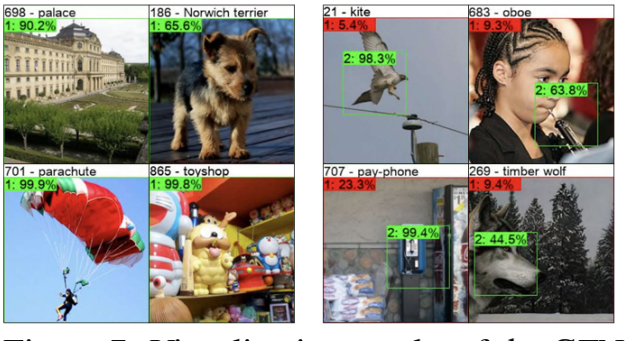
DL End-Sem Exam Part-1

By:- Kwanit Gupta
B19EE046

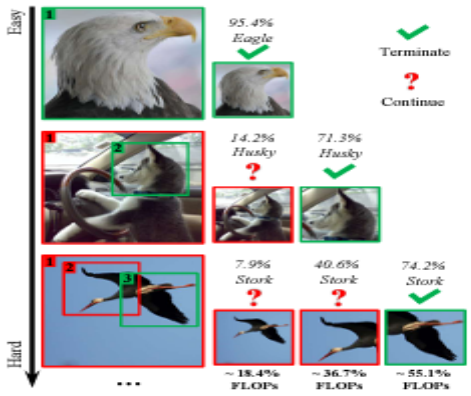
For describing the whole situation in **"Glance and Focus: a Dynamic Approach to Reducing Spatial Redundancy in Image Classification", the** following would be the required explanatory portions:-

- ***Main Focus of the Paper:-***



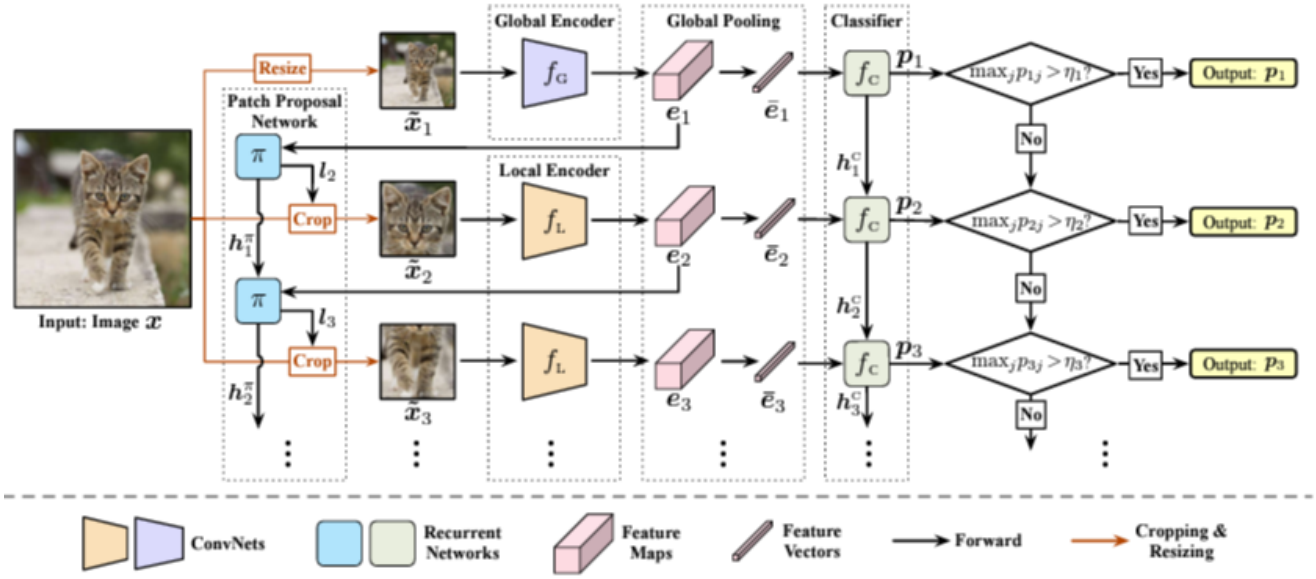
The authors aim to decrease the computational cost incurred by high-resolution inputs in classification tasks, by offering a unique framework that conducts effective image classification by processing a succession of relatively tiny patches that are carefully picked from the original source image, using reinforcement learning, inspired by the fact that not all areas in an image are task-relevant. Such a dynamic decision procedure naturally enables adaptive inference at test time, since it may be ended once the model is sufficiently confident in its prediction, avoiding additional unnecessary computation.

