LayerOut: Freezing Layers in Deep Neural Networks

Kelam Goutam, S. Balasubramanian, Darshan Gera, and R. Raghunatha Sarma

Department of Mathematics and Computer Science
Sri Sathya Sai Institute of Higher Learning, Prashantinilayam, India

kelamgoutam.sssihl@gmail.com
{sbalasubramanian, darshangera, rraghunathasarma}@sssihl.edu.in

Abstract. Deep networks involve a huge amount of computation during the training phase and are more prone to overfitting. To ameliorate these, several techniques such as DropOut, DropConnect, Stochastic Depth, and BlockDrop have been proposed. These techniques regularize a neural network by dropping nodes, connections, layers, or blocks within the network. However, their functionality is limited in that they are suited for only fully connected networks or ResNet-based architectures. In this paper, we propose LayerOut - a regularization technique. This technique can be applied to both fully connected networks and all types of convolutional networks such as, VGG-16, ResNet etc. In LayerOut, we stochastically freeze the trainable parameters of a layer during an epoch of training. Our experiments on MNIST, CIFAR-10, and CIFAR-100 show that LayerOut generalizes well and reduces the computational burden significantly. In particular, we have observed up to 70% reduction, on an average, and in an epoch, in computation and up to 2% enhancement in accuracy as compared to baseline networks.

Keywords: LayerOut, DropOut, DropConnect, Stochastic Depth, BlockDrop

1 Introduction

The recent trend in the deep learning community is to use deeper neural networks [8] [24] to solve real-life problems such as image classification [13] [23], language translation [15] [27], object detection [5] [18], speech recognition [6] [2] etc. However, deeper neural networks have been empirically found to display the undesirable characteristic of being prone to overfitting. Further, the computational load of training a deeper network is by no means trivial. This makes the deployment of deeper models in real-time environment such as interactive applications on mobile devices, autonomous driving etc. a challenging task. Few of the methods to reduce overfitting include early stopping [4], regularization techniques like L2 weight normalization [16] and local response normalization (LRN) [19]. An efficient solution to reduce variance and thereby overfitting would be to use an ensemble of architectures for decision making. However, explicit ensembling adds to computational burden.

Some of the popular methods such as DropOut [25], DropConnect [29], Stochastic Depth [9], and BlockDrop [30] were introduced that performed implicit ensembling by randomly dropping nodes, connections, layers, or blocks within the network during each training iteration. They address overfitting effectively and reduce the computational overload.

In this paper, we propose "LayerOut" - a regularization technique that is modeled along the lines of the methods mentioned above. LayerOut randomly freezes the layers of the neural network instead of dropping the nodes, connections, layers or blocks. LayerOut also reduces computational cost during training. We demonstrate using our LayerOut a considerable improvement in generalization capability of the architecture on the popular benchmark datasets MNIST [14], CIFAR-10, and CIFAR-100 [12].

2 Related Work

The principle behind "LayerOut" draws inspiration from the following works: DropOut [25], DropConnect [29], Stochastic Depth [9], and BlockDrop [30].

2.1 DropOut

Hinton et al. introduced a regularization technique called as dropout [25] to avoid overfitting the neural networks. The idea of dropout is to drop the nodes in the network along with their connections randomly during the training phase. This random dropping of the nodes during the forward propagation prevents the weights of the layers to converge to an identical position, thus reducing the interdependent learning among the nodes. Dropout forces the nodes in the network to learn better robust features independently. Assuming the model has $\mathbf{H}$ hidden nodes, then with dropout, there are $\mathbf{2^H}$ possible models while training as each node is capable of being dropped out. During the train phase, the nodes are kept with probability $\mathbf{p}$ and the nodes are dropped with probability $(\mathbf{1-p})$. During the test phase the entire model is considered, but to account for the number of dropped nodes, each activation is compensated by a factor of $\mathbf{p}$.

2.2 DropConnect

Li Wan et al. generalized the dropout technique where they dropped the connections of the nodes with the probability of $(\mathbf{1-p})$ instead of dropping the nodes. Dropconnect [29] makes the fully connected network a sparsely connected network by dropping the connections between the nodes randomly. Dropconnect helps to train many models during training compared to dropout since the number of connections will always be more than the number of nodes in the model. Since only a subset of the connections is dropped, almost all the nodes of the network are partially active during the training phase, in contrast to dropout where most of the nodes become inactive. In summary, we can say that dropout makes any fully connected network "thinner" and dropconnect makes it "sparse".

