Deeper Networks For Image Classification

A.Hasan

May 2020

With special thanks to L.Dijkshoorn

1 Introduction

1.1 Deep Learning

Research into image classification has been following a trend for the past five to ten years. In 2012, a new benchmark was set out with AlexNet [5], a CNN designed by Alex Krizhevsky, which won the 2012-ILSVRC competition with a ten percentage point gap on the runner up. This leap spurred research into the utilisation of deep nets for image classification and, after two years, VGG16, NIN and Inception were breaking new records [8]. In 2016, ResNet [4] was published and it pushed what was considered the limits of deep networks with an incredible 152 layers. By comparison, VGG-19 had 19 layers and GoogLeNet had 22.

The introduction of new techniques to overcome many issues faced by deeper networks eventually allowed deep nets to dominate image classification, and till today continues to be a large topic of research.

1.2 VGG

The VGG architecture was published as part of the ICLR 2015 conference and it proved that increased depth resulted in state of the art accuracy on ImageNet [2]. The proposed architecture features repeated max pooling layers after multiple 1x1 and 3x3 convolution layers. The multiple 3x3 layers allowed the model to capture the same effective receptive field as a single 7x7 filter while using significantly fewer parameters. This functionality allowed the network to go deeper while avoiding the problem of over fitting due to excessive parameters.

The single pixel convolution layer is designed to increase non linearity between layers through the rectification function. According to the initial paper [2], non linearity does help increase the overall accuracy of a model but it is also more important to capture spatial context. 1x1 layers are still being used to introduced non linearity in more recent models such as GoogLeNet [3].

1.3 ResNet

The idea for deep residual learning in image recognition was proposed in 2015 by [4] . The architecture aimed to tackle the vanishing gradient problem and, more importantly, the fact that overly deep nets result in a higher training error. This was surprising since the addition of identity layers still increased the training error, which could only mean that the identity mapping was not a trivial solution. To combat this, a residual block is introduced that learns the overall mapping with regard to the original input. This presented a perfect solution for the degradation problem in deep networks, and when combined with other parameter reduction solutions to prevent overfitting, the model becomes an extremely powerful classifier. ResNet went on to win the 2015 ImageNet challenge and set a high standard for future architectures.

The overall architecture of ResNet (ResNet-50) is very similar to VGG in that it employs the use of multiple 3x3 layers, but with high-way connections and less max pooling layers.

2 Model Description

In this paper, I will investigate the effectiveness of deeper CNN models for image classification. I have chosen to use the VGG-16 and ResNet architectures on the MNIST dataset and I will investigate any improvements that can be made on the original designs in terms of hyper parameter settings, pre-processing and pre-training.

2.1 Architecture

2.1.1 Training Environment and Hyper Parameters

1. Environment

   All models were built, trained and tested in python using the following packages

   (a) Tensorflow V2 for Nvidia

   (b) Keras (built into tensorflow)

   The intensive training process was carried out on the following hardware hosted by google compute engine

   (a) n1-standard-8 type machine (8vCPUs, 30GB Memory)

   (b) Intel Skylake CPU

   (c) 1 x NVIDIA Tesla V100

   Graphs were generated using matplotlib and seaborn.

2. Weights and Pre-Training

   Weights for each module were set to a random initialisation. This helped me fine-tune hyper-parameters later to ensure I would not be stuck in local minima during training. I also investigated using pre-trained weights to speed up training time, hoping that the weights would be closer to the optimal solution. To reduce training time, I made use of transfer learning to provide a starting point for weight initialisation. The weights were taken from https://github.com/fchollet who ported them from the original authors of VGG at Oxford under the creative commons license.

3. Hyper-parameters

   Most parameters were easily decided such as the loss function. Since this model aims to classify more than two classes, I chose to minimise categorical cross entropy during training.

   Preparing the model for training required some fine tuning for several parameters. I decided to use an SGD optimiser for this model as it was known for giving more accurate results [6]. The SGD optimiser was initialised with the following parameters

Param	Value
lrate	0.01
decay	0.001
momentum	0.9
nesterov	False

   This initially required some fine-tuning to prevent getting stuck in local minima and achieve good results.

   I then also used the default keras adam optimiser to observe any obvious differences in results. According to [7], it is possible that more than 23 per cent of scientists use an Adam optimiser when training, which is interesting if you consider that it is known to give more accurate results.