- ***Motivation behind the Approach:-***



For most of the Computer Vision-based Application Areas (specifically Object Detection, Segmentation, or Classification Tasks), the Region of Interest plays a key role in defining the utmost purpose of the implementation. Although there exist many advanced Deep Learning Architectures like Transformers, Autoencoders, GANs, etc. it isn't necessary that each and every spatial information of a vision-based modality would be that significant to the task and the above-mentioned models would definitely provide the justified metrics on it.

- ***Key Contributions of the Paper:-***



- The region selection operation is specifically constructed as a sequential decision process, in which our model analyses a relatively modest input at each stage, generating a classification prediction with a confidence value as well as a region recommendation for the following phase.

- Because the suggested region is often a small piece of the original picture with full resolution, we refer to the next phases as the focus stage. This step works its way through the picture iteratively by localizing and processing the class-discriminative picture areas, allowing for adaptive early termination.

● ***Related Literature:-***

The 3 main aspects associated with this research work, were showcased as follows:-

1. Architecture Design:-

Some of the recent CNNs (Like MobileNet, EfficientNet, etc.) become contemporary with traditional components (like Convolutions, Pooling, etc.) for attaining performance optimization as well as efficiently reducing the runtime and computational memory. Apart from these, some focused on quantizing weights, pruning the layers, masking the connections, ensembling the sub-networks, etc.

2. Attention support via Reinforcement Learning:-

$$L_t^{\text{CLIP}} = \min\left\{\frac{\pi(l_t|s_t)}{\pi_{\text{old}}(l_t|s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi(l_t|s_t)}{\pi_{\text{old}}(l_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right\},$$

While most of the Vision-based models grab the context of the input via contours and contrast variations, some even try to generate a focused heatmap on decisive sub-patches. For this particular work, the authors made use of an Agent (Patch Proposition Network) with Proximal Policy Optimization (PPO), alongside the necessary clipped policy gradient method for generating the reward (Higher Confidence Percentage) with limited actions performed. It can be with the Network Approach as well as Statistical Perspective, with an aim of maximizing the discounted reward.

$$\underset{\pi}{\text{maximize}} \; \mathbb{E}_{\mathbf{x},t}\left[L_t^{\text{CLIP}} - c_1 L_t^{\text{VF}} + c_2 S_\pi(s_t)\right].$$

3. **Spatial Context Refinement:-**

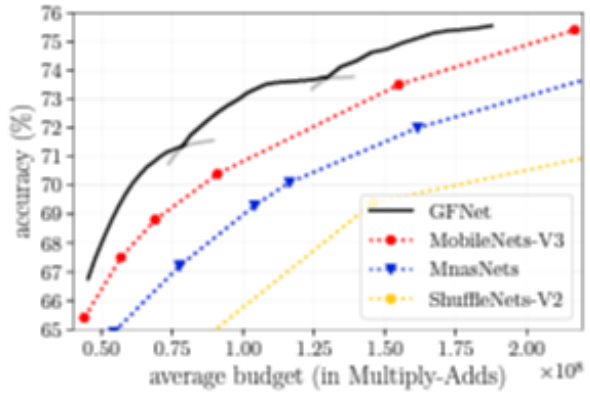
Recent research has shown that there is significant spatial redundancy when inferring CNNs. Several ways to reduce superfluous computing in the spatial dimension have been presented. The Spatially Adaptive Computation Time (SACT) dynamically adapts the number of layers run for various picture areas. By using low-frequency characteristics, the OctConv lowers the spatial resolution. These efforts mostly minimize spatial redundancy by changing convolutional layers, whereas the authors suggest processing the picture sequentially.

