Dense2Net: A Hybrid Res2Net/DenseNet Architecture

Tanner Songkakul, Nathan Starliper
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27603
`tpsongka@ncsu.edu, nstarli@ncsu.edu`

Abstract

Deep neural networks are used for a variety of computer vision tasks, but are susceptible to the vanishing gradients as the number of layers increases. A solution to this issue is to create residual connections in the network which allows for a residual mapping to deeper layers in the network. In this work, we create a novel architecture, Dense2Net, by combining 2 pre-existing neural network architectures, DenseNet and Res2Net. DenseNet achieves high performance by connecting each layer to every future layer in the network, thus creating a dense network which encourages feature reuse and gives implicit deep supervision. Res2Net augments bottleneck blocks in existing residual network type architectures by replacing the 3x3 convolution with a split-filter-concatenate residual network to increase granularity and multi-scale resolution. To build Dense2Net, Res2Net modules were integrated into the bottleneck blocks inside DenseNet. The Dense2Net model was trained and tested on CIFAR100. We show that our Dense2Net model achieves nearly 6% relative performance gains to a standard DenseNet model.

1 Introduction

Modern deep neural networks utilize many layers to achieve high performance on classification or other computer vision tasks [10]. However, as these networks rely on gradient descent via backpropagation to optimize their output, many layers can lead to these gradients being very small, limiting the ability of the neural network to learn [14]. This is referred to as the vanishing gradient problem.

Many current and previous state of the art convolutional neural networks (CNNs) directly connect early and later layers to avoid this vanishing gradient problem. Research on this subject has led to an ongoing evolution of these types of architectures. An early realization of this technique was Highway Networks, which use parameterized bypass paths and gating units for training of 100 layer networks [15]. ResNet, similarly utilizes non-parameterized "shortcut connections" to feed layer outputs forward to later layers [5]. Variations on ResNet include Inception-ResNet [17], which uses split-transform-merge functionality to perform customized filtering on split versions of the input and concatenates them together, and ResNext [18], which introduces cardinality to create aggregated transforms consisting of split-transform-merge operations utilizing identical modules for each transform path. An orthogonal modification to these and other residual architectures is Res2Net which replaces the 3x3 convolutions within the bottleneck blocks of these systems with multiple channels of hierarchical filter blocks [3].

Another realization of feed-forward architecture is DenseNet, in which each layer is directly connected to every future layer [7]. Dual Path Networks (DPN) utilize features of ResNet and DenseNet to increase the ability of the network to utilize both new feature exploration and feature re-usage [1]. Deep Layer Aggregation (DLA) combines layers of the neural network into aggeregated blocks through various structures to fuse resolution, scale, or features [19].

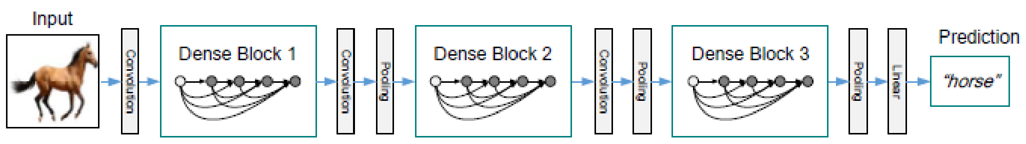

Figure 1: DenseNet architecture.

In this work, we combine the DenseNet and Res2Net approaches to create an new architecture, which we refer to as Dense2Net. Theoretically, this architecture combines the benefits of both implementations, with strengthened feature propagation and feature reuse from DenseNet and multi-scale feature extraction advantages from the incorporation of Res2Net modules. The Res2Net modules are incorporated by directly substituting for 3x3 convolutions in each of the bottleneck layers within the layers of the DenseNet. This work is motivated by the fact that Res2Net has been used to augment ResNet, ResNeXt, and DLA, but not DenseNet [3]. This work also investigates potential for performance improvements from adding group convolutions and squeeze-and-excitation blocks to the Res2Net modules inside of Dense2Net.

2 Background & Related Works

2.1 DenseNet

The DenseNet architecture (Figure 1) maximizes information flow between network layers by connecting all layers together in feed forward fashion such that each layer has concatenated input from all preceding layers, creating $L(L+1)/2$ connections for a network with L layers [7].

