Deeper Networks for Image Classification

Oscar Wyatt

Abstract

Broadly, there has been a clear pattern in the architectures of the deep learning networks for image classification tasks in improving the performance of deeper networks while reducing the number of parameters to be optimised. I trained some of these networks and here examine the impact of the architectural differences on performance. Additionally, I propose a method of improving the convergence and performance of GoogLeNet by adding identity mapping to its Inception module.

1. Critical Analysis / Related Work

Using deep learning for image classification tasks has been an area of huge interest and research. This was triggered by the release of large annotated datasets[1][2][3] which obviated the need to source training data, thus making researching deep models easier. Furthermore, the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC), in which entrants would compete to get the best performance on the ImageNet dataset, has spurred innovation in the field[4].

A landmark network was Krizhevsky et al.'s AlexNet[5], which won the ILSVRC-2012 competition, outperforming the next best model by a significant degree (top 5 error of 15.3% to 26.2%). The key contribution was a deeper architecture with 8 weighted layers and some new features to improve performance and reduce training time. It showed that deeper networks can significantly improve performance and heralded a new era of research on deeper networks.

The authors assert that their network is limited by it's computational requirements and envisage that further improvements would come from faster GPUs and larger datasets. While more powerful GPUs does help experimentation, later work shows that network depth and architecture is more important to network performance. Their network has 60 million parameters and as such was prone to overfitting. They argued that Local Response Normalisation (LRN) improved generalisation and aimed to show this by demonstrating that including it in a simple network improved performance by 2%. They made use of ReLU to model a neuron's output and showed that it improved convergence time from around 35 epochs to fewer than 10 compared to tanh.

Some of the work that followed this made attempts to improve on AlexNet with minor architectural changes[6][7]. Most, however, did not investigate increasing the depth of the network. Simonyan et al.'s paper[8] introducing VGG was a seminal work that comprehensively investigated the effect of even deeper networks on image classification tasks. They compared networks with 11, 13, 16 and 19 weight layers and showed that as the networks got deeper, their error rate fell and thus demonstrated the importance of network depth to high performing networks.

One contribution was an experiment to ascertain the improvement of using LRN in a network deeper than that which Krizhevsky et al. tried it with. Simonyan et al. showed that on an 11 layer network the inclusion of LRN made a negligible impact on performance and as it increases the memory consumption and computational requirements, it could be omitted for deeper networks.

One shortcoming of the VGG architecture is the reliance on large convolutional filters, with the E variant (VGG19), having 8 layers with a filter size of 512. This means that VGG19 has 144M learnt parameters, storing the weights requires ~1GB of disk space and inference requires approximately 40 G-ops[9]. These high costs makes training the model and using it for inference very computationally expensive and power hungry, putting limitations on usage and deployment. Additionally, inference is slow which means it cannot be used on real time systems without added complexity.

Separately, some of these shortcomings were addressed in GoogLeNet[10] (V1). They were heavily inspired by Network In Network[11], in which Lin et al. used 1x1 convolutional layers to learn patterns across the channels of an image. The authors took this idea further and used it to find find ways towards solving two fundamental problems they'd identified in wider and deeper networks.

Firstly, a larger network has more parameters and thus may be prone to overfitting. One conceptually simple way to correct for this is to use a larger dataset – however this is laborious and expensive – and ideally the network architecture itself should foster good generalisation.

Secondly, larger networks require significantly more computational resources. Earlier networks such as AlexNet required circa 2.5 G-ops[9], which ballooned to around 40 G-ops in VGG19[9]. Simply increasing the depth and width without improving the efficiency would result in networks with no merit beyond academic interest. The authors explicitly set out to create a more efficient architecture, their network was designed to

keep within a computational budget of 1.5 billion multiply-adds (approximately a 26x decrease compared to VGG19) during inference – with the aim of producing a model that could be used on mobile and embedded devices. While their network was slightly deeper (22 weight layers) it was also wider. Their key innovation was the introduction of the Inception module, whose contribution was twofold.

Firstly, the Inception module uses 3x3 and 5x5 convolutional layers in order to learn spatial patterns at different dimensions. The module has a concatenation layer to combine these filters and create a single output that can represent the features at different scales, as well as patterns across the depth of the input.

