REPRODUCABILITY CHALLENGE: DELTA

Analle Abuammar
aa8n19@soton.ac.uk

Steven Larry Ball
sldb1g19@soton.ac.uk

Swetha Pillai
sp3e19@soton.ac.uk

ABSTRACT

For the Deep Learning Reproducibility Challenge we chose to study and reproduce Li et al's paper: 'DELTA: Deep Learning Transfer using Feature Map with Attention for Convolutional Networks' (1). This paper proposes a novel regularised transfer learning framework 'DELTA' in the context of convolutional networks, and evaluates its performance against state of the art algorithms. With this method, Li et al seeks to solve the problem of bottle-necked accuracy during transfer learning.

1 INTRODUCTION

DELTA uses the distance between the outer layer outputs of the source and target network feature maps by weighting them according to their loss per feature map. This allows us to see which channels contribute the most towards the performance of the model. This stage is done in the 'feature_weights' function (channel_eval.py) and is described as regularising the behaviour of the networks. The channels that seem to contribute most is preserved through regularisation. As for the weaker channels, they reuse them using attention with feature map regularisation in the function 'reg_att_fea_map' (train.py). This is what they call unactivated channel re-usage. First we train ResNet-101 architecture with respect to Stanford Dogs 120 dataset, starting with the weights of the pre-trained ImageNet dataset. Here the method is incorporating the technique of 'Starting Point As Reference' strategy (SPAR) as we are taking advantage of ω^* as our initialisation parameters to eventually learn a new set of parameters ω. This step allows an increase in optimisation speed and generalisability of the model. Now that we have our source model parameters ω^* we proceed onto using the feature_weights function, to record the weights of the channels based on their aggregate performance when the channel is present in the network and when it is not. In this step we take the average loss of over all the batches of images as our channel weight. These newly produced weights, $\hat{\mathrm{W}}_{\mathrm{j}}$, are used in our train.py code to help form our regularisation metric for loss. These weights are first standardise by subtracting the mean and dividing by the standard deviation of the weights and then normalise using a softmax function. This results in the set of weights lying between 0 and 1 and being non-negative. Then we begin the training process of training our model, using eq.5 in the paper, as our 'behavioural regulariser'. The behavioural regulariser term is incorporated along side an extra regularising term which regularises a subset of parameters unique to the target network which claims to 'improve the consistency of inner layer parameters estimation'. The regulariser terms are each accompanied by their own tuning parameters which we kept as 0.01. The loss function that needs to be minimised includes: the cross entropy loss, followed by the aforementioned tuned regularisation terms.

2 THE IMPLEMENTATION

The implementation of the paper is available on Github using Pytorch, so we used the code provided. It took us some searching and brainstorming to understand each line of code. Due to time constraints we sought more GPU power through Microsoft Azure services, giving us access to 60 GPU GB memory and Google Colab providing 12GB NVIDIA for a continuous use of 12 hours. The Stanford Dogs 120 benchmark dataset was chosen to represent a task that offered fine grained object recognition. We decided to focus on reproducing the results related to Stanford Dogs 120 dataset, as most of the results were related to this dataset in the paper. In addition, Stanford Dogs had the small enough data size, to allow us to explore more experiments and perform more analysis under a smaller period of run-time. Also, we chose to work with ImageNet because Stanford

Dogs 120 is more similar to ImageNet than to Places 365 (2). Instead of collecting results for several state-of-the-art regularisation methods, like the paper had done, we chose to focus on L^2-SP as a baseline method to compare DELTA, because this was used in the majority of the results and it seemed to be the best performing technique amongst the other methods according to the results presented for Stanford Dogs 120. Our code for reproducing Fig.3 (Distribution of distance of parameters) and Fig.4 (Activation maps) of the paper are included in our GitHub repository: https://github.com/COMP6248-Reproducability-Challenge/DELTA . As for modifications we made to the code, the lrscheduler.step() was called before the optimizer.step(), in the original train.py code, which can result in PyTorch skipping the first value of the learning rate schedule, so we swapped the order to prevent this occurring.

3 Comparisons with performance (Results)

The first set of results we sought to reproduce was from Fig.1 in the paper (Fig.**??**), which plotted the top-1 testing and training accuracy of both L^2-SP and DELTA regularisation methods under different learning rate schedulers: Step learning rate scheduler (STEP-LR) and exponential learning rate scheduler (EXP-LR) (Fig.**??**). We realised that due to the stochastic gradient descent optimiser, the graphs could differ for every run of the experiment. To help us understand the differences between the results more, we plotted the average results (over 5 experiments) along with their interval standard deviation bars to help expose the statistical significance of any differences in the graphs. Our results support the original findings that although step learning rate scheduler (STEP-LR) was implemented for both methods. However, it would be interesting to see if L^2-SP does indeed achieve a higher accuracy after a longer number of epochs, as its trajectory implies. As for the presentation of the graphs themselves, perhaps it would be more intuitive to plot these graphs with respect to epochs rather than iterations, since we are essentially plotting the average top-1 accuracy per epoch.

