Increasing the Accuracy of Convolutional Neural Networks with Progressive Reinitialisation

Craig Atkinson
Department of Computer Science
University of Otago
Dunedin, New Zealand
Email: atkcr398@student.otago.ac.nz

Brendan McCane
Department of Computer Science
University of Otago
Dunedin, New Zealand
Email: mccane@cs.otago.ac.nz

Lech Szymanski
Department of Computer Science
University of Otago
Dunedin, New Zealand
Email: lechszym@cs.otago.ac.nz

***Abstract*—This article introduces a training technique called progressive reinitialisation. This technique involves training a Convolutional Neural Network layer by layer. This is achieved by training the whole network and progressively freezing lower layers' weights until the whole network is frozen. When a layer is frozen, all weights in higher layers (unfrozen layers) are reinitialised before training continues. This training procedure is shown to boost the final network's performance by about 2% on CIFAR-10 and 1% on SVHN in the absence of image augmentation.**

I. Introduction

Convolutional Neural Networks (CNNs) are a network architecture proposed by LeCun et al. [1] for character recognition and since then have been successfully applied to a wide range of classification tasks, with the majority of research focusing on image classification. Standard CNNs comprise of a number of convolution and max-pooling layers, followed by at least one fully-connected layer and then a softmax layer. Typically, the weights in all layers are initialised before training begins and then they are updated each mini-batch to minimise the network's error on the problem.

Previous research has shown that during training, early layers can be frozen before later layers because they converge on their feature detectors earlier in the training procedure compared to later layers [2]. This article reports on a variation of this scheme, which freezes consecutive layers and reinitialises all unfrozen weights before continuing training on the unfrozen weights. This results in the network's layers being learnt in a forward order.

A. Related Work

CNNs are generally trained via stochastic gradient descent which changes the network's weights in the direction that minimises the error on a dataset, according to a cost function. This is often paired with techniques such as momentum which allows each weight to be changed by a very different amount than other weights in the network. However, it has been observed that when using these techniques it is common for all weights in a layer to take approximately the same sized step, which is not the case between layers [3]. Therefore, an adaptive learning parameter was proposed to control the step size for all units in a layer and thus, reducing the number of learning parameters. This technique even increased the network's accuracy compared to the other methods tested, suggesting that different layers can have different training requirements.

In CNNs, weights in each of the network's layers become feature detectors which recognise patterns in the input images. Features detected by early layers tend to be basic (e.g. lines and dots), whereas features detected by later layers are more complex (e.g. eyes and ears) [4]. Brock et al. [2] suggested that early layers converge on their feature detectors earlier than later layers because they are learning simpler patterns. This means that during training you can freeze earlier layers before later ones. This reduces the number of parameter updates being made and thus reduces the training time of a CNN without having a major impact on the network's accuracy. This is important as it conveys that it is possible to finish training earlier layers of a network before later ones.

Researchers have also constructed algorithms which train random subsets of a CNN's layers such that collectively the layers are an effective classifier [5]. This was achieved by using an architecture called ResNet [6] which essentially has each layer output:

$$o = ReLU(bF(x) + x), \tag{1}$$

where x is the input to the layer, o is the output of the layer and $F(x)$ represents the convolutional operation that applies the layer's weights to the input. The only difference between a standard ResNet and the one used by [5] is that b, which is just a random boolean (0 or 1), is introduced into the latter. If this value is 0 the layer is skipped and no learning occurs and if it is 1 the layer's convolutional operation is not skipped and the network's weights are updated. This technique allowed them to train a very large CNN by training subsets of it at one time and this network outperformed the state of the art solution on the image recognition dataset CIFAR-10. For this training procedure to be effective the probability that each layer was skipped (i.e. $b = 0$) was very small for the initial layers of the network and then larger for later layers. This suggests that it was important to train these earlier layers before significant training occurred in the later layers.

B. *Progressive Reinitialisation*

Currently the standard practice for training CNNs is to train all weights in the network at once, regardless of which layer they belong to. However, this paper investigates whether it can be beneficial to learn one layer at a time, starting with the first layer. This is done by training the whole network until it converges and then freezing the first layer's weights to the values that produced the lowest error on the validation set. All other weights in the network are reinitialised and training continues until convergence. At this point, the next layer's weights are also frozen and this process continues until all layers in the network are frozen. The motivation for this work was the hypothesis that this training procedure will result in a network that outperforms the same network trained with standard stochastic gradient descent. We present experiments to show this is indeed the case, and also experiments that attempt to illuminate why this is the case.