Secondly, the Inception module made significant usage of dimensionality reduction. Based on the idea that even low dimensional embeddings can represent a lot of information about a higher dimensional space, they used sparse representations to keep the computational complexity of the network down and ensure the data is easier to process. Thus, they used 1x1 convolutions in the Inception module before the more expensive 3x3 and 5x5 to achieve this. This reduces the number of multiplication operations in a single module by tenfold, from ~120M to ~12M.

Another innovation is their introduction of auxiliary classifiers. Their concern was that with an (at the time) very deep network, backpropagating the gradients during training would be less effective and the vanishing gradient problem would arise. Their contribution was that in addition to the final softmax layer, two additional classifiers were included in the mid to late stages of the network (specifically, the 3rd and 6th inception models). They later found that the effect of these networks was minor and that they could remove one entirely without any impact on performance.

One limiting factor on deeper networks is that as the depth increases, they become harder to train. One obstacle is the vanishing/exploding gradients problem[12] where gradients (respectively) grow or fall very quickly, and the network is not able to improve it's performance. This had been largely addressed by normalised initialisation and intermediate normalisation layers[13][14][15][16]. However, another problem that was unsolved is degradation, in which the accuracy of deeper networks can become saturated and degrade. This is not caused by overfitting as the training error also suffers.

This problem motivated He et al. in their development of their ResNet[17]. They propose that the output of a residual layer can be of the form

$$y = F(x, \{W_i\}) + x$$

where y is the output, W_i the learnt weights for that layer and and x the input. This can be implemented with "shortcut connections" that could be performed with identity mapping. Identity mapping can be applied to fully connected or convolutional layers and does not require any extra parameters or computational complexity. In their work they show that they show that a 34 layer plain net has a higher training error than that of an 18 layer network. They argue that this is unlikely to be due to vanishing gradients and that instead deeper networks simply have significantly lower convergence rates. Their key contribution is that they then show that a 34 layer residual network has a lower training and test error than an 18 layer residual network. Thus, they demonstrate that, although the deeper network has improved performance, the addition of the identity mapping is responsible for the improvement. Additionally, their network does not require auxiliary classifiers.

While computational efficiency was not their chief motivating factor, their residual networks were significantly more efficient than VGG. ResNet 18 has a similar depth to VGG19 and has just ~11M parameters, a 10x improvement. Additionally, their improvements mean their 18 layer variant requires just ~3 G-ops at inference, while their 152 layer variant uses ~22 G-ops – 1.8x fewer than VGG19 while having 8x more weight layers[9].

2. Method / Model Description

In this paper, I use VGG, GoogLeNet and ResNet with two image classification datasets to evaluate the effectiveness of deeper models and their architectures on these tasks. The code for all three models was taken from publicly available repositories[18][19][20], with minor modifications to remove code to download trained weights, as all models were trained from scratch. Apart from one minor bug (which had little impact on performance, see 3.2.4) the code was faithful to the original papers.

2.1 Model Architecture

2.1.1 VGG19

VGG19 consists of 19 weight layers, for a total of 144M parameters. Sixteen of these are convolutional layers, interspersed with max pooling layers. These convolutional layers feed in to three fully connected layers before the final soft max layer. The convolutional layers consist of 64x64 filters in the first layer, increasing to 128, 256 and 512. The size and number of these large filters is the reason the network has so many more parameters and

stand in contrast to the much smaller filters in the other networks.

2.1.2 GoogLeNet

GoogLeNet consists of 22 weight layers, with a total of ~10M parameters. The network begins with two convolutional layers, each followed by a max pool layer. The main body has 7 Inception modules, interspersed with two max pool layers and then an average pool layer, dropout and then a final fully connected layer that feeds into a softmax classifier.

2.1.3 ResNet18

ResNet18 has 18 weight layers and a total of ~11M parameters. It consists almost entirely of convolutional layers, all with a filter size of 3x3. After the first convolutional layer, it is broken into 4 blocks – the first has 64 filters in each layer and the number of filters doubles in each block. After these blocks comes an average pooling layer, a fully connected layer and a softmax output.

2.2 Implementation and training

2.2.1 VGG

In these experiments, I used VGG19 with many of the same settings as per the original paper, summarised in the table below (asterisks indicating where the my implementation differed).