==========

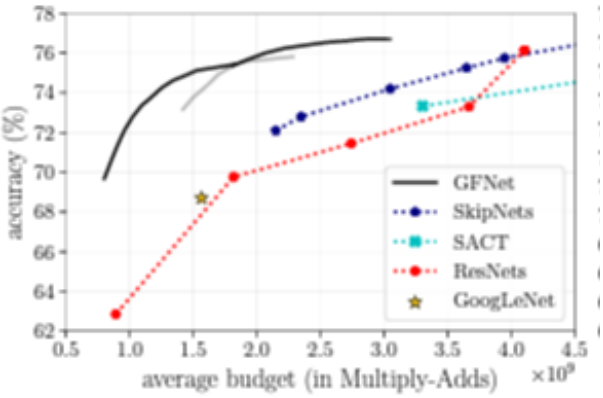
Reproduced Results on the given benchmarks

Since the authors of Github's implementation **"Glance and Focus: a Dynamic Approach to Reducing Spatial Redundancy in Image Classification"** only uploaded the initialization and stage-wise checkpoints of "ResNet50" and "MobileNetV3-Large", thereby the results on the newer benchmark were calculated in consideration of that.

In the context of the reproduction of results with the original benchmark (ImageNet), Top-1 Accuracy v/s FLOPS (Multiply-Adds Budget) was portrayed as follows:-



(a) MobileNet-V3



(d) ResNet

With respect to the evaluation run from the Github's implementation, the non-graphical representation constitutes as follows:-

```
model_arch : resnet50
patch_size : 96
flops : [761186240, 1568511872, 2375837504, 3183163136, 3990488768]
model_flops : 750725056
policy_flops : 46139392
fc_flops : 10461184
anytime_classification : [68.85, 73.708, 74.654, 75.338, 75.934]
```

```
model_arch : densenet121
patch_size : 96
flops : [527729504, 1082724032, 1637718560, 2192713088, 2747707616]
model_flops : 520414048
policy_flops : 27265024
fc_flops : 7315456
anytime_classification : [68.212, 72.994, 74.23, 74.7, 74.818]
```

==========

Reproduced Results on “mini-imagenet 200”

Since the authors of Github's implementation **“**[Glance and Focus: a Dynamic Approach to Reducing Spatial Redundancy in Image Classification](#)**”** only uploaded the initialization and stage-wise checkpoints of “ResNet50” and “MobileNetV3-Large”, thereby the results on the newer benchmark were calculated in consideration of that.

For ResNet50, the following were respective screenshots of Stage-Wise Evaluation Runs:-

- ***Stage-1 Training***

```
Training Stage: 1, lr:
0.01
0.01
0.1
Traceback (most recent call last):
  File "train.py", line 512, in <module>
    main()
  File "train.py", line 224, in main
    args.print_freq, epoch, train_configuration['batch_size'], record_file, train_configuration, args)
  File "train.py", line 331, in train
    output, state = model(patches)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/parallel/data_parallel.py", line 165, in forward
    return self.module(*inputs[0], **kwargs[0])
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/content/drive/MyDrive/EndSem_DL/GFNet-Pytorch/models/resnet.py", line 175, in forward
    x = self.layer1(x)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/container.py", line 119, in forward
    input = module(input)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/content/drive/MyDrive/EndSem_DL/GFNet-Pytorch/models/resnet.py", line 98, in forward
    out = self.bn3(out)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/batchnorm.py", line 140, in forward
    self.weight, self.bias, bn_training, exponential_average_factor, self.eps)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py", line 2150, in batch_norm
    input, weight, bias, running_mean, running_var, training, momentum, eps, torch.backends.cudnn.enabled
RuntimeError: CUDA out of memory. Tried to allocate 144.00 MiB (GPU 0; 14.76 GiB total capacity; 13.10 GiB
```