II. Method

A. *Dataset*

Experiments have been performed using the CIFAR-10 and SVHN datasets. The CIFAR-10 dataset comprises 60,000 coloured images of the following classes; airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Each image's dimension is 32×32 and the goal for the network is to determine which of 10 possible classes the image belongs to. The CIFAR-10 dataset was split up such that 37,500 images were used for training, 12,500 for validation and 10,000 for testing the network. Images were cropped to 24×24 and standardisation was individually applied to each image.

The SVHN dataset comprises 99,289 coloured images of the digits 0-9 taken from street addresses. Each image is 32×32 and the goal for the network is to determine which of 10 possible classes the image belongs to. This dataset was split up such that 54,942 images were used for training, 18,315 for validation and 26,032 for testing the network. Standardisation was individually applied to each image.

B. *Architecture*

Two network architectures were tested; one architecture was used exclusively for tests on the CIFAR-10 dataset and the other exclusively on the SVHN dataset. A summary of these architectures can be found in Table I. All convolutional layers apply 3×3 filters with a stride of 3 and all max-pooling layers apply a 3×3 window with a stride of 2. All neurons in the convolutional and fully-connected layers use the ReLU activation function. L_2 regularisation with a scale of 0.004 is applied to every layer in the network (except the softmax layer and bias weights).

When initialising and reinitialising, all convolutional layer weights are randomly sampled from a normal distribution with a mean of 0 and a standard deviation of 0.05, except for bias weights which are set to the value 0.1. For the fully-connected layers, weights are randomly sampled from a normal distribution with a mean of 0 and a standard deviation of 0.004 and the biases are set to 0.1. Finally, the softmax weights are randomly sampled from a normal distribution with a mean of 0 and a standard deviation of 1/192 and biases are set to 0. These values are recommended by Tensorflow's CIFAR-10 example.

TABLE I
ARCHITECTURE OF THE TWO CNNS TESTED. CONV-n REPRESENTS A CONVOLUTIONAL LAYER WITH n FILTERS, FC-m REPRESENTS A FULLY-CONNECTED LAYER WITH m UNITS AND SM REPRESENTS THE SOFTMAX LAYER WITH 10 UNITS.

CIFAR-10	SVHN
Conv-64 Conv-64	Conv-64 Conv-64 Conv-64
max-pool	max-pool
Conv-128 Conv-128	Conv-128 Conv-128
max-pool	max-pool
	Conv-256 Conv-256
	max-pool
FC-384 FC-192	FC-192
SM	SM

C. *Training and Evaluation*

The networks are built, trained and evaluated using the Python library Tensorflow [8]. Networks are trained with the Adam optimiser [9], using a learning rate of 0.001, first moment decay rate of 0.9, second moment decay rate of 0.999 and an epsilon of 1×10^{-8}. The network is trained on the training examples using a mini-batch size of 512 and after each epoch the current network's loss is recorded for the validation examples. After training is completed, the network at the epoch with the lowest validation loss is evaluated on the test examples. This process is repeated 10 times for each condition and the mean result is compared using Welch two sample t-test with a significance level of 5×10^{-2}.

D. *Experimental Conditions*

1) Without Image Augmentation: The $reinit$ condition trains using the procedure specified in section I-B where the freezing and reinitialisation occurs after every 40 epochs of training for a total training time of 280 epochs. 40 epochs is chosen, as the network's weights converge well before this point. The std condition trains the network for the same number of epochs as the $reinit$ condition, i.e. 280 epochs, but does not freeze or reinitialise any weights and thus, carries out standard stochastic gradient descent.

The rev_reinit condition is identical to the $reinit$ condition except the freezing of layers begins at the last layer of the network and stops at the first layer of the network with reinitialising occurring on all weights that are not frozen. The $rand\_reinit$ condition freezes the layers in a random order, while reinitialising all weights in layers that are not frozen.

The $reinit\_w/o\_freezing$ condition does not freeze any weights but every 40 epochs it reinitialises its weights. Reinitialisation begins for all weights except the first layer and then all weights except those in the first two layers and so on.