	MNIST	CIFAR10
Optimiser	SGD	SGD
Mini batch size	256	256
Momentum	0.9	0.9
Weight decay	0.0005	0.0005
Starting learning rate	0.01	0.01
Learning rate schedule	Decrease by 10x after plateau, patience 5*	Decrease by 10x after plateau, patience 5*
Augmentation	None*	Horizontal flipping, rotation, colour jitter, random resize cropping*

One significant hyperparameter for tuning was the patience value, which controls the number of epochs of plateaued test performance that should pass before the learning rate is decreased. This had a major effect on performance, experimentation showed that a higher patience resulted in better overall performance as the learning rate stayed higher for longer, allowing the network to escape local minima. Lowering the momentum made this effect more pronounced.

Patience	LR at best test error	Best test error %	Achieved at epoch
0	1.0×10^{-8}	14.05	26
1	1.0×10^{-7}	9.58	44
5	1.0×10^{-5}	6.73	88
10	0.001	5.96	97
15	0.001	6.12	99

2.2.2 ResNet18

In these experiments, I used ResNet18 with many of the same settings as per the original paper, summarised in the table below (asterisks indicating where the my implementation differed).

	MNIST	CIFAR10
Optimiser	SGD	SGD
Mini batch size	256	256
Start learning rate	0.1	0.1
Learning rate schedule	Decrease by 10x after plateau, patience 1	Decrease by 10x after plateau, patience 1
Augmentation	None*	Horizontal flipping, rotation, colour jitter, random resize cropping*

2.2.3 GoogLeNet

In these experiments, I used GoogLeNet with many of the same settings as per the original paper, summarised in the table below (asterisks indicating where the my implementation differed).

	MNIST	CIFAR 10
Optimiser	SGD	SGD
Mini batch size	256	256
Start learning rate	0.1	0.1
Learning rate schedule	Decrease by 4% every 8 epochs	Decrease by 4% every 8 epochs
Momentum	0.9	0.9
Augmentation	None*	Horizontal flipping, rotation, colour jitter, random resize cropping*

3. Experiments

3.1 Datasets

3.1.1 MNIST

The Modified Institute of Standards and Technology (MNIST) database is a dataset of handwritten digits (0 to 9). It has 60,000 training images and 10,000 test images; each is 28x28 pixels and greyscale (one channel)[2]. With only 10 classes and a relatively small training set, it is much faster to train than larger datasets like ImageNet, which has 1000 classes and consists of 1.2M images of varying sizes[1]. Thus, MNIST is a good choice for quickly and easily investigating a network and understanding whether changes are improving performance.

Additionally, MNIST has long been used as a benchmark for comparing image classifiers[2] and as such, it makes a good choice for quickly understanding how well a network performs compared to earlier work.

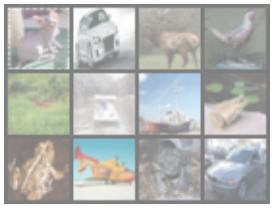

Figure 1: Sample CIFAR10 images

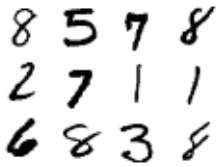

Figure 2: Sample MNIST images

3.1.2 CIFAR 10

The Canadian Institute for Advanced Research dataset[3] (CIFAR-10) consists of images of objects, each of which belongs to one of ten classes (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks). It was constructed from the 80 million tiny images dataset. There are 50,000 training and 10,000 test images, with 5000 images from each class in the training set and 1000 from each class in the test set.

The images are all 32x32 pixels and full colour and as they are larger and have 3 channels, it is more computationally expensive to train a network on CIFAR 10 than MNIST. Furthermore, the nature of the images is quite different, this makes the task harder for two reasons.

Firstly, the classes are at different scales (airplane, bird) and the objects within them are different sizes and are seen from different viewpoints. For example, the network must not overfit and learn that airplanes always have a blue background.

Secondly, some of the classes (deer and horses, cars and trucks) can be quite similar to one another and so the network must be able to distinguish detailed differences between them. For example, the network must be able to generalise beyond a horse and deer being a brown foreground on a green background.

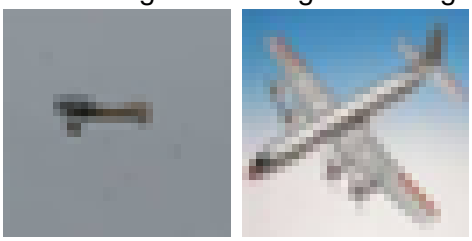

Figure 3: Images of the airplane class at different sizes, from very different angles and with different backgrounds.