   Validation was used to train this model, with a fifth of the dataset reserved for this purpose.

2.1.2 VGG-16

1. Structure

   For the general structure of this model, I decided to use VGG-16. This can be identified as model D in the original paper [2]. The minimal gains in accuracy for deeper VGG models were not worth the complexity for the purpose of this experiment. It should also be noted, with regard to the dataset, that only ten classes needed to be identified. The last three dense layers are removed from the architecture defined above and replace with a dropout layer followed by a soft-max activated dense layer for 10 classes.

The benefits of this are two-fold. Firstly, it enabled me to use VGG to classify ten classes rather than the standard thousand. Secondly, It allowed me to use a unique input shape (32x32) which saved pre-processing time on the data-set.

2. Model Summary - Short

Layer	Parameters
VGG16 (Model)	14714688
Dropout	0
Dense	5130

Total parameters: 14,719,818

3. Training Training took place over 15 epochs with a batch size of 32. The figures 7 and 8 display the training accuracy and loss using both optimisers with both random weight initialisation and transferred weight initialisation. The models generally converged after a few epochs and then it became a matter of how accurate they could become with the remaining epochs.

4. Evaluation All variations of the model were testing against the test set to evaluate how well they generalise.

Optimiser	Weights	Train Accuracy	Train Loss	Test Accuracy	Test Loss
Adam	Random	0.9872	0.0399	0.9924	0.0243
SGD	Random	0.9884	0.0371	0.9914	0.0258
Adam	Weighted	0.9857	0.0658	**0.9807**	0.1523
SGD	Weighted	**0.9977**	**0.0076**	0.9969	**0.0116**

The table above clearly shows that, even with a few number of epochs, SGD results in the best training results and fairly high testing results. This is especially interesting because according to a medium article [7] almost 23 of deep learning tasks utilise Adam. It also confirms the conclusion of the paper "Gradient-based learning applied to document recognition" [1] in that SGD performs better than Adam with regard to accuracy.

2.1.3 ResNet

1. Structure

For the purpose of simplicity, I decided to use ResNet50 as my second model. ResNet50 is a 50 layer model that makes use of residual blocks in order to go deeper without facing issues like vanishing gradient. As before, slight modifications had to be made to use ResNet50 on MNIST due to the shape of the dataset and the number of classes.

A similar method was employed again to integrate with the MNIST dataset. The main ResNet50 model was added to a sequential layer without the final soft max dense layer. A single dropout layer is added followed by a soft max layer for ten classes, as shown in the figure above.

2. Model Summary

Layer	Parameters
ResNet50 (Model)	23587712
Dropout	0
Dense	20490

Total parameters: 23,608,202

Non-Trainable parameters: 53,120

3. Training

Training for ResNet50 took place over 15 epochs with a batch size of 32. Fig. 5 and 6. show the pattern of convergence of the model. It was also vital to use validation sets for training as it helped prevent overfitting (characterised by the large variation of accuracy for validation accuracy in comparison to training accuracy).

4. Evaluation

Optimiser	Weights	Train Accuracy	Train Loss	Test Accuracy	Test Loss
Adam	Random	0.9669	**0.1272**	**0.9840**	**0.0501**
SGD	Random	0.9335	0.3346	0.9711	0.0945
Adam	Weighted	**0.9688**	0.1466	0.9608	0.6885
SGD	Weighted	0.9247	0.3370	0.9692	0.1058

The model successfully converged in all cases. For this model, the Adam optimiser performed much better over 15 epochs than SGD. Random weight initialisation generally outperformed pre-trained initialisation.

2.2 Experimentation

I was unsatisfied with the result of ResNet50 so I attempted to boost the results by initially tweaking some parameters. I halved the learning rate of the SGD optimiser to 0.005 and enabled nesterov in the hope it would eventually produce better results. I also trained the model for 50 epochs to investigate how much of an improvement I could achieve. I managed to achieve a training accuracy of **0.9739233** and a loss of **0.0835**. This still does not compare to VGG-16 and I can only speculate that ResNet50 needed more fine-tuning of hyper-parameters to better learn the dataset being used.

2.3 Datasets

The dataset used for these experiments is MNIST [1], as sourced from the built in keras dataset loader, includes a training set of 60,000 samples and a testing set of 10,000 samples. MNIST is a subset derived from the much larger NIST. The samples were size-normalized to 28x28 and centered. The dataset is of handwritten digits and thus includes 10 classes of each digit.