TABLE II
RESULTS FOR CONDITIONS THAT WERE TRAINED ON THE CIFAR-10 DATASET, WITHOUT IMAGE AUGMENTATION. THE $*$ REPRESENTS CONDITIONS THAT PRODUCED NETWORKS WHOSE ACCURACIES WERE SIGNIFICANTLY DIFFERENT TO THE $reinit$ CONDITION.

Conditions	Test Accuracy (%)	P-value (vs. reinit)
reinit	77.2	
std	75.2	0.00*
rev_reinit	75.3	0.00*
rand_reinit	76.5	0.00*
reinit_w/o_freezing	76.8	0.12

TABLE III
RESULTS FOR CONDITIONS THAT WERE TRAINED ON THE SVHN DATASET, WITHOUT IMAGE AUGMENTATION. THE $*$ REPRESENTS CONDITIONS THAT PRODUCED NETWORKS WHOSE ACCURACIES WERE SIGNIFICANTLY DIFFERENT TO THE $reinit\_svhn$ CONDITION.

Conditions	Test Accuracy (%)	P-value (vs. reinit_svhn)
reinit_svhn	94.7	
std_svhn	93.4	0.00*

Finally, the $reinit\_svhn$ and std_svhn conditions are identical to the $reinit$ and std conditions respectively, except training and testing is done on the SVHN dataset with a larger network architecture. The $reinit\_svhn$ condition applies its freezing and reinitialisation every 40 epochs for 360 epochs and thus, the std_svhn condition trains for a total of 360 epochs.

2) With Image Augmentation: The $reinit\_aug$ condition is the same as the $reinit$ condition, except that it freezes and reinitialises every 256 epochs for a total of 1792 epochs and is trained on the augmented dataset. 256 epochs is chosen because the network has converged well before this point. The std_aug condition is the same as the std condition, except it trains for 1792 epochs and also learns from the augmented training dataset. The $reinit\_aug$ and std_aug conditions are run 3 times each and it is only the training data which is augmented each epoch.

The image augmentation applied by the $reinit\_aug$ and the std_aug conditions involves cropping random 24×24 images from the training set, randomly flipping images left or right, adjusting brightness randomly between -63 and 63, randomly adjusting contrast between 0.2 and 1.8 and then individually applying standardisation to the images.

TABLE IV
RESULTS FOR CONDITIONS THAT WERE TRAINED ON THE CIFAR-10 DATASET, WITH IMAGE AUGMENTATION. THE $*$ REPRESENTS CONDITIONS THAT PRODUCED NETWORKS WHOSE ACCURACIES WERE SIGNIFICANTLY DIFFERENT TO THE $reinit\_aug$ CONDITION.

Conditions	Test Accuracy (%)	P-value (vs. reinit_aug)
reinit_aug	85.1	
std_aug	84.9	0.40

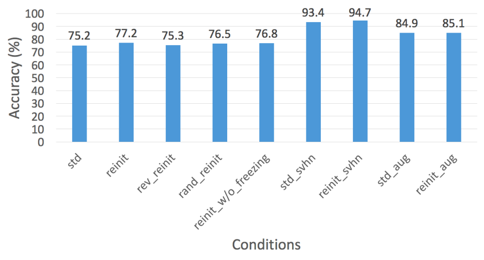

Fig. 1. Average accuracy of networks produced by the various conditions.

III. RESULTS

The results of the exhaustive conditions tested are displayed in Table II-IV and summarised in Fig. 1. Results in Table II displays the $reinit$ condition significantly out-performing all other conditions, except for the $reinit\_w/o\_freezing$ condition, on CIFAR-10 when image augmentation was absent. Table III displays the $reinit\_svhn$ condition significantly out-performing the std_svhn condition and Table IV conveys no significant difference between the $reinit\_aug$ and the std_aug condition on CIFAR-10 when image augmentation was present.

IV. DISCUSSION

The aim of the $reinit$ and std conditions was to determine whether the performance of a CNN could be improved by freezing and reinitialising sequential layers. The results demonstrate that the $reinit$ condition significantly outperforms the std condition. This confirms the hypothesis that the performance of a CNN can be improved by freezing and reinitialising sequential layers.