Figure 4: Images from the deer and horse classes (respectively). Both are visually more similar to one another than in figure 3 yet are from different classes.

In all, this makes CIFAR10 a much harder test for a network and makes it easier to distinguish the effect of the different architectures.

3.2 Experiments and testing results

3.2.1 Evaluating differences in model performance and training convergence on MNIST dataset

In this experiment, I trained VGG19, ResNet18 GoogLeNet with MNIST to compare their performance and observe how their architecture affects their convergence. I chose the 19 layer version of VGG and the 18 layer version of ResNet as it means all three networks have similar numbers of layers.
I wanted to find what difference the architecture made to the network performance while keeping the number of layers (more or less) constant. All figures are top 1 error percentage.

When comparing the models in this fashion, the improvements in architecture are readily apparent. The difference in rate at which each network's test error falls demonstrates the importance of architectural differences in training speed and overall accuracy.

The difference in convergence speed between VGG and the other models is likely due to the much larger number of parameters and shows why reducing the number of parameters to learn is desirable.

Remarkably, ResNet achieves a test error of just 1.82% after the first epoch. This indicates that even though GoogLeNet has slightly fewer parameters, residual nets are able to converge much faster. The fact that both networks achieve a similar final performance underlines the impact of the residual layers on convergence.

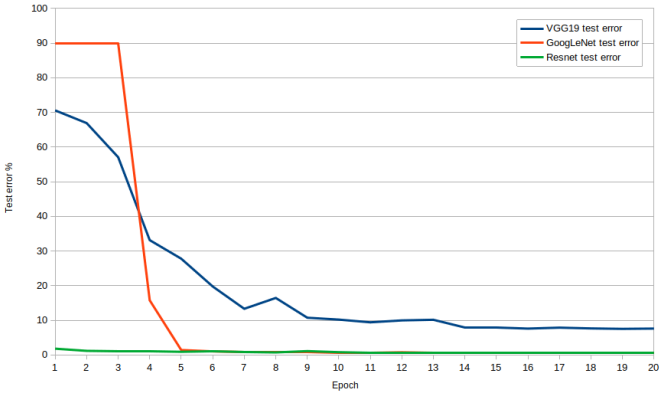

Figure 5: VGG19, ResNet18 and GoogLeNet trained on MNIST

3.2.2 Evaluating differences in model performance and training convergence on CIFAR10 dataset

In this experiment, I trained the same three models (from initialisation) on CIFAR 10. As the task was much harder and early experiments showed performance was poor, the dataset was augmented with horizontal flipping, rotation, colour jitter and random resize cropping. The best hyperparameters from training on MNIST were used as performing a grid search would have been too computationally expensive. Further work could look into this although it's doubtful it would significantly alter the results.

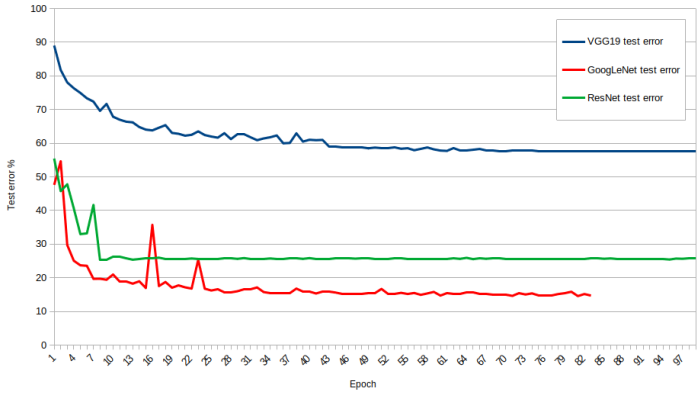

Figure 6: VGG19, ResNet18 and GoogLeNet trained on CIFAR10

Here, the differences between the three models are even more pronounced. All three take longer to converge, with VGG stabilising after 70 epochs, GoogLeNet on it's 81st and ResNet still converging significantly faster on it's 9th epoch. There is a much larger difference in performance, highlighting the impact of architecture.