In recent studies conducted, it was observed that dropout and dropconnect fail to generalize well beyond fully connected neural networks [11]. The reason for this is dropout and dropconnect were designed to be used with fully connected

networks. However, with the success of convolutional neural networks (CNN) [13] which share weights across different layers and reduces the number of parameters to be trained, newer architectures started replacing fully connected networks with large number of convolutional layers. Both dropout and dropconnect were primarily implemented for fully connected layers and as a result, the modern network did not reap many benefits out of these techniques. In case of convolutional layers, the number of parameters is less, so they need a small regularization as such and that is provided using batch normalization [10].

2.3 Stochastic Depth

Gao Huang et al. introduced stochastic depth [9] which allow training a shallow network during the train phase and a deeper network during the test phase. In this technique, during training, a subset of layers are randomly dropped/bypassed using identity connections. The deeper networks have two major concerns. The first is that of vanishing gradients [3] where the gradients become very small during the backward propagation and the earlier layers learn very slow. The second problem is analogous to the vanishing gradient problem, known as diminishing feature reuse [26] during forward propagation. The features computed by the earlier layers start getting washed out as we progress deeper during the forward propagation due to repeated convolutions with weight matrices. This makes learning better features by later layers difficult. These problems were eliminated by ResNets [8] using their skip connections. These skip connections make ResNets a better option for stochastic depth. Training the models using the stochastic depth reduces the test error and the training time. The reason for the reduction in test error could be attributed to the fact that the network trained using stochastic depth behaves as an ensemble network where the networks are of different depths and the reduction in depth of the network helps in avoiding both vanishing gradient and diminishing feature reuse problems. The reason for the reduction in training time can be attributed to the fewer number of computations to be performed during forward and backward propagation due to the dropping of layers.

2.4 BlockDrop

Tushar Nagarajan et al. introduced BlockDrop [30] as a reinforcement learning technique. In this paper, they used a pretrained ResNet-110 model and trained a policy network which outputs the probabilities for making a binary decision of keeping or dropping the blocks of the pretrained ResNet model. The policy network is trained using curriculum learning where the policy network is rewarded if it uses fewer blocks of the pretrained ResNet model while maintaining the accuracy produced by the ResNet model. Unlike stochastic depth where layer dropping happens only during training, in BlockDrop, blocks of ResNet are also dropped during test phase. BlockDrop naturally reduces computational burden.

3 Motivation

Dropout and dropconnect are primarily useful for fully connected layers. Stochastic depth and BlockDrop assume that underlying architecture is a ResNet archi-

tecture. They leverage on the idea that unrolling ResNet enumerates a plethora of paths from input to output and, some of them may be redundant. The redundant ones are dropped thereby minimizing overfitting and reducing computational load. But restricting these advantages only to specific architectures such as ResNet and fully connected architectures will limit their applicability. In this work, we propose a new technique called LayerOut - a regularization technique. LayerOut can be applied to both fully connected networks and all type of convolutional networks such as, VGG-16, ResNet etc. LayerOut also reduces computational cost during training as it does not compute gradients for the frozen layers during the backpropagation. In the following section, we describe the LayerOut technique in detail.

4 Proposed Method - LayerOut

LayerOut proposes a simple modification to the backpropagation algorithm by requiring that only a few randomly chosen layers be updated. For a given architecture and a given epoch during train phase, a uniform random vector of probabilities is sampled wherein the i^{th} component represents the probability of binary decision to freeze or update layer i of the architecture. When a decision to freeze a layer is made, the parameters corresponding to that layer are omitted from being updated during the backpropagation. We refer to these decisions as ***freezing strategy***.

4.1 Formal Framework

Let K be the number of layers in a given architecture. Let $v = (v_1, v_2, ..., v_i, ..., v_K)$ be a random vector where v_i's are independently sampled from uniform distribution over $[0, 1]$. We call v as freeze probability vector. Let $s = (s_1, s_2, ..., s_i, ..., s_K)$ be a Bernoulli random vector where s_i denote whether layer i is frozen or updated. s_i assume value 1 (i.e layer i is frozen) with probability v_i. Let π denote the joint probability distribution of s_1, s_2, ..., s_K. Then,

$$\pi(s) = \prod_{k=1}^{K} v_k^{s_k}(1 - v_k)^{1-s_k} \tag{1}$$

$\pi(s)$ is the joint probability of freezing K layers in the architecture in an epoch during training. $\pi(s_i)$ is the probability of layer i being frozen without any parameter update. In other words $1 - \pi(s_i)$ is the probability that layer i is updated in an epoch during training. During the forward propagation the entire network is fully functional but during the backward propagation, only those layers which are not frozen are updated.