However, these results are limited as they do not determine why this procedure outperforms the other. This leaves unanswered questions such as:

- Is the order of freezing and reinitialisation important to the performance of the network?
- Is the freezing of layers necessary to progressive reinitialisation or is it solely the reinitialisation which results in these performance gains?

The rev_reinit and the $rand\_reinit$ conditions were introduced to determine whether the order of reinitialisation was important and the $reinit\_w/o\_freezing$ condition was introduced to determine whether freezing is important to progressive reinitialisation.

The $reinit$ condition significantly outperformed both the rev_reinit and the $rand\_reinit$ conditions which confirms that the order of freezing and reinitialisation does matter for progressive reinitialisation. More specifically, it is more advantageous to start freezing weights and stop reinitialising weights from the beginning of the network compared to reversing this order or freezing and reinitialising weights in a random order.

Progressive reinitialisation essentially learns the final weights for a single layer in a CNN at one time. More specifically, the procedure begins training the whole network

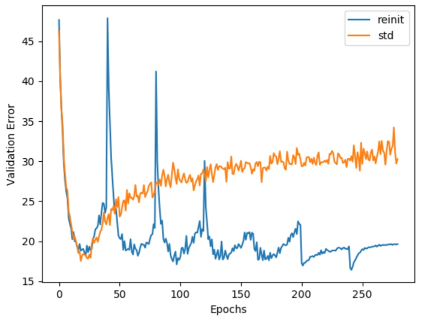

Fig. 2. Validation error for each epoch of the $reinit$ and std conditions. The validation error is the average cross entropy loss across the mini-batches.

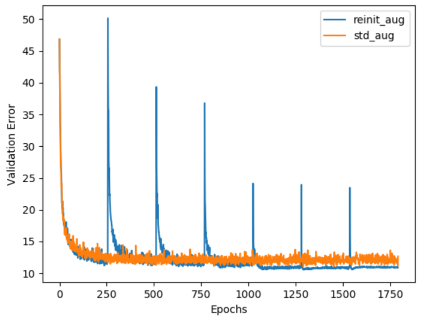

Fig. 3. Validation error for each epoch of the $reinit\_aug$ and std_aug conditions. The validation error is the average cross entropy loss across the mini-batches.

but only freezes the weights for the first layer and then reinitialises all others. Then the network trains all remaining layers and freezes the weights found for the second layer and so on. This means that the network is essentially learning the best weights for early layers before later layers. Therefore, our results suggest that perhaps CNNs should not be trained all at once as they are currently, but rather in this forward order.

From [2] we can hypothesise that the freezing of early weights to later weights is not necessary to progressive reinitialisation. The results of the $reinit\_w/o\_freezing$ condition compared to the $reinit$ condition supports the idea that the freezing of weights does not significantly increase the error of the network. This demonstrates that it is only the reinitialising of layers that produces the performance increase.

Although progressively freezing layers of weights did not improve the accuracy of the CNN, it did not decrease its accuracy either. This is important as freezing the weights has another benefit; it reduces the training time of the network. This is because, when a layer is frozen, its weights do not need to be changed and errors do not need to be propagated to that layer and therefore, the computations involved in training are reduced. More specifically, the 10 trials for the $reinit$ condition took 175 minutes to train, whereas the $reinit\_w/o\_freezing$ trials took 254 minutes. This is about a 30% reduction in training time and thus, freezing layers should still be used with progressive reinitialisation.

The $reinit\_svhn$ condition was found to out-perform the the std_svhn condition by at least 1% on the SVHN dataset. This demonstrates that the improved performance gained from progressive reinitialisation is not limited to a single architecture or dataset.

The results of the $reinit\_aug$ and the std_aug conditions show no significant difference. Therefore, the freezing and reinitialisation done by progressive reinitialisation was not beneficial for training on this dataset when image augmentation was being used. However, it should be noted that progressive reinitialisation did not perform worse than the standard procedure and therefore, can still be used with augmentation, without decreasing the network's accuracy.