- ***Stage-2 Training***

```
Epoch: [8][180/391]     Time 1.196 (1.301)      Loss 10.9844 (11.0302)
accuracy of each step:
[0.04774305555555555, 0.045572916666666664, 0.03689236111111111, 0.043402777777777776, 0.0390625]
reward of each step:
[tensor(-1.4211e-05, device='cuda:0'), tensor(-3.4170e-07, device='cuda:0'), tensor(7.1051e-06, device='cuda:0'), tensor(-1.5701e-06, device='cuda:0')
Epoch: [8][190/391]     Time 1.220 (1.296)      Loss 11.3202 (11.0311)
accuracy of each step:
[0.045230263157894474, 0.04317434210526316, 0.03495065789473684, 0.04111842105263158, 0.03700657894736842]
reward of each step:
[tensor(-1.3425e-05, device='cuda:0'), tensor(1.9199e-07, device='cuda:0'), tensor(6.9730e-06, device='cuda:0'), tensor(-1.6025e-06, device='cuda:0')]
Epoch: [8][200/391]     Time 1.192 (1.291)      Loss 10.8666 (11.0290)
accuracy of each step:
[0.044921875, 0.04296875, 0.03515625, 0.041015625, 0.037109375]
reward of each step:
[tensor(-1.4094e-05, device='cuda:0'), tensor(-1.6642e-07, device='cuda:0'), tensor(6.5160e-06, device='cuda:0'), tensor(-8.2100e-07, device='cuda:0')
Epoch: [8][210/391]     Time 1.200 (1.287)      Loss 11.1691 (11.0320)
accuracy of each step:
[0.042782738095238096, 0.04092261904761905, 0.033482142857142856, 0.0390625, 0.035342261904761904]
reward of each step:
[tensor(-1.2839e-05, device='cuda:0'), tensor(-2.4996e-07, device='cuda:0'), tensor(6.7033e-06, device='cuda:0'), tensor(1.1719e-07, device='cuda:0')]
```

- ***Stage-3 Training***

```
Training Stage: 3, lr:
0.01
0.01
0.01
Traceback (most recent call last):
  File "train.py", line 512, in <module>
    main()
  File "train.py", line 224, in main
    args.print_freq, epoch, train_configuration['batch_size'], record_file, train_configuration, args)
  File "train.py", line 331, in train
    output, state = model(patches)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/parallel/data_parallel.py", line 165, in forward
    return self.module(*inputs[0], **kwargs[0])
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/content/drive/MyDrive/EndSem_DL/GFNet-Pytorch/models/resnet.py", line 175, in forward
    x = self.layer1(x)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/container.py", line 119, in forward
    input = module(input)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/content/drive/MyDrive/EndSem_DL/GFNet-Pytorch/models/resnet.py", line 101, in forward
    identity = self.downsample(x)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/container.py", line 119, in forward
    input = module(input)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py", line 399, in forward
    return self._conv_forward(input, self.weight, self.bias)
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/conv.py", line 396, in _conv_forward
    self.padding, self.dilation, self.groups)
RuntimeError: CUDA out of memory. Tried to allocate 144.00 MiB (GPU 0; 14.76 GiB total capacity; 13.06 GiB
```

For MobileNetV3-Large, the following were respective screenshots of Stage-Wise Evaluation Runs:-