4.2 Effectiveness of LayerOut

Due to random layer freezing, the hidden nodes are not activated deterministically but non-deterministically. Let $W_e^{[i]}$ denote the weight matrix in layer i of the network in the beginning of epoch e during training. Then,

$$W_e^{[i]} = \begin{cases} W_{e,updated}^{[i]} & with\ probability\ 1 - v_i \\ W_{e-1}^{[i]} & with\ probability\ v_i \end{cases}$$

where,

$$W_{e,updated}^{[i]} = W_{e-1}^{[i]} - \alpha \bigtriangledown W_{e-1}^{[i]}$$

Here, $W_{e-1}^{[i]}$ is the weight matrix in layer i at the end of epoch $e-1$ during training. Further, α is the learning rate and $\bigtriangledown W_{e-1}^{[i]}$ is the loss gradient with respect to $W_{e-1}^{[i]}$. Since the realization of $W_e^{[i]}$ in epoch e is random, the realization of hidden activation in layer i in epoch $e+1$ denoted by $h_{e+1}^{[i]}$ is random. That is, assuming bias vector is zero,

$$h_{e+1}^{[i]} = \begin{cases} W_{e,updated}^{[i]}\ h_{e+1}^{[i-1]} & with\ probability\ 1 - v_i \\ W_{e-1}^{[i]}\ h_{e+1}^{[i-1]} & with\ probability\ v_i \end{cases}$$

This implicit source of randomness or noise means that every layer has to learn to be more robust to a lot of variation in its input. This implies LayerOut acts as a regularizer. This is an argument similar to why batch normalization [10] is considered as a regularizer.

Further, because layers are randomly frozen and hence do not participate consistently in backpropagation, the amount of computation time during back propagation significantly falls down. This reduces computational burden during training.

LayerOut stands out from the dropping techniques due to following reasons.

- During the test phase, there is no need to scale the activations as all of the layers participate in the forward propagation.
- LayerOut can be used for both convolutional layer as well as a fully connected layer.

For example, in Fig. 1, the first and the third hidden layers of the fully connected network (FC) are frozen simultaneously. Thus all the layers participate during the forward propagation but during the backpropagation, the gradients for the first and the third hidden layers are not computed. As a result, these layers are not updated. Note that the concept of freezing the layers can be applied to any of the deep neural networks (CNN and FC).

5 Experimental Set Up

In order to evaluate the proposed technique we designed the following experiments:

- Train baseline without LayerOut
- LayerOut since start
- LayerOut after warm-up with random generation of freeze probability vector v

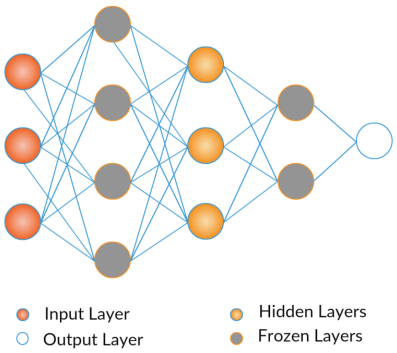

Fig. 1. Illustration of LayerOut.

- LayerOut after warm-up with manual fixing of v

The first experiment refers to baseline model without LayerOut. In the second experiment, LayerOut is incorporated from the first epoch. The third and fourth experiments bring in LayerOut after a few epochs. We have set this warm-up period to 20 epochs. The third and fourth experiments differ in the way freeze probability vector v is generated: random in the former case and manual in the latter case. It is to be noted that in the second experiment we report results only for random v since, in general, LayerOut after warm-up performed better than LayerOut since start with or without random generation of v. The goal of all these experiments is to demonstrate that LayerOut influences performance positively and also significantly reduces training time. We do not consider a complete LayerOut of all layers as this would freeze all parameters resulting in zero learning and wasted forward computations.

All implementations are carried out in PyTorch [17]. We set weight decay to 5×10^{-4}, momentum to 0.9 and stochastic gradient descent (SGD) [20] as the optimizer. Further, the input data is augumented using the following techniques:

- Pad by 4 pixels on all the sides
- Random crop
- Random Horizontal Flip
- Standard normalization

During test phase, we only adopted standard normalization.

Specific details of experiments conducted on different datasets are described in the subsequent subsections.