Because progressive reinitialisation does not improve the network's accuracy when image augmentation is used, it suggests that image augmentation provides one of the same benefits as progressive reinitialisation. To help identify what this benefit might be, we plotted the network's validation error for progressive reinitialisation vs. the standard training procedure when image augmentation was absent vs. present. Fig. 2 and Fig. 3 display these results for a single trial. Fig. 2 demonstrates that the std condition is over-fitting at around epoch 25 as its validation error begins to increase. The $reinit$ condition also begins over-fitting, however this is periodically counteracted (every 40 epochs) by reinitialisation which forces the later layers to relearn their weights. The weights that the later layers learn generally cause the validation error to reach a new minimum, reiterating that learning the weights of later layers after earlier layers is more effective for minimising the network's error. To our knowledge there is no other research suggesting that earlier layers of a neural network should be trained before its later layers and thus, future research is possible in this area.

Currently freezing and reinitialisation occurred every 40 epochs. Fig. 2 demonstrates that it does not take the whole 40 epochs for the network to reach its minimum validation error. This is especially noticeable at epoch 200 and 240 where it takes the network only a couple of epochs to reach the minimum error after freezing and reinitialisation had occurred. This demonstrates how the speed of this algorithm could be increased if freezing and reinitialisation occurred when the network's error converges at its minimum rather than naively occurring every 40 epochs.

Fig. 3 demonstrates that the neural network does not suffer from over-fitting when image augmentation is used. This is a well known principle, as image augmentation enlarges the training dataset such that the network cannot over-fit by learning the correct classification to the individual training images. We believe that image augmentation stops the later layers of the network from over-fitting so that they can continue effectively learning once the earlier layers have converged on their feature detectors. Therefore, explicitly forcing later layers of the network to learn their weights after earlier layers is not necessary when image augmentation is being used.

Because image augmentation boosted the performance of the CNN from 75% to 85%, image augmentation should definitely be preferred over progressive reinitialisation as a technique for increasing a network's accuracy. However, CNNs can be applied to tasks not involving images and thus, image augmentation is not always an option. It is in these cases where progressive reinitialisation might be preferred over image augmentation as a method to boost the performance of a CNN during training.

Finally, given that the progressive reinitialisation condition did not perform significantly different to the standard condition when image augmentation was present, it could still be beneficial to use progressive reinitialisation as this improves training time without sacrificing accuracy. For example, the training time for the $reinit\_aug$ condition was 2100 minutes, whereas the std_aug condition was 2242 minutes. This is about a 6% saving[1].

V. Conclusion

In conclusion, this article demonstrated that progressive reinitialisation can be used to increase the accuracy of various CNN architectures on CIFAR-10 and SVHN. Progressive reinitialisation involves periodically freezing the network's layers, starting from the beginning of the network and reinitialising all layers' weights that are not frozen. Experiments demonstrated that the freezing of layers did not produce the increase in accuracy, but was advantageous as it reduced training time by about 30%, whereas reinitialising layers was important for increasing the accuracy of the network. It was also demonstrated that the order of this reinitialisation was important, and that early layers should stop being reinitialised before later layers.

Progressive reinitialisation was also tested with image augmentation. In this experiment, progressive reinitialisation did not increase the network's accuracy significantly. We suggested that both progressive reinitialisation and image augmentation is beneficial because it allows later layers in the network to learn after earlier layers have finished learning. We identified that image augmentation was more beneficial than progressive reinitialisation. However, input to a CNN can be any type of data with spatial structure, not just images. In some cases, the application of augmentation is not clear and therefore, progressive reinitialisation might be useful as an alternative method for boosting your network's performance. Therefore, future research might investigate whether progressive reinitialisation is effective on other datasets, more specifically datasets which do not lend well to augmentation.

Acknowledgment

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the TITAN X GPU used for this research.

[1]This saving is not as substantial as the saving between the $reinit$ and the $reinit\_w/o\_freezing$ conditions because the augmentation introduced is a relatively expensive operation.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] A. Brock, T. Lim, J. Ritchie, and N. Weston, "Freezeout: Accelerate training by progressively freezing layers," *arXiv preprint arXiv:1706.04983*, 2017.

[3] B. Singh, S. De, Y. Zhang, T. Goldstein, and G. Taylor, "Layer-specific adaptive learning rates for deep networks," in *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*. IEEE, 2015, pp. 364–368.

[4] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[5] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision*. Springer, 2016, pp. 646–661.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[7] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: when to warp?" in *Digital Image Computing: Techniques and Applications (DICTA), 2016 International Conference on*. IEEE, 2016, pp. 1–6.

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[9] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.