1. Introduction

This repository describes and contains unoffical implementation of some of the experiments in the paper "Calibrating CNNs for Lifelong Learning" published in NeurIPS 2020. Paper presents a way to train CNNs for different tasks contiually without getting caugt in the catastrophic forgetting phenomenon by incorporating calibration modules after each convolutional layer. Aim of this project is to reproduce the experiments described in the paper, in detail, we want to achive gracefully degrading classification performance as the number of tasks increases. Moreover, the aim of this project is to shift the attention of the readers to the subject of "Continual Learning".

1.1. Paper Summary

In lifelong learning, if the network forgets information from old tasks during training on a new task, the network is called plastic network; on the contrary, if the network focuses on older tasks and not learning the new task, then the network is called stable network.

In plastic networks, catastrophic forgetting is observed such that as the new taks is learned, older tasks are being forgotten. Conventionally, lifelong learning are done in three methods:

- **Regularization-based methods:** when the experimenter wants to ensure that the network outputs do not drastically change while training on new tasks, when the old task knowledge cannot be preserved to some extent. In this respect, the paper does not need to use regularization loss since it does not suffer from catastrophic forgetting.
- **Memory-based methods:** a dedicated memory for storing old task exemplars are needed to use them in training for new tasks. However, the paper only store taks-adaptive calibration parameters for new tasks so that catastrophic forgetting is reduced for older tasks.
- **Dynamic network methods:** dynamically modify the network to deal with training on new tasks, usually by network expansion. However, number of parameters in such methods can quickly become very large while re-calibration based approach in the paper requires less storage and computation.

Another approach for continual learning might be to train the network starting from the most recent task, store all parameters for for each task and repeat this procedure for each new task. In this respect, the paper proposes a balanced network between a stable network and a plastic network through a calibration modules added after each convolutional layer of the base model. The base layers of the network extracts the common features on the initial task so these layers are frozen at the end of the training of the first task. On the other hand, the new tasks are learned via training only the calibration modules to make older tasks relevant to the current task, hence, number of parameters and computational cost do not increase drastically. In a sense, the proposed method promotes transfer learning via sequentially trained calibration modules and offers an efficient way to deal with the catastrophic forgetting.

The proposed calibration modules meet the following criteria:

- near-zero catstrophic forgetting
- not drastically increased number of parameters
- forward transfer learning

So the article offers a novel method that involves (re)calibrating the activation maps generated by the network trained on older tasks. Each calibration module consists of a spatial calibration module (SCM) followed by a channel-wise calibration module (CCM). The spatial calibration module learns weights to calibrate each point in the activation maps while the channel-wise calibration module learns weights to scale each channel of the activation maps to adjust the impact on further layers.

For each task followings are stored and it has been assumed that during inference task labels are provided:

- After 1st task, base models conv layers
- For upcoming tasks since base models layers are frozen only calibration models and classifier heads are stored.

2. Method and My Interpretation

2.1. Original method

The paper considers sequence of classification tasks with same base convolutional layers such that after each convolutional layer SCM and CCM modules are incorporated to calibrate activation maps of convolutional layers.

- The **SCM module** consist of a group convolution layer with kernel=3×3, stride=1, padding=1 and num_groups=$\frac{C}{\alpha}$ (most of the experiments in the paper are realized with $\alpha = 1$) and performs element-wise addition for the given input activation map $x \in \mathbb{R}^{H \times W \times C}$:
$$y = SCM(x) = x \oplus gconv_{\alpha}(x)$$
- The **CCM module** consist of gloabal average pooling (GAP), group convolution layer with kernel=1×1, stride=1, padding=0, num_groups=$\frac{C}{\beta}$ (most of the experiments utilized $\beta = 1$ and note that GAP followed by group convolution produces output $m \in \mathbb{R}^{1 \times 1 \times C}$), batch-normalization, sigmoid activation and performs channel-wise multiplication on the output of SCM module $y \in \mathbb{R}^{H \times W \times C}$:
$$z = CCM(y) = y \otimes \sigma(BN(gconv_{\beta}(GAP(y))))$$
- The overall calibration module $CCM \circ SCM$ is added after each convolutional layer of the base model:
$$x_{out} = CCM(SCM(conv(x_{in})))$$

For the 1st task, both base model's convolutional layers and incorporated calibration modules are trained. At the end of the training of 1st task, the base model's convolutional layers are frozen and never trained again. For the new tasks, only the calibration modules are trained where they are initialized with the most recent tasks' calibration modules and each task has its own classifier head initialized randomly at the start of training. The models for each task are saved at the end of the training sessions and saved models utilized during inference/test session.