5.1 MNIST

The MNIST [14] dataset consists of grayscale images of handwritten digits of size 28×28. The dataset contains 60,000 training images and 10,000 test images belonging to 10 classes i.e. digits from 0 to 9. For this dataset, we designed as

baseline a shallow network. This network consists of 5 convolution layers and 2 fully connected layers. We omitted the use of any pooling layer and compensated for it by performing a strided convolution on 2^{nd} and 4^{th} layer. The activation, Relu [1] is used after every layer. The architecture of the shallow network is shown in Fig 2.

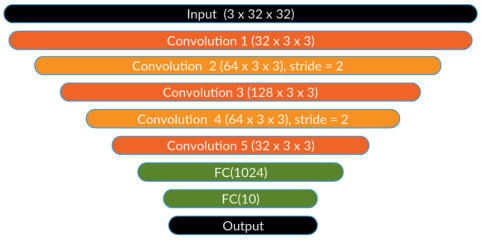

Fig. 2. Shallow network architecture used to train on MNIST dataset

The model was trained for 100 epochs with the learning rate as 0.01. The weights were initialized using Xavier initialization.

With regard to experiment 4, we consider two cases: (i) fixing v in decreasing order as $v = (0.9, 0.7, 0.6, 0.5, 0.4, 0.3, 0.1)$ and (ii) fixing v in increasing order as $v = (0.1, 0.3, 0.4, 0.5, 0.6, 0.7, 0.9)$. In the former case, earlier layers are more likely to be frozen than the deeper layers. The latter case is the counterpart of the former case. To differentiate between these two cases, we name them as LayerOut-efreeze and LayerOut-dfreeze respectively. Our implementations of experiments on MNIST dataset are available at MNIST code [1].

5.2 CIFAR-10 and CIFAR-100

CIFAR-10: The CIFAR-10 dataset [12] consists of 32×32 colour images from 10 classes, where each class has 5000 training images and 1000 test images. In total, CIFAR-10 consists of 50000 images for training and 10000 images for testing.

For baseline architectures we consider VGG-16 and ResNet-110. In our implementation of VGG-16, we have modified 7 x 7 convolutions in first fully connected layer to 1 x 1 convolutions. With abuse of nomenclature, we still call the modified VGG-16 as VGG-16 for convenience. Apart from aforesaid data augumentations, we also used random erasing [31].

For both baseline architectures weights were initialized using He initialization [7]. Relu [1] was the preferred activation function. VGG-16 model was trained for 300 epochs with the initial learning rate as 0.1. The learning rate was reduced by a factor of 10 on the 150^{th} and the 225^{th} epoch. The ResNet-110 model was trained for 164 epochs. The initial learning rate for ResNet-110 was set to 0.01 and after training for 4 epochs, it was changed to 0.1 and then it was reduced by a factor of 10 during 82^{nd} and 123^{rd} epoch. The other hyperparameters chosen

[1] https://github.com/Goutam-Kelam/LayerOut/tree/master/MNIST

for ResNet-110 are same as those defined in [8]. The batch size for VGG-16 and ResNet-110 were fixed to be 128 and 32 respectively.

We consider two cases here for experiment 4 as in MNIST. The freeze probability vectors for LayerOut-efreeze and LayerOut-dfreeze with regard to baseline VGG-16 were set to $v = (.9, .8, .8, .7, .7, .6, .6, .5, .5, .4, .4, .3, .3, .2, .2, .1)$ and its reverse respectively. In case of ResNet-110 we randomly generated the probability vector v, as manual fixation of probabilities was practically infeasible. Our implementations of experiments on CIFAR-10 dataset are available at CIFAR-10 code [2].

CIFAR-100: The CIFAR-100 dataset [12] consists of 32×32 RGB images from 100 classes, where each class has 500 train images and 100 test images. The dataset consists of 50000 images for training and 10000 images for testing. The experimental setup for CIFAR-100 dataset is exactly the same as in case of CIFAR-10 dataset. Our implementations of experiments on CIFAR-100 dataset are available at this CIFAR-100 code [3].

6 Results and Analysis

Note: In Tables 1-5, the last column denotes the percentage of parameters frozen per epoch on an average during training.