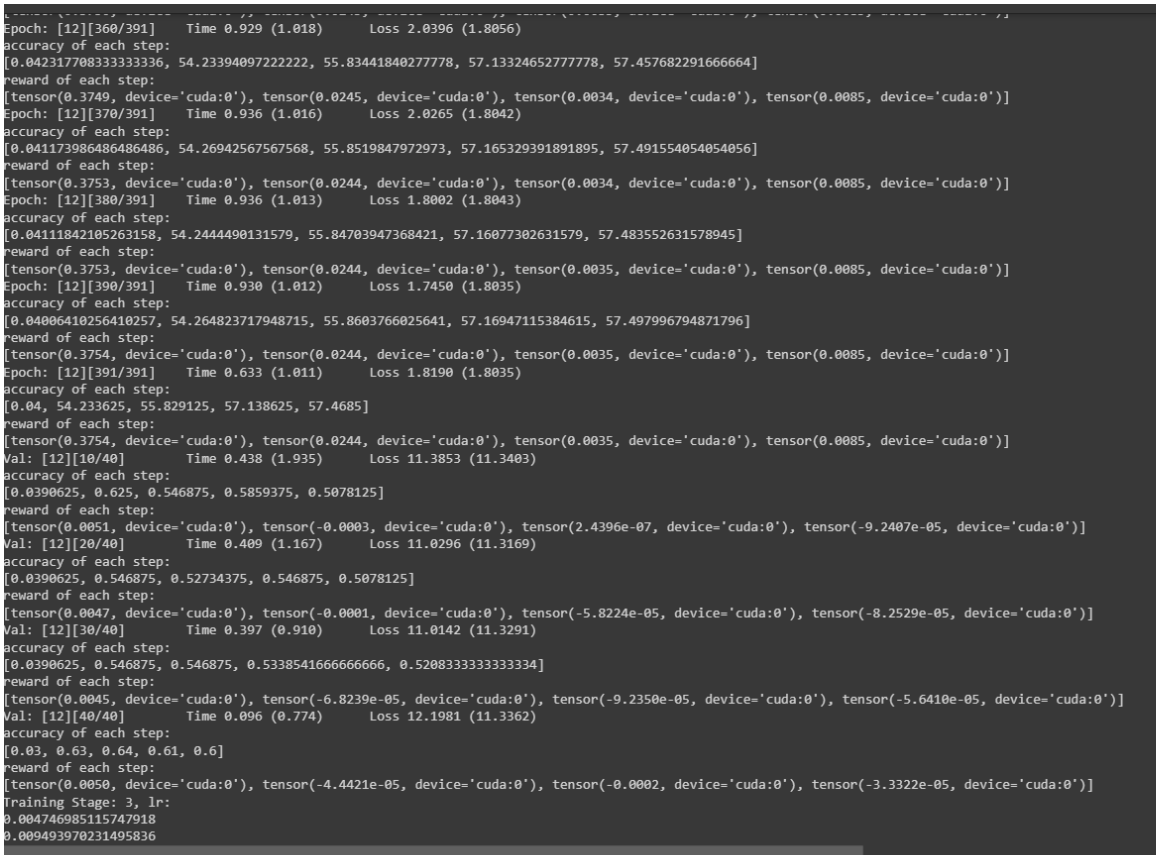
- ***Stage-1 Training***

```
[0.0537109375, 59.892578125, 60.419921875, 60.595703125, 60.4736328125]
reward of each step:
[tensor(0.4554, device='cuda:0'), tensor(0.0104, device='cuda:0'), tensor(0.0069, device='cuda:0'), tensor(0.0053, device='cuda:0')]
Epoch: [10][90/391]     Time 0.936 (1.148)      Loss 1.6674 (1.6024)
accuracy of each step:
[0.04774305555555555, 59.76996527777778, 60.31684027777778, 60.54253472222222, 60.43836805555556]
reward of each step:
[tensor(0.4549, device='cuda:0'), tensor(0.0105, device='cuda:0'), tensor(0.0069, device='cuda:0'), tensor(0.0053, device='cuda:0')]
Epoch: [10][100/391]    Time 0.920 (1.127)      Loss 1.7021 (1.6012)
accuracy of each step:
[0.05078125, 59.87109375, 60.44140625, 60.6796875, 60.5625]
reward of each step:
[tensor(0.4559, device='cuda:0'), tensor(0.0105, device='cuda:0'), tensor(0.0070, device='cuda:0'), tensor(0.0052, device='cuda:0')]
Epoch: [10][110/391]    Time 0.919 (1.109)      Loss 1.5489 (1.6021)
accuracy of each step:
[0.04971590909090909, 59.90056818181818, 60.4723011363636363, 60.67826704545455, 60.58948863636363]
reward of each step:
[tensor(0.4559, device='cuda:0'), tensor(0.0105, device='cuda:0'), tensor(0.0071, device='cuda:0'), tensor(0.0051, device='cuda:0')]
Epoch: [10][120/391]    Time 0.930 (1.095)      Loss 1.8898 (1.6073)
accuracy of each step:
[0.048828125, 59.850260416666664, 60.44921875, 60.631510416666664, 60.611979166666664]
reward of each step:
[tensor(0.4555, device='cuda:0'), tensor(0.0102, device='cuda:0'), tensor(0.0071, device='cuda:0'), tensor(0.0051, device='cuda:0')]
