In the above example the normal layer has 10 CNN filters having a kernel size of 2 and `sigmoid` activation. After the normal layer it has 2 reduction layers with 10 filters, kernel size of 3 and `relu` activation function. This example

therefore has 3 internal layers. Finally, the genome specifies `relu` activation for the final layers(FL).

The format for each genome part for NC and RC layers is as follows,

`<NC or RC> <no. of CNN filters> <kernel size LESS THAN 8> <activation function>;`

After all NC and RC genome parts the final layer genome part has the following format,

`FL <activation function>;`

With the above, the full genome format is,

`<any number of FC/RC genome parts>;FL <activation function>;`

Here,

- NC is normal genome and RC is reduction genome. Normal genomes keep the dimensions of the input image intact (i.e. stride=1), while reduction genomes make the output image dimension half of the input dimension.
  - See the constraints section 3.1 above for specific details.
- Every genome **MUST** have **EXACTLY 2** RC parts. These RC parts can be placed anywhere in the genome.

4 Learning Resources

This section explains the basic idea behind CNN and NAS, and it also provides links to resources that you can use to learn about these concepts.

4.1 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) performs *convolution* operations on an input image to produce an output image of the same or smaller dimensions. A convolution operation uses a *convolution kernel/filter* which has a width and height ‘¡=‘ to the dimensions of the input. This filter slides on the input and at each position, its elements are multiplied with the corresponding aligning elements of the input and the results are all added to get the corresponding pixel value in the output. This was an explanation for a 2D convolution but a convolution can be performed on any N dimensional input. For this project, your model will only be using 2D convolutions.

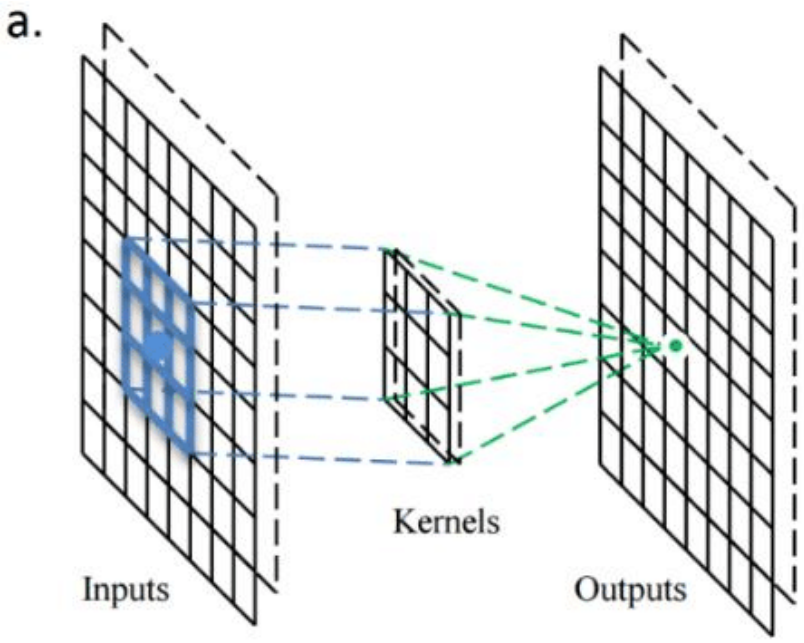

Figure 1: Figure from [1].

Each filter acts on all the input channels to produce **one** output channel. Thus, by using multiple filters on the same input one can obtain any number of channels in the output. This is the basic idea behind a CNN.

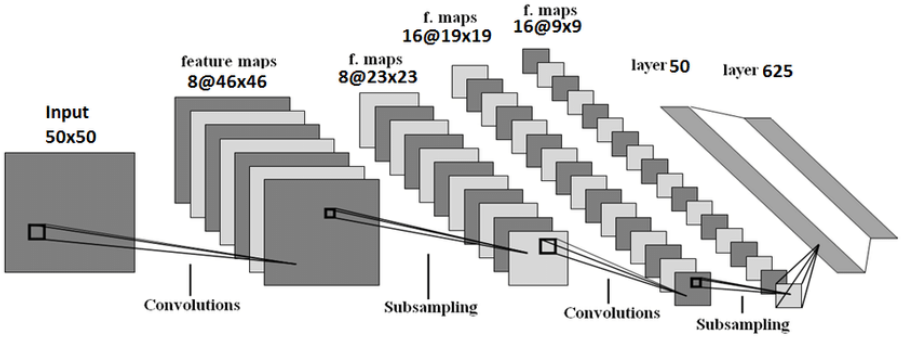

Figure 2: Figure from [1].

A CNN generally tries to transform a $W \times H \times C_{in}$ input tensor to a $1 \times 1 \times C_{out}$ tensor which is then fed to a *linear layer* for further transformation to the desired output shape. The *learning* in a CNN happens by updating the filters in each layer during *gradient back-propagation*.

See:

1. https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7
2. https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add

4.2 Neural Architecture Search (NAS)

Usually a neural network architecture is manually designed by an expert to give high accuracy on a target data-set. In recent development however, people have been trying to automate this architecture designing process such that given a data-set and target constraints, a computer can automatically determine the best performing neural network architecture for the task. This project requires you to develop such an algorithm.

There have been many different approaches proposed in literature to create such a Neural Architecture Search (NAS) algorithm. Initially, many approaches used Reinforcement Learning (RL) to learn a *controller* that generates an network architecture and learns on the feedback about its performance on the data-set. Some also used evolutionary algorithms to improve the performance of a population of algorithms. Recent works focus on parameter sharing and super-network training approaches to counter the large training time required for profiling performance of each architecture in the search space.

For this project, the data-set is simple enough that training time for individual architectures is small. The search space is also deliberately kept very simple to match the simplicity of the data-set. You can use the code provided for approaches like RL, evolutionary search etc. However, if you wish to, you can delve into parameter sharing and super-network style approaches **as long as the final architecture your algorithm provides fits the constraints above**.

See,

1. https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html
2. https://towardsdatascience.com/everything-you-need-to-know-about-automl-and-neural-architecture-search-8db1863682bf1

References

[1] I. Poletaev, Konstantin Pervunin, and M. Tokarev. "Artificial neural network for bubbles pattern recognition on the images". In: *Journal of Physics Conference Series* 754 (Oct. 2016), (072002)–13. DOI: 10.1088/1742-6596/754/7/072002.