Following datasets are used for the experiments:

- **SVHN:** 2 consequtive classes are grouped for a task such that 5 tasks in total
- **CIFAR100:** 10 classes are grouped together to form 10 tasks in total
- **splitCIFAR:** 1st task considered as CIFAR10 task, then 5 new tasks are formed by randomly choosing 10 classes from CIFAR100 to construct a task, 6 tasks in total
- **ImageNet-100:** subset of ImageNet used containing 100 classes in total, grouped into 10 tasks of 10 classes each
- **MS-Celeb-10K:** contains 10000 classes, grouped into 10 tasks each containing 1000 classes

Following models, hyperparameters, optimizers are utilized for aforementioned datasets:

- **SVHN:** model=ResNet18, epochs=150, lr=0.01, lr_decay=0.1 @ epochs=(50, 100, 125), optimizer=SGD
- **CIFAR100:** model=ResNet18, epochs=150, lr=0.01, lr_decay=0.1 @ epochs=(50, 100, 125), optimizer=SGD
- **splitCIFAR:** model=ResNet18 and ResNet32, epochs=150, lr=0.01, lr_decay=0.1 @ epochs=(50, 100, 125), optimizer=SGD
- **ImageNet-100:** model=ResNet18, epochs=150, lr=0.01, lr_decay=0.1 @ epochs=(50, 100, 125), optimizer=SGD
- **MS-Celeb-10K:** model=ResNet18, epochs=70, lr=0.01, lr_decay=0.1 @ epochs=(20,40,60), optimizer=SGD

2.2. My Interpretation

Some implementation details are not given in the paper, so I made my own assumptions on them:

1. The blocks of ResNet has the following structure: Conv --> BN --> ReLU --> Conv --> BN. Moreover, the CCM modules also contain BN for calibration. Therefore, we seperate SCM and CCM modules for each conv layer as the following: Conv --> SCM --> BN --> CCM --> ReLU --> . . . In our experiments we have also tried the following configuration: Conv --> SCM --> CCM --> BN --> ReLU --> . . . but the performance was worse than the former configuration, that is why, we did not report results for latter configuration.
2. For CIFAR100 experiment, the paper did not mention how classes are distributed to the tasks. Therefore, we sequentially grouped classes with respect to class indices provided by the dataset.
3. I could not find the mentioned subsets of **ImageNet-100** and **MS-Celeb-10K** in the provided references of the paper. Therefore, we tried to modify the full-datasets, however, due to limited memory capabilities we did not perform large-scale experiments on Colab.
4. The preprocessing and data augmentation steps also are not mentioned in the paper. Therefore, we only utilized normalization on the input images and did not perform data augmentation in any of the tasks.
5. Batch sizes are not provided in the paper. Therefore, we utilized the following batch sizes:

- **SVHN:** batch_size=128 for each of the 5 tasks
- **CIFAR100:** batch_size=80 for each of the 10 tasks
- **splitCIFAR:** batch_size=64 for each of the 6 tasks

3. Experiments and Results

3.1. Experimental Setup

- I have implemented the modified ResNet model with calibration modules using PyTorch (and related libraries) as mentioned in Chapter 2.2. Remark 1.
- The paper also includes experiments on large-scale datasets ImageNet and MS-Celeb-10K, however, due to memory constraints and computation capabilities/training times these experiments are not reproduced. Only SVHN, CIFAR100 and splitCIFAR experiments are ran.
- The computation time without disconnection is limited on Colab. Therefore, it is not possible for us to train network for 150 epochs for each task. Eventhough, utilization of momentum acceleration are not mentioned in the paper, I have implemented in our training procedure for speed up and train for smaller number of epochs. Correspondingly utilized learning-rate scheduling also modified. The following shows the hyperparameters utilized in our setup for experiments:

1. **SVHN:** model=ResNet18, batch_size=128, lr=0.01, epochs=20, optimizer=SGD, momentum=0.9
2. **CIFAR100:** model=ResNet18, batch_size=80, lr=0.01, epochs=50, lr_decay=0.1 @ epoch=(20, 35), optimizer=SGD, momentum=0.9

3.2. Running the code

- Each provided experiment has its own notebook (some overlap between codes evident).
- ***Note**: Provided notebooks contains some of the experiments in the paper (**SVHN**, **CIFAR100** and **splitCIFAR**), the uncovered experiments can be realized by the reader by using the provided notebooks as reference.* Feel free to copy the notebooks to your drive and update hyper-parameters etc. for your own experiments.
- Notebooks contains sections (in order):

1. **Libraries:** Google Drive access and import required libraries.
2. **Utils:** Utility functions such as accuracy computation or specific layer freezing.
3. **Model:** Modified ResNet implementation with calibration modules.
4. **Dataset:** Dataset and dataloader prepration for train, val and test subsets
5. **Main:** Contains training and testing functions for experiment.