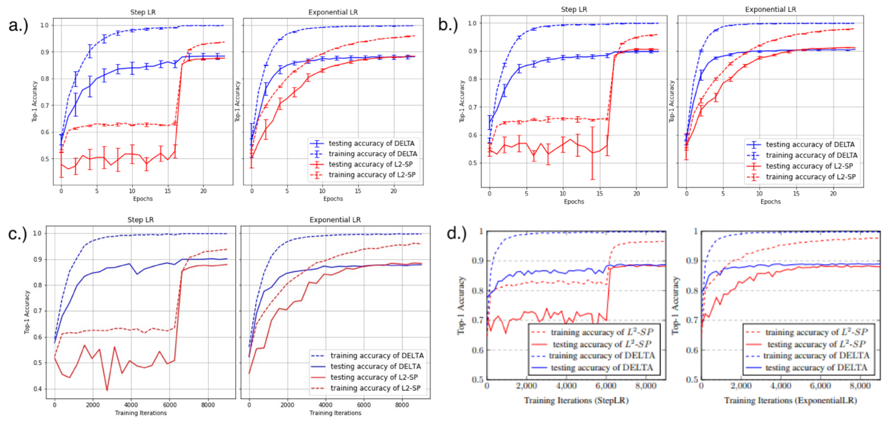

Figure 1: a) average accuracies over 5 runs, b) average accuracies including the augmentation step for the source model. c) our reproduced results of the original graph (d).

We found that even though the difference is statistically significant between L^2-SP and DELTA in the averaged top-1 accuracy comparison in Table.1 of the paper, interestingly this was not the case in the results post-data augmentation. For Table.2 of the paper, the top-1 accuracy was recorded after the addition of data augmentation techniques: different resizing method and the addition of a 10-crop testing step. The results presented show that the higher bound of the L^2-SP result matches the lower bound of the DELTA result, which means we cannot conclude that DELTA still outperforms L^2-SP based on these results. Therefore the declaration of that DELTA 'still delivers the best one' (accuracy) is unfounded. Having said that to investigate this method further one would need to increase the number of runs from the original 5 runs, in hope to infer a better understanding on the performance of the two techniques in the context of data augmentation. Increasing the total number of runs also applies to the results in Table 1. To study the convergence path of our two models after the added stage of augmentation, we plotted a similar graph of top-1 accuracy, Fig.**??**. This shows relatively less variation with regards to error bars, but still has overlapping results for the last epoch for test accuracy of the two regularisation methods.

Our Results	L^2**-SP**	**DELTA**
Without AUG	87.5 ± 0.2	88.3 ± 1.2
With AUG	90.4 ± 0.3	89.3 ± 1
With Aug (source model augmented)		89.8 ± 0.6
Original Results	L^2-SP	DELTA
Without AUG	88.3 ± 0.2	88.7 ± 0.1
With AUG	90.8 ± 0.2	91.2 ± 0.2

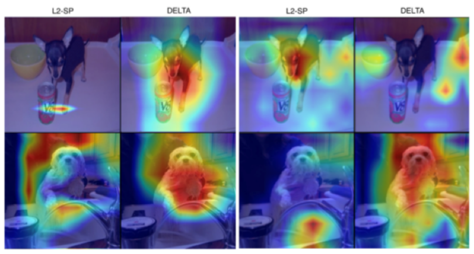

Figure 2: a) average accuracies over 5 runs, b) average accyr

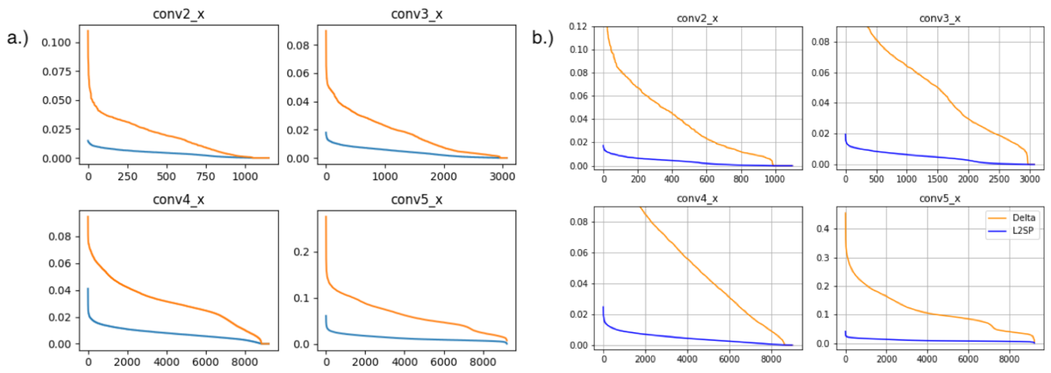

Figure 3: (a.) Original Results: Distribution of the distance of parameters from the starting point. The Euclidean distance between the flattened weights of the source and target models over each of the channels in each block : conv2x, conv3x, conv4x, conv5x. These distances are plotted for both DELTA and L^2-SP (Fig.3 in the paper).(b.) Our Results: Our re-production of the original graph.