6.1 MNIST

S.No	Architecture	Accuracy (%)	Parameters Frozen (%)
1	Baseline model	98.87	-
2	LayerOut since start	98.87	31.20
3	LayerOut with warm-up	99.01	28.50
4	LayerOut-dfreeze	99.03	34.90
5	**LayerOut-efreeze**	**99.08**	22.30

Table 1. Performance on MNIST dataset

Table 1 tabulates the performance of LayerOut against the baseline shallow network in terms of accuracy and percentage of reduction in trainable parameters. Clearly, LayerOut after warmup (rows 3, 4 and 5 in the table) outperforms baseline and LayerOut since start significantly. Specifically, Layerout-efreeze dominates over all other cases with 99.08% accuracy. Though LayerOut-dfreeze also exhibits a relatively good performance, we see that it does not generalize as well as LayerOut-efreeze. We interpret this as being evidence of the undesirability of freezing the learning for more complex features in the data that might require a continuous learning.

Further, the number of trainable parameters in our shallow network is about 0.49 million. LayerOut-dfreeze freezes on average about 0.17 million parameters (34.9%) per epoch and LayerOut-efreeze freezes on average about 0.11 million

[2] https://github.com/Goutam-Kelam/LayerOut/tree/master/CIFAR-10
[3] https://github.com/Goutam-Kelam/LayerOut/tree/master/CIFAR-100

parameters (22.3%) per epoch. This is a significant reduction which demonstrates that LayerOut eases training and improves accuracy.

6.2 CIFAR-10

		Accuracy (%)		
S.No	**Architecture**	**With R.E**	**Without R.E**	**Parameters Frozen (%)**
1	Baseline model	93.79	93.75	-
2	LayerOut-dfreeze	94.05	94	69.91
3	**LayerOut-efreeze**	**94.07**	**94.02**	53.66

Table 2. Performance of VGG-16 with and without Random Erasing (R.E) on CIFAR-10

		Accuracy (%)		
S.No	**Architecture**	**With R.E**	**Without R.E**	**Parameters Frozen (%)**
1	Baseline model	93.66	93.57	-
2	LayerOut since start	94.97	95.07	49.12
3	**LayerOut with warm-up**	**95.01**	**95.27**	47.34

Table 3. Performance of ResNet-110 with and without Random Erasing (R.E) on CIFAR-10

Table 2 and Table 3 tabulates the performance of LayerOut against the baseline VGG-16 and ResNet-110 on CIFAR-10 respectively. As in the case of MNIST, LayerOut improves the baseline network's accuracies. Specifically in VGG-16, LayerOut-efreeze achives 94.07% accuracy with random erasing and 94.02% accuracy without random erasing. For the baseline ResNet-110, manual setting of freeze probabilities is infeasible. Hence we report results only with respect to first three experiments. LayerOut after warm-up achives the best accuracy of 95.27% without random erasing.

The total number of learnable parameters in our VGG-16 implementation for CIFAR-10 is about 33.64 million. LayerOut-efreeze and LayerOut-dfreeze achieves 53.66% and 69.91% reduction in number of trainable parameters respectively. Similarly ResNet-110 has about 1.70 million trainable parameters and LayerOut on an average, achieves a reduction by 48.23% per epoch.

6.3 CIFAR-100

Table 4 and Table 5 tabulates the performance of LayerOut against the baseline VGG-16 and ResNet-110 on CIFAR-100 respectively. LayerOut clearly outperforms baseline's accuracies. Specifically in VGG-16, LayerOut-efreeze achieves the best accuracy of 73.81% with random erasing. For baseline ResNet-110, LayerOut after warm-up achieves the best accuracy of 77.57% with and without random erasing.

The total number of learnable parameters in our VGG-16 implementation for CIFAR-100 is about 34 million. LayerOut-efreeze and LayerOut-dfreeze achieves 53.90% and 70.05% reduction in number of trainable parameters respectively.

		Accuracy (%)		
S.No	**Architecture**	**With R.E**	**Without R.E**	**Parameters Frozen (%)**
1	Baseline model	73.37	72.83	-
2	LayerOut-dfreeze	73.54	73.13	70.05
3	**LayerOut-efreeze**	**73.81**	**73.28**	53.90

Table 4. Performance of VGG-16 with and without Random Erasing (R.E) on CIFAR-100

		Accuracy (%)		
S.No	**Architecture**	**With R.E**	**Without R.E**	**Parameters Frozen (%)**
1	Baseline model	76.48	73.33	-
2	LayerOut since start	77.22	77.14	50.07
3	**LayerOut with warm-up**	**77.57**	**77.57**	47.66

Table 5. Performance of ResNet-110 with and without Random Erasing (R.E) on CIFAR-100

Similarly ResNet-110 has about 1.70 million trainable parameters and LayerOut on an average, achieves a reduction by 48.87% per epoch.

The above report emphasizes the fact that (i) LayerOut generalizes much better than the baseline and (ii) requires far fewer backward computations thereby achieving a significant reduction in number of trainable parameters on an average in an epoch.