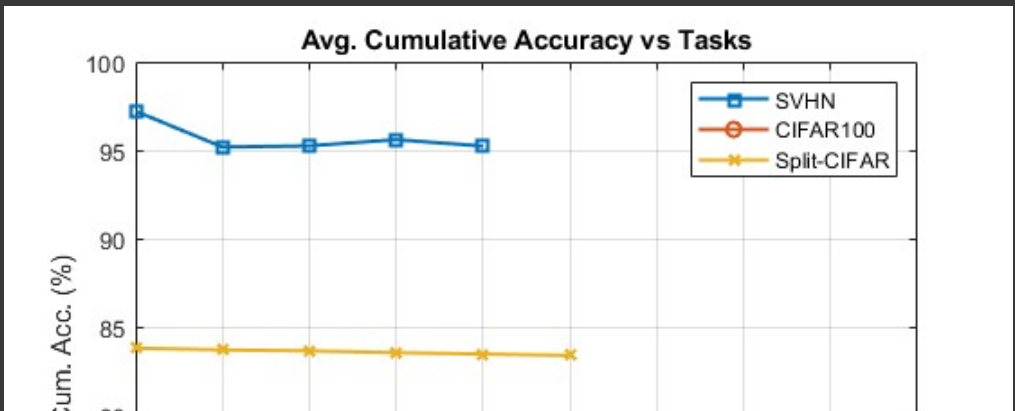
- Models and tensorboard logs are automatically saved to your drive during execution.

3.3. Results

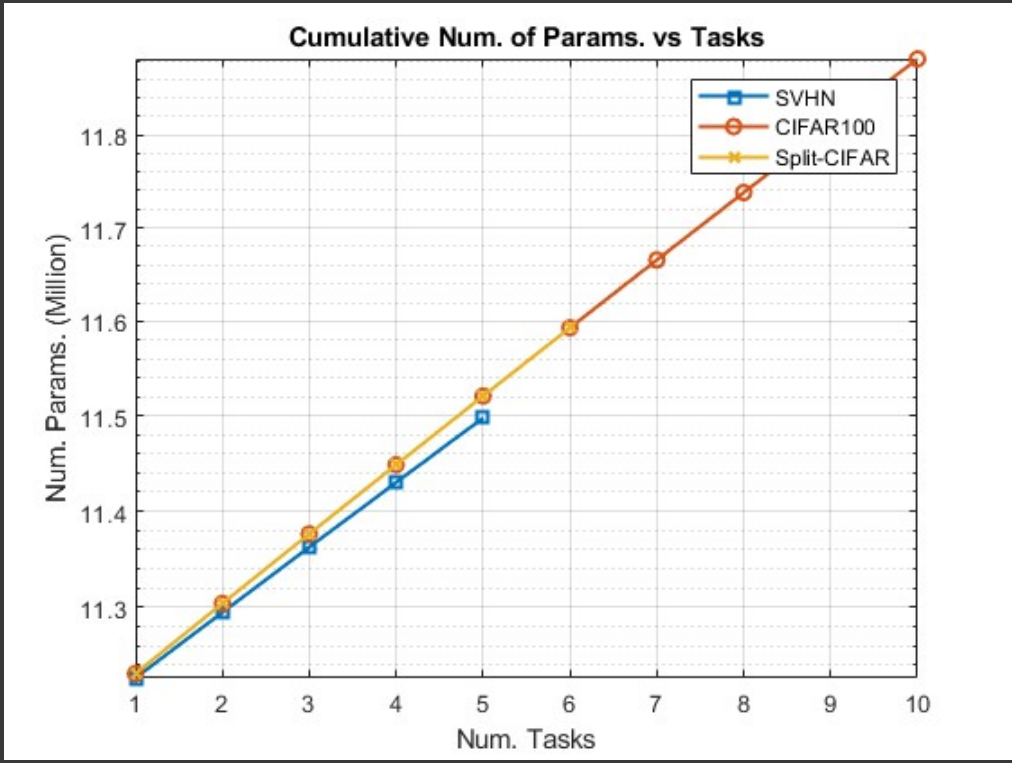
```
Table 1: Average accuracies of tasks for given experiments (Paper vs. Ours)
```

Dataset	Paper	Ours
SVHN	98.2	95.1
CIFAR100	82.3	69.9
splitCIFAR	89.7	84.7

- Comparison of our foundings and paper's are shown in Table 1. Eventhough, our performances are lower than what has been reported in paper due to limited/constrained hardware capabilities, for **SVHN** and **splitCIFAR** experiments we achieved good performances. Our **CIFAR100** experiment on the other hand was not that good.



degredation on the average accuracies as the number of tasks increases. Moreover, we also observed that performance on the rest of tasks is highly dependent on the performance on the 1st task.




```
Figure 2: Number of parameters vs. Number of Tasks
```

- Most of the memory consumed by the initial tasks parameters, since 1st task starts with base model's parameters. On the other hand incorporated calibration modules do not cause significant memory overhead as the number of tasks increases. As you can see from the figure above, each added task only increases the number of total parameters approximately 0.9% with respect to number of initial parameters.

4. Conclusion

Our results are aligned with paper's claims in a sense that:

1. We did not observe significant performace degregation as the number of tasks increases, eventhough for some experiments our performance were worse than the paper's foundings.

overhead caused by calibration modules are very small compared to initial model size. Still, it would be more impressive if we could add these calibrators only after each block rather than each convolutional layer.

5. References

[1] P. Singh, V. Verma, P. Mazumder, L. Carin and P. Rai, "Calibrating CNNs for Lifelong Learning", Papers.nips.cc, 2021. [Online]. Available: https://papers.nips.cc/paper/2020/hash/b3b43aeeacb258365cc69cdaf42a68af-Abstract.html.