Epoch: [10][130/391]    Time 0.934 (1.083)      Loss 1.6157 (1.6073)
accuracy of each step:
[0.045072115384615384, 59.85276442307692, 60.49879807692308, 60.67007211538461, 60.64302884615385]
reward of each step:
[tensor(0.4558, device='cuda:0'), tensor(0.0103, device='cuda:0'), tensor(0.0071, device='cuda:0'), tensor(0.0050, device='cuda:0')]
Epoch: [10][140/391]    Time 0.940 (1.073)      Loss 1.5905 (1.6055)
accuracy of each step:
[0.044642857142857144, 59.907924107142854, 60.535714285714285, 60.689174107142854, 60.68080357142857]
reward of each step:
[tensor(0.4551, device='cuda:0'), tensor(0.0104, device='cuda:0'), tensor(0.0070, device='cuda:0'), tensor(0.0052, device='cuda:0')]
Epoch: [10][150/391]    Time 0.938 (1.063)      Loss 1.4546 (1.6074)
accuracy of each step:
[0.041666666666666664, 59.859375, 60.484375, 60.6171875, 60.658854166666664]
reward of each step:
[tensor(0.4548, device='cuda:0'), tensor(0.0104, device='cuda:0'), tensor(0.0070, device='cuda:0'), tensor(0.0052, device='cuda:0')]
Epoch: [10][160/391]    Time 0.938 (1.055)      Loss 1.6196 (1.6098)
accuracy of each step:
[0.04638671875, 59.833984375, 60.4833984375, 60.60546875, 60.6494140625]
reward of each step:
[tensor(0.4546, device='cuda:0'), tensor(0.0105, device='cuda:0'), tensor(0.0069, device='cuda:0'), tensor(0.0052, device='cuda:0')]
Epoch: [10][170/391]    Time 0.940 (1.049)      Loss 1.8061 (1.6137)
accuracy of each step:
[0.043658088235294115, 59.71047794117647, 60.372242647058826, 60.526194852941174, 60.528492647058826]
reward of each step:
[tensor(0.4538, device='cuda:0'), tensor(0.0104, device='cuda:0'), tensor(0.0069, device='cuda:0'), tensor(0.0051, device='cuda:0')]
```