Table**??** shows our reproduced results of both Table.1 and Table.2 from the paper. Our results show that even though the average values are better for DELTA, without the augmentation step, there is an overlap between the standard deviations, therefore we cannot conclude that DELTA performs better than L^2-SP in this case, unlike the original results which finds a clear 0.1 difference between standard deviations. As for our results after including data augmentation, we find that there is still an overlap between the results. It is also interesting to see how much our results for DELTA varies (1.2 and 1.0), which is much more spread than the original results. These differences is probably a result of not tuning our α that balances our novel regularisation term (kept it at 0.01 default), unlike the original paper which tuned this parameter using 5-fold cross validation, due to time restrictions. During this process, we found that we may have overlooked incorporating the data augmentation stage for the initial training process of Stanford Dogs 120 on ResNet 101 prior to the feature map weighting in channeleval.py. Hence we ran the five experiments with augmenting the data for our initial training phase which presented the results as shown in Table **??**. The extra results show that there still seems to be an overlap between the spread of top-1 accuracy for L^2-SP and DELTA, however the spread of our results seem significantly lower than our previous set of results.

Next we attempted to reproduce the results of Fig.3 of the paper Fig.3. These graphs show the distribution of the distance between the parameters of the pre-training source network and the target models: DELTA and L^2-SP for each block of the network. The process of obtaining the values were described well and these results were relatively reproducible, taking into account that due to stochasticity of the optimiser the parameters would have some variation over the runs. We found finding a suitable filter that reflected the results shown in Fig.4 of the Appendix (Fig.**??**), particularly challenging. This was because, although it was explained that the convolution filters for the activation mapping were low activation, we struggled to find a filter that showed the difference in activation maps as seen from the original images. Instead, we found that the difference in activated pixel distributions after different methods of regularisation was applied was not as evident.

4 General challenges we faced

During the process of re-implementation we encountered several challenges. This included incurring an 'out-of-memory' error in two different scenarios, even when using our GPU access for Google-Colab and Microsoft Azure. First we found this error in the context of 'channeleval.py' while running the code on Google-Colab. This was handled by adding before the evaluating phase "torch.nograd()". In another instance, we experienced this error in the context of 'train.py' when using the Microsoft Azure computational resources. This was due to the fact that we kept the batch size for learning our target model as 64, as advised by the document. This was solved by changing the size of our batch to 32, which did not change the results per se but resulted in the model training over a total of 24 epochs rather than 48 (keeping the maximum number of iterations at 9000). When we began to reproduce the Table.2 results, after including a stage of data augmentation before training the model, we found that the description was not explicit in its instructions of whether or not to include this augmentation step during the initial stages of weighting feature maps with supervised attention models (channel_eval.py), so initially we decided not to include this step when training our source model. However, in reflection, we realised that it would be important to augment the data in the same way as the core train.py stage, as the supervised attention stage would need to work with the same augmented data in order to work effectively. Another key issue we faced was the fact that no information about the computational resources used or average run time taken during the experimentation was provided. This meant that we lost much of our time trying to run the procedure and plan around this unknown, and ultimately led us in trying to get a hold of larger memory. Our runs took an average of 8 hours, ranging from 4 hours for our runs using EXP-LR to 12 hours on average in the runs that included the data augmentation step.

5 Reflections and Conclusion

Some of information of the parameters used in the experimentation phase was only clarified in the code provided, for example the tuning parameters α and β. The code, however, could have been more helpful if it had been accompanied by some comments. Due to time constraints we chose not to re-implement Table.3 of the paper which compared average activations of 15 discriminate parts of CUB-200-2011 dataset with different regularisation methods, as we were focusing our efforts on reproducing results around Stanford Dogs 120. We feel that although its core idea of a novel regularisation method offered a promising and intuitive view on how one should regularise the loss function, we found this intuition was not reflected in the explanations of the method in the paper. To continue this study further, we would have looked into tuning the hyper-parameters of the network to help obtain a better reproduction of the results (e.g. α). The results presented in the paper was not as straight forward to reproduce as we had hoped. From our set of results we cannot support the assertion that DELTA does in fact provide a better accuracy than L^2-SP, however we cannot say that this is in turn means that the assertion is false due to the differences in our approach to the experiments.

References

[1] Xingjian Li, Haoyi Xiong, Hanchao Wang, Yuxuan Rao, Liping Liu, and Jun Huan. Delta: Deep learning transfer using feature map with attention for convolutional networks. *arXiv preprint arXiv:1901.09229*, 2019.

[2] Xuhong Li, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. *arXiv preprint arXiv:1802.01483*, 2018.