Traditional ResNets can be described by Equation 1, where H describes a non-linear transformation and the output of the previous layer $l-1$ is summed to create the output at layer l:

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1} \tag{1}$$

DenseNet instead concatenates feature-maps of all preceding layers as input to layer l (Equation 2):

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]) \tag{2}$$

The basic DenseNet architecture uses DenseLayers consisting of a batch normalization [8], a rectified linear unit [4], and a 3x3 convolution . The DenseNet is comprised of groups, referred to as DenseBlocks (Figure 2), containing multiple sets of DenseLayers. These DenseBlocks are connected by Transition Layers, which consist of a batch normalization [8], a 1x1 convolution layer, and a 2x2 average pooling layer [11]. The Transition Layers are necessary to down-sample feature maps such that they can concatenated. Densenets also have a growth rate hyperparameter k, which determines the number of channels in each input layer [7]. DenseNets can be modified by adding bottlenecks and compression. The bottleneck contains a batch normalization [8], rectified linear unit [4], and 1x1 convolution which reduces the quantity of input feature maps [16]. The compression is within the transition layer, where the number of output feature maps is reduced by compression factor θ<1. A DenseNet with both bottleneck and compression is referred to as Densenet-BC.

DenseNets have a few advantages due to their densely connected nature. One is that a DenseNet requires less parameters than a similarly complex ResNet because DenseNet layers are narrow and distinguishes between new and previously acquired information. Another advantage is that the connections between layers mean that each layer can access gradients from previous layers as well as the original input, giving implicit supervision. A final advantage is regularization from densely connected layers can reduce overfitting effects. In classification tasks on CIFAR10 and CIFAR100 [9], DenseNet showed similar or better performance than other competing methods (Figure 3) [7].

2.2 Res2Net

The Res2Net module improves multi-scale processing for multi-scale representation by incorporating a new module which improves multi-scale resolution at a granular level. This is achieved by replacing the 3x3 convolution filter with a Res2Net module. In a Res2Net module (Figure 4), the input is split into s subsets, then connected in a residual hierarchy of 3x3 convolutions [3]. Every subset except

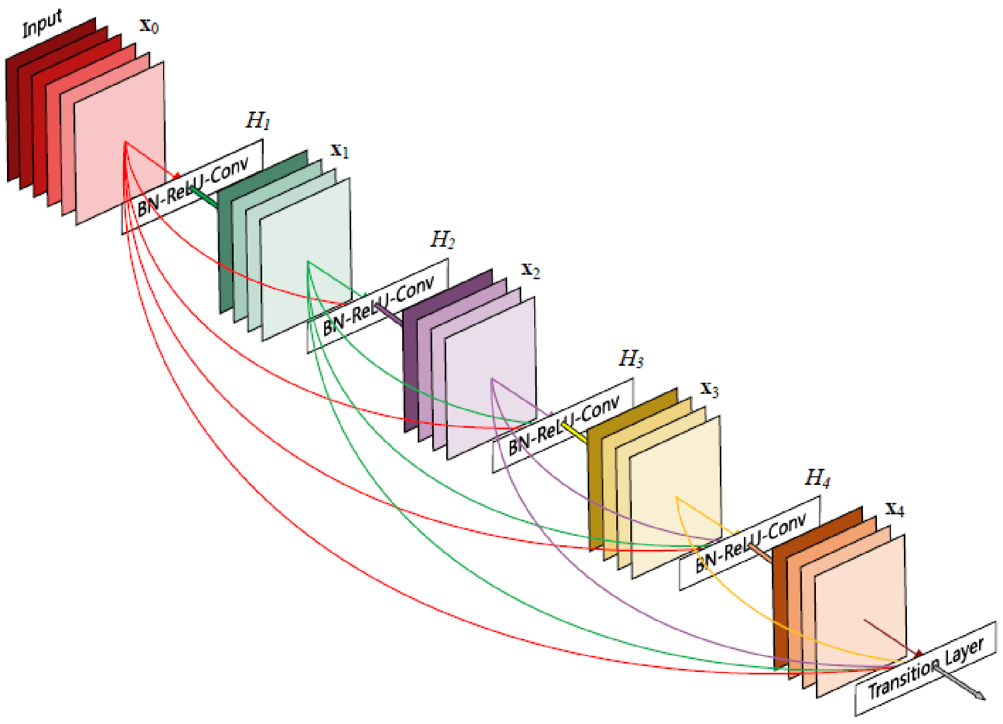

Figure 2: Structure of a 5 layer dense block with growth rate k=4.

Method	Depth	Params	C10	C10+	C100	C100+
Network in Network [22]	-	-	10.41	8.81	35.68	-
All-CNN [32]	-	-	9.08	7.25	-	33.71
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57
Highway Network [34]	-	-	-	7.72	-	32.39
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73
ResNet [11]	110	1.7M	-	6.61	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58
	1202	10.2M	-	4.91	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07
	28	36.5M	-	4.17	-	20.50
with Dropout	16	2.7M	-	-	-	-
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33
	1001	10.2M	10.56*	4.62	33.47*	22.71
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**

Figure 3: Comparison of DenseNet architecture performance on Cifar10 (C10) and Cifar100 (C100) vs other techniques. DenseNet was also tested on augmented Cifar10 (C10+) and Cifar100 (C100+) datasets.