Additionally, we can see that the harder task reveals a much larger difference in performance between GoogLeNet and ResNet, with a 10.73% gap in test error between the models. This refutes the simplistic hypothesis that could be taken from 3.2.1 that there is not a great deal of difference between the two models. It thus suggests, therefore, that GoogLeNet's architecture is better able to represent more complex image classes. Furthermore, the difference between the test and train error in GoogLeNet is constant after convergence, while in ResNet the train error continues to fall while the test error remains constant. Together, these suggest that GoogLeNet is better able to generalise and represent invariant features of each class better than ResNet.

3.2.3 Architectural improvements to GoogLeNet

While experimenting with GoogLeNet, I found a known bug[21] in the PyTorch implementation of GoogLeNet. This meant that the 5x5 convolutional filters in it's Inception modules were in fact 3x3. As I had already trained the model, I wanted to see what impact changing it to 5x5 would have on convergence and performance.

Additionally, having observed the improvement in convergence afforded by residual layers, I wanted to asses the extent to which making Inception modules residual would improve convergence or performance. By combining both these experiments, I'd be able to investigate the impact of both on convergence and performance.

Firstly, we observe from Figure 3 that both variants with residuals converge much faster. This confirms my hypothesis that adding identity mapping to the Inception modules would result in faster convergence. Due to time constraints on implementation, the identity mapping was only added to modules whose outputs were the same size as the inputs, further work could investigate whether adding it to all layers with downsampling where necessary (as per [17]) would further improve convergence. Further work could also look into whether this could benefit fine tuning a

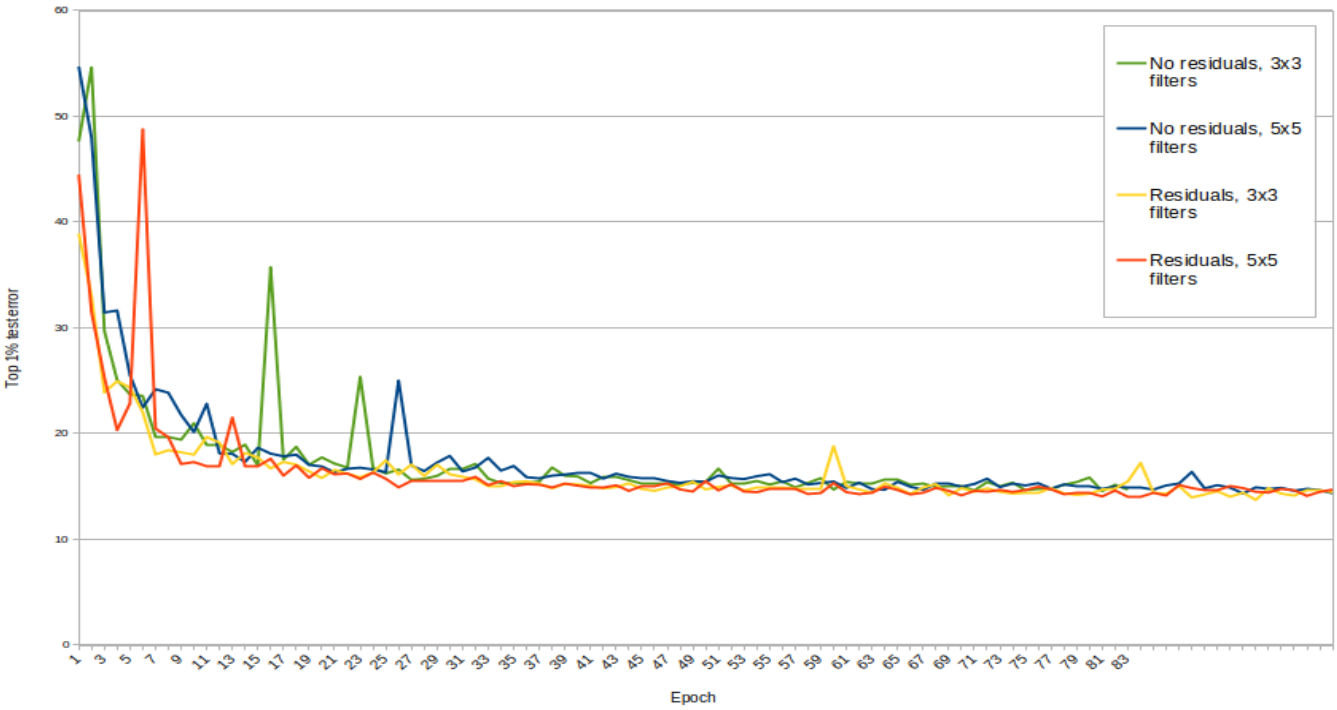

Figure 7: Effect of using residual layers and changing the Inception module filter size in GoogLeNet. Trained on CIFAR10

pretrained model by making it faster and thus easier to experiment with.