- ***Stage-2 Training***

```
Epoch: [10][80/391]     Time 0.519 (0.777)      Loss 6.9090 (6.9421)
accuracy of each step:
[0.029296875, 0.0390625, 0.0390625, 0.15625, 0.087890625]
reward of each step:
[tensor(6.3021e-05, device='cuda:0'), tensor(-9.3736e-06, device='cuda:0'), tensor(0.0004, device='cuda:0'), tensor(-5.2707e-06, device='cuda:0')]
Epoch: [10][90/391]     Time 0.538 (0.761)      Loss 6.9596 (6.9426)
accuracy of each step:
[0.030381944444444444, 0.0390625, 0.034722222222222224, 0.1779513888888889, 0.08680555555555555]
reward of each step:
[tensor(5.5974e-05, device='cuda:0'), tensor(-9.5728e-06, device='cuda:0'), tensor(0.0004, device='cuda:0'), tensor(-5.1854e-06, device='cuda:0')]
Epoch: [10][100/391]    Time 0.492 (0.738)      Loss 6.9097 (6.9431)
accuracy of each step:
[0.03125, 0.0390625, 0.03515625, 0.171875, 0.09765625]
reward of each step:
[tensor(5.9320e-05, device='cuda:0'), tensor(-1.3073e-05, device='cuda:0'), tensor(0.0004, device='cuda:0'), tensor(-6.9654e-06, device='cuda:0')]
Epoch: [10][110/391]    Time 0.588 (0.723)      Loss 6.9612 (6.9431)
accuracy of each step:
[0.028409090909090908, 0.0390625, 0.03196022727272727, 0.1669034090909091, 0.08877840909090909]
reward of each step:
[tensor(6.4908e-05, device='cuda:0'), tensor(-1.3132e-05, device='cuda:0'), tensor(0.0004, device='cuda:0'), tensor(-6.8120e-06, device='cuda:0')]
Epoch: [10][120/391]    Time 0.517 (0.711)      Loss 6.9745 (6.9423)
accuracy of each step:
[0.029296875, 0.0390625, 0.029296875, 0.16927083333333334, 0.087890625]
reward of each step:
[tensor(7.4472e-05, device='cuda:0'), tensor(-1.3457e-05, device='cuda:0'), tensor(0.0004, device='cuda:0'), tensor(-5.5318e-06, device='cuda:0')]
Epoch: [10][130/391]    Time 0.535 (0.697)      Loss 6.9664 (6.9427)
accuracy of each step:
[0.027043269230769232, 0.036057692307692304, 0.030048076923076924, 0.1622596153846154, 0.08713942307692307]
reward of each step:
[tensor(7.5581e-05, device='cuda:0'), tensor(-1.2363e-05, device='cuda:0'), tensor(0.0004, device='cuda:0'), tensor(-6.0414e-06, device='cuda:0')]
Epoch: [10][140/391]    Time 0.570 (0.687)      Loss 6.9296 (6.9426)
accuracy of each step:
[0.030691964285714284, 0.0390625, 0.03627232142857143, 0.16741071428571427, 0.08928571428571429]
reward of each step:
[tensor(7.0818e-05, device='cuda:0'), tensor(-6.4672e-06, device='cuda:0'), tensor(0.0004, device='cuda:0'), tensor(-6.1701e-06, device='cuda:0')]
Epoch: [10][150/391]    Time 0.564 (0.680)      Loss 6.9012 (6.9421)
accuracy of each step:
[0.03125, 0.0390625, 0.033854166666666664, 0.16666666666666666, 0.08854166666666667]
reward of each step:
[tensor(7.3703e-05, device='cuda:0'), tensor(-2.4795e-06, device='cuda:0'), tensor(0.0004, device='cuda:0'), tensor(-6.5082e-06, device='cuda:0')]
Epoch: [10][160/391]    Time 0.553 (0.672)      Loss 6.9419 (6.9418)
accuracy of each step:
[0.029296875, 0.03662109375, 0.0341796875, 0.16845703125, 0.0830078125]
reward of each step:
[tensor(7.4997e-05, device='cuda:0'), tensor(-2.4424e-06, device='cuda:0'), tensor(0.0004, device='cuda:0'), tensor(-5.7493e-06, device='cuda:0')]
```

- ***Stage-3 Training***



The discrepancy in the results was caused due to computational constraints (Non-Responsive DGX Server and Lower GPU Configurations for Google Colab) as well as Time Limitations. Also, with the heavier architectures and detailed pipeline, even fine-tuning and optimizations with hardware restrictions led to CUDA-Memory crashing out each time.

With respect to the paper, the accuracy discrepancies ranged from 20% to even 50-60% reduced output, with the given GPU Configurations and Limited Colab Version.

==========

References

- “Glance and Focus: a Dynamic Approach to Reducing Spatial Redundancy in Image Classification” Research Paper