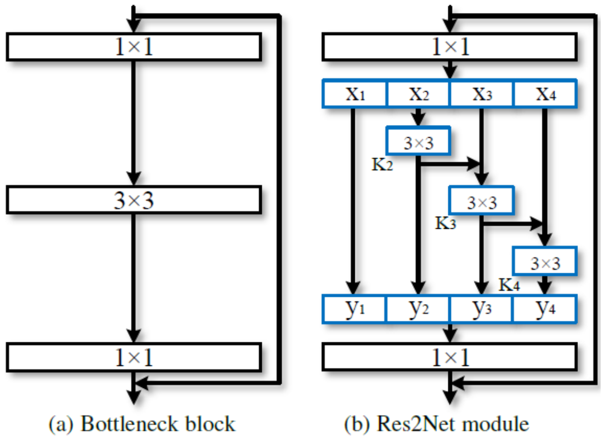

Figure 4: Comparison between a bottleneck block in ResNet and a Res2Net module with scale=4.

	Params	top-1 err.
Wide ResNet [48]	36.5M	20.50
ResNeXt-29, 8c×64w [43] (base)	34.4M	17.90
ResNeXt-29, 16c×64w [43]	68.1M	17.31
DenseNet-BC (k = 40) [20]	25.6M	17.18
Res2NeXt-29, 6c×24w×4scale	24.3M	16.98
Res2NeXt-29, 8c×25w×4scale	33.8M	16.93
Res2NeXt-29, 6c×24w×6scale	36.7M	**16.79**
ResNeXt-29, 8c×64w-SE [19]	35.1M	16.77
Res2NeXt-29, 6c×24w×4scale-SE	26.0M	16.68
Res2NeXt-29, 8c×25w×4scale-SE	34.0M	16.64
Res2NeXt-29, 6c×24w×6scale-SE	36.9M	**16.56**

Figure 5: Classification accuracy on CIFAR100 of Res2NeXt over ResNet, DenseNet-BC, and ResNeXt.

subset x_1 is passed through a 3x3 convolution, with the filtered output of subset $i-1$ is add to subset x_i as an input to convolution K_i (Equation 3):

$$\mathbf{y}_i = \begin{cases} \mathbf{x}_i & i = 1 \\ \mathbf{K}_i\left(\mathbf{x}_i + \mathbf{y}_{i-1}\right) & 1 < i \leq s \end{cases} \tag{3}$$

The 3x3 convolution outputs are then concatenated, and then passed through a 1x1 convolution to fuse information at different scales [3]. The hyperparameter of scale s determines how many subsets the input is initially split into. Group convolutions and squeeze-and-excitation blocks [6] can also be added into the Res2Net module.

The Res2Net module can be considered orthogonal to many existing architectures, so it can be used within the bottleneck blocks in many architectures. In their implementation, Gao et al integrated the Res2Net module into ResNet, ResNext, and DLA to create Res2Net, Res2Next, and Res2Next-DLA [3]. These augmented modules showed improvements in performance in object detection, semantic segmentation, instance segmentation [3]. It also showed reduced error classification on CIFAR100 as compared to other neural networks (Figure 5). However, Res2Net has not been integrated with DenseNet due to computational limitations [3].