2.3.1 Image Depth Considerations

The samples provided by MNIST are grayscale images with a single layer of pixel intensity values for each image. The model architectures used demand RGB images, or in the very least, three layers. There are few solutions that were considered. I investigated converting the images to RGB in pre-processing, however this often caused out of memory issues. I finally settled on a much more efficient solution. I duplicated the single layer three times to create the necessary input shape that the models could accept. While this is not a perfect solution, it worked fairly well based on preliminary results.

2.3.2 Image Size Considerations

The size of the MNIST dataset was an issue in this investigation. The standard size for both models used are much larger (224x224) than the default image sizes provided by MNIST (24x24). Resizing the samples to 224x224 was both illogical and computational infeasible given the limited resources I have to work with. Such a large scaling operation would lose valuable spatial data and severely affect model performance. I decided to resize the images to 32x32 and apply structural changes to the models to allow for training. The additional pre-processing of the images was acceptable since the scaling operation was not as large.

3 Conclusion

I would have liked to carry out more investigations into why ResNet50 did not perform as well as VGG-16 under these circumstances. Even after modifying hyper-parameters I could not reach an accuracy to compete with VGG-16. The only other options I can speculate are causing this are pre-processing of the dataset. It is possible the residual blocks do not interact well with the depth solution I provided for gray-scale images.

VGG-16 performed very well and I was able to generate fairly accurate models for handwritten digit recognition. For further work, I would like to attempt introducting highway connections to VGG-16 to see how much of an improve could be made. I was also able to explore the differences between different optimisers and generally concluded that SGD is more accurate while Adam is faster. The relevant code and trained models can be found at this GitHub repository

https://github.com/Dullaz/cw3

4 Appendix

References

[1] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[2] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[3] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[4] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[6] Liangchen Luo et al. *Adaptive Gradient Methods with Dynamic Bound of Learning Rate*. 2019. arXiv: 1902.09843 [cs.LG].

[7] Synced. *ICLR 2019: 'Fast as Adam Good as SGD'-New Optimizer Has Both*. Mar. 2019. URL: https://medium.com/syncedreview/iclr-2019-fast-as-adam-good-as-sgd-new-optimizer-has-both-78e37e8f9a34.

[8] Aramis et al. *ImageNet: VGGNet, ResNet, Inception, and Xception with Keras*. Apr. 2020. URL: https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/.