Secondly, we observe that the variant with 3x3 filters and residual layers performs better than both variants (3x3 and 5x5) without residuals. This suggests that adding identity mapping does help improve classification performance.

Thirdly, we see that although the 5x5 variant with residuals is the best performing variant, it achieved this score on it's 158th epoch, taking significantly longer to converge than the other variants. This suggests a downside of the 5x5 filter sizes in that convergence is slower and the final performance improvement may not justify the additional computational cost in training.

Variant	**Best top 1 error %**
No residuals, 3x3 filters	14.56
No residuals, 5x5 filters	14.33
Residuals, 3x3 filters	13.73
Residuals, 5x5 filters	13.39

3.2.4 Effect of GoogLeNet's Auxiliary Classifiers

I also wanted to understand how GoogLeNet's auxiliary classifiers altered training convergence and whether they aided convergence. The network with the auxiliary classifiers had a lowest test error of 14.56% while without, it's best performance was 14.69% - a difference of only 0.13%. There is scant difference between how well the networks converge during training, suggesting they have no impact on aiding convergence. This experimental result, combined with that of 3.2.3 shows that adding identity mapping to a deeper network has a much larger impact on convergence and performance than adding auxiliary classifiers. Thus, auxiliary classifiers should be removed to reduce the computational cost during training. Identity mapping, which requires no additional parameters should be used instead.

4. Conclusions

In this work, I have shown that although deeper models do improve performance, other architectural decisions also have a large impact on performance and that they should be carefully considered. It was demonstrated that adding identity mapping to GoogLeNet improved performance and convergence, thus confirming that it can be applied to other architectures. Furthermore, I was able to show that increasing the filter size in each Inception module improved performance only slightly and with an increase in computational cost. This serves to show that further work to improve the efficiency of deeper networks could benefit from studies on the choice of filter size and that networks designed for mobile or low powered devices could use smaller filters with only a minor performance hit. Additionally, I showed that it had a much larger impact on performance than auxiliary classifiers.

Bibliography

1: 2021. [online] Available at: <https://www.image-net.org/static_files/papers/imagenet_cvpr09.pdf> [Accessed 15 May 2021].
2: Yann.lecun.com. 2021. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [online] Available at: <http://yann.lecun.com/exdb/mnist/> [Accessed 15 May 2021].
3: Cs.toronto.edu. 2021. CIFAR-10 and CIFAR-100 datasets. [online] Available at: <https://www.cs.toronto.edu/~kriz/cifar.html> [Accessed 15 May 2021].
3: Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. and Berg, A.C., 2015. Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3), pp.211-252.
5: Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, pp.1097-1105.
6: Zeiler, M.D. and Fergus, R., 2014, September. Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.
7: Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R. and LeCun, Y., 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229.
8: Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
9: Canziani, A., Paszke, A. and Culurciello, E., 2016. An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678.
10: Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).
11: Lin, M., Chen, Q. and Yan, S., 2013. Network in network. arXiv preprint arXiv:1312.4400.
12: Bengio, Y., Simard, P. and Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), pp.157-166.
13: LeCun, Y.A., Bottou, L., Orr, G.B. and Müller, K.R., 2012. Efficient BackProp BT-Neural Networks: Tricks of the Trade. Neural Networks: Tricks of the Trade.
14: Glorot, X. and Bengio, Y., 2010, March. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings
15: Saxe, A.M., McClelland, J.L. and Ganguli, S., 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv:1312.6120.
16: He, K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).
17: He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
18: GitHub. 2021. pytorch/vision. [online] Available at: <https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py> [Accessed 15 May 2021].
19: GitHub. 2021. pytorch/vision. [online] Available at: <https://github.com/pytorch/vision/blob/master/torchvision/models/googlenet.py> [Accessed 20 April 2021].
20: GitHub. 2021. pytorch/vision. [online] Available at: <https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py> [Accessed 21 April 2021].
22: GitHub. 2021. I find some error in googlenet.py ? · Issue #906 · pytorch/vision. [online] Available at: <https://github.com/pytorch/vision/issues/906> [Accessed 15 May 2021].