We also compared LayerOut with ResNet-110, BlockDrop and Stochastic Depth. Table 6 tabulates these comparison and illustrates that LayerOut overwhelms these state-of-the-art models on CIFAR-10 and CIFAR-100 datasets.

		Accuacy(%)	
S.No	**Architecture**	**CIFAR-10**	**CIFAR-100**
1	ResNet-110	93.20	72.20
2	BlockDrop	93.60	73.70
3	Stochastic Depth	94.75	75.02
4	**Our Method - "LayerOut"**	**95.27**	**77.57**

Table 6. Comparison of our method with ResNet-110, BlockDrop, and Stochastic Depth

7 Conclusion and Future Work

In this paper, we proposed "LayerOut", a simple yet effective method that freezes the layers randomly at every epoch during the training. It generalizes well and eases computational burden. In comparison to recent techniques like DropOut, DropConnect, Stochastic Depth, BlockDrop, LayerOut reports better accuracy and significant reduction in number of trainable parameters during training. LayerOut can be incorporated for both fully connected neyworks and various type of convolutional networks. The freeze probabilities can be set randomly or manually. We observed that freezing deeper layers degrades accuracy. This is

because deeper layers extract complex features in a hierarchical way and hence require more continual learning.

In this work the proposed method is tested on standard benchmark datasets such as MNIST, CIFAR-10 and CIFAR-100. We would like to extend our work to other useful applications like ImageNet, face recognition that need very deep networks [21][22][28]. In this scenario, our work can definitely ease training and maintain accuracy. Further, we would like to inspect the effectiveness of our proposed method on sequential models, such as multi-layered LSTM and GRU. Sequential models are hard to train and more so if it is multi-layered. Layer freezing can simplify training multilayered sequence models

Acknowledgment

We dedicate our work to our founder chancellor, Bhagawan Sri Sathya Sai Baba and the Department of Mathematics and Computer Science, SSSIHL for providing us with the necessary resources needed to conduct our research.

References

1. Agarap, A.F.: Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375 (2018)
2. Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., Bengio, Y.: End-to-end attention-based large vocabulary speech recognition. In: Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on. pp. 4945–4949. IEEE (2016)
3. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks **5**(2), 157–166 (1994)
4. Caruana, R., Lawrence, S., Giles, C.L.: Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In: Advances in neural information processing systems. pp. 402–408 (2001)
5. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 580–587 (2014)
6. Graves, A., Mohamed, A.r., Hinton, G.: Speech recognition with deep recurrent neural networks. In: Acoustics, speech and signal processing (icassp), 2013 ieee international conference on. pp. 6645–6649. IEEE (2013)
7. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
9. Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K.Q.: Deep networks with stochastic depth. In: European Conference on Computer Vision. pp. 646–661. Springer (2016)
10. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
11. Jansma, H.: Dont use dropout in convolutional networks. (Aug 2018), https://towardsdatascience.com/dont-use-dropout-in-convolutional-networks-81486c823c16

12. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
13. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
14. LeCun, Y.: The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/ (1998)
15. Luong, M.T., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025 (2015)
16. Ng, A.Y.: Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In: Proceedings of the twenty-first international conference on Machine learning. p. 78. ACM (2004)
17. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., et al.: Automatic differentiation in pytorch (2017)
18. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. pp. 91–99 (2015)
19. Robinson, A.E., Hammon, P.S., de Sa, V.R.: Explaining brightness illusions using spatial filtering and local response normalization. Vision research **47**(12), 1631–1644 (2007)
20. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2016)
21. Russakovsky, O., Su, Z., Karpathy, A., et al.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision **115**(3), 211–252 (2015)
22. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 815–823 (2015)
23. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229 (2013)
24. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
25. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
26. Srivastava, R.K., Greff, K., Schmidhuber, J.: Highway networks. arXiv preprint arXiv:1505.00387 (2015)
27. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in neural information processing systems. pp. 3104–3112 (2014)
28. Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1701–1708 (2014)
29. Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R.: Regularization of neural networks using dropconnect. In: International Conference on Machine Learning. pp. 1058–1066 (2013)
30. Wu, Z., Nagarajan, T., Kumar, A., et al.: Blockdrop: Dynamic inference paths in residual networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8817–8826 (2018)
31. Zhong, Z., Zheng, L., Kang, G., Li, S., Yang, Y.: Random erasing data augmentation. arXiv preprint arXiv:1708.04896 (2017)