4.1 Figures

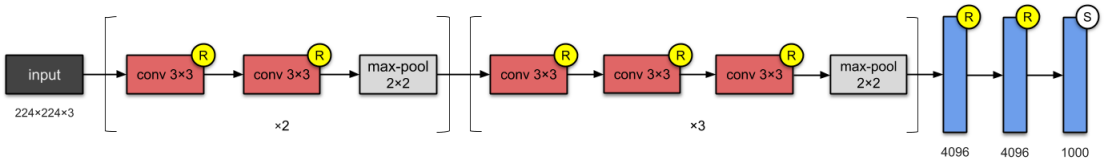

Figure 1: VGG-16 Structure

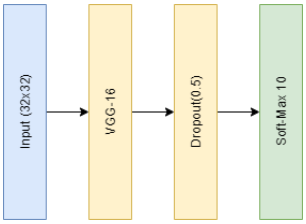

Figure 2: VGG Modification

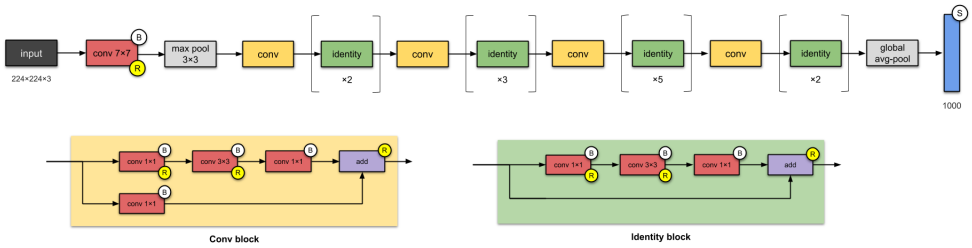

Figure 3: ResNet50 Structure

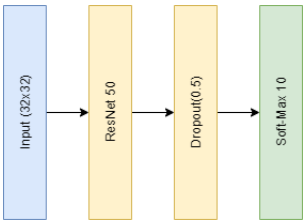

Figure 4: ResNet50 Modification

Figure 5: ResNet-50 Training

Figure 6: ResNet50 Loss

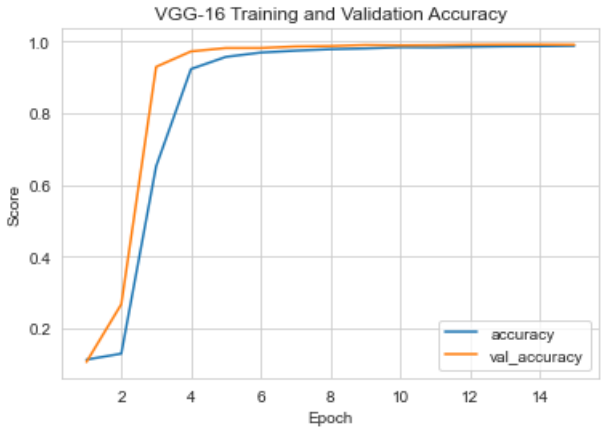

Figure 7: VGG-16 Training

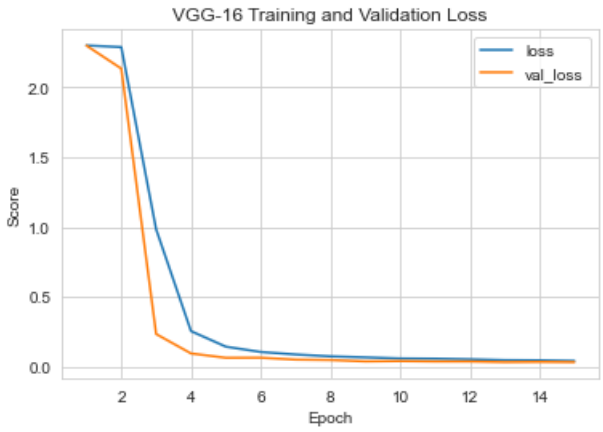

Figure 8: VGG16 Loss

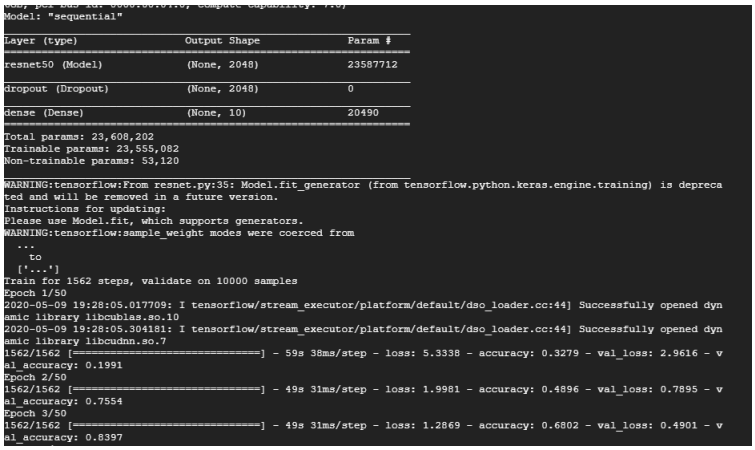

Figure 9: ResNet50 Runtime

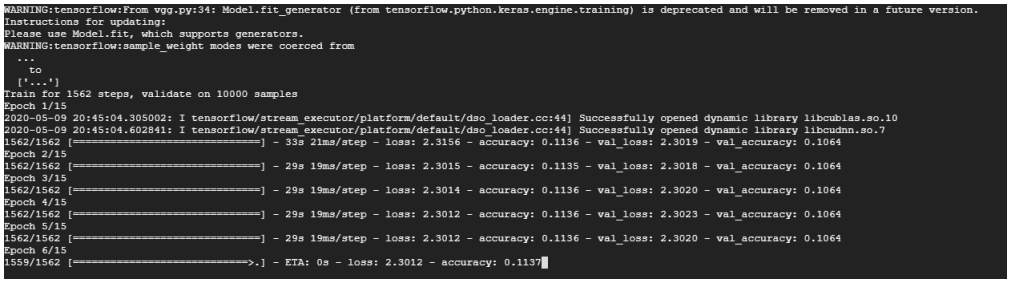

Figure 10: VGG Runtime