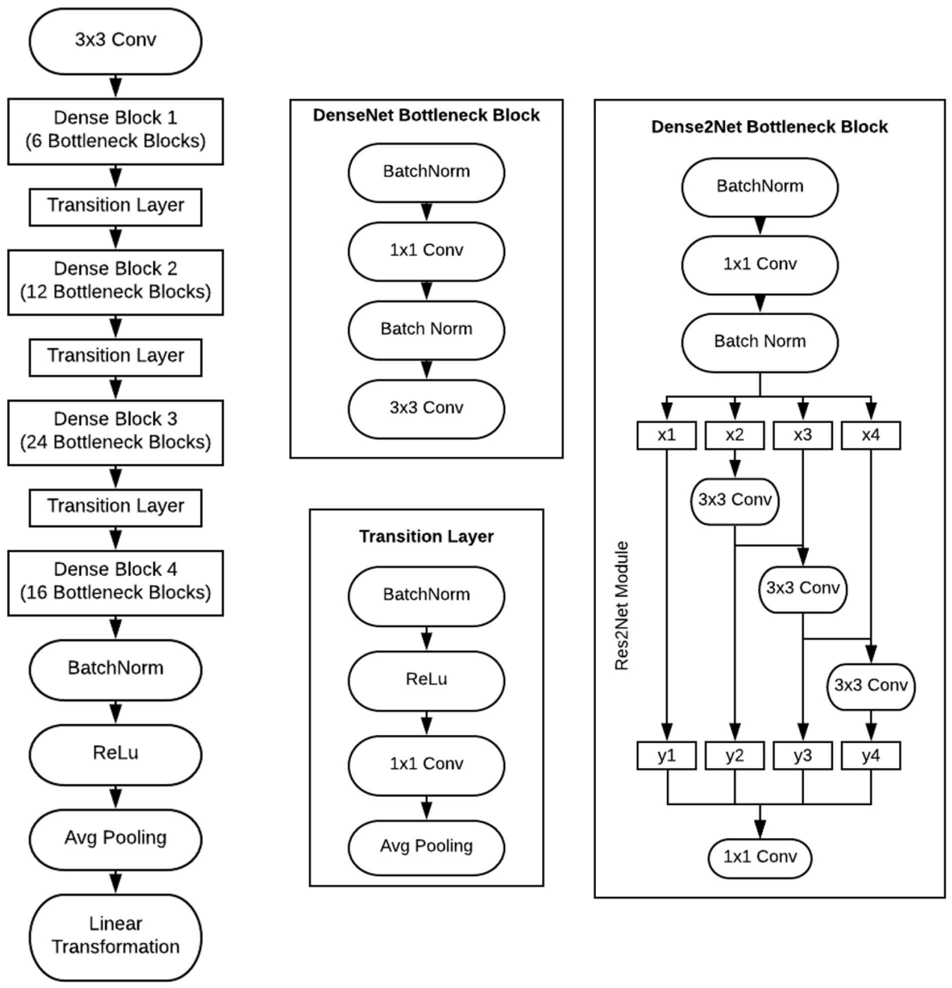

Figure 6: The structure of the Dense2Net 121 model used for the majority of experiments as well as the underlying transition layers and bottleneck blocks.

3 Dense2Net

3.1 Dense2Net Architecture

The Dense2Net architecture (Figure 6) is relatively straightforward. In DenseNet, each bottleneck block in within the DenseLayers comprising each DenseBlock contains a 3x3 convolution. In Dense2Net, this 3x3 convolution is replaced by the Res2Net hierarchical residual network of 3x3 convolutions followed by a 1x1 convolution to merge features and match dimensions between Dense2Net Bottleneck blocks. This means each DenseLayer in the Dense2Net architecture contains a Res2Net block. Dense2Net otherwise contains equivalent 1x1 convolution, ReLU, batch normalization, pooling, and transition layers as the standard DenseNet.

3.2 Implementation

Dense2Net was implemented using the Pytorch framework. A Res2Net module [13] was integrated into a pre-existing DenseNet implementation [12]. The DenseNet121 architecture consisting of 6 layers in the DenseBlock 1, 12 layers in DenseBlock 2, 24 layers in DenseBlock 3, and 16 layers in DenseBlock 4 was used for the majority of tests due to limitations in GPU memory. The network was trained using SGD with weight decay 0.0001, momentum 0.9, and batch size of 128 for 100 epochs on a few different GPUs. We also explored automated learning rate reduction, over-fit detection, and usage of group convolution and SE blocks.

4 Experiments

4.1 Experimental Setup

We run all of our experiments on the CIFAR-100 dataset which consists of 100 classes of images and is widely used for benchmarking new computer vision models and techniques [9]. We split the dataset into disjoint train and validation sets using PyTorch Torchvision's built in dataloader. We first ran all experiments for 100 epochs. However, due to limited time and computational resources, we implemented early stopping for our later experiments after noticing over-fitting on our models. We also initially ran a step-wise learning rate decay schedule where learning rate is decreased by a factor of 10 at epochs 30 and 60. However, to increase efficiency of runs, we implemented a learning rate decay schedule based on performance plateauing. This was motivated by the fact that the model was plateauing sooner than 30 epochs and substantial jumps in performance were observed after decreasing the learning rate, as evident in Figure 5. We also implemented a learning rate decay schedule which monitors the validation loss and decays the learning rate if a new minimum loss is not detected for 10 epochs.

4.2 Model Configurations

We experiment with various model configurations implementing different state of the art techniques and models to analyze the affect of these techniques on performance. Our baseline for comparison is a vanilla DenseNetBC model (without the Res2Net block) without data augmentation. The results for the baseline model are shown in figures 5. We use this baseline DenseNetBC model with growth rate $k = 32$ as the backbone architecture for the rest of our experiments. We compare the baseline DenseNetBC model with our Dense2Net model described previously which uses the Res2Net block inside of the bottleneck block of each dense layer. After noticing that our model was over-fitting quite a bit, we also experimented with other techniques described in the following subsections. The various model configurations and results are shown in Table 1.

4.2.1 Data Augmentation

Data augmentation has been shown to reduce overfitting of most neural network models and was used to artificially increase the size of the data set. We use the standard augmentations (mirroring/shifting) widely used in the deep learning field. We show that data augmentation has one of the most substantial effects on the network.

4.2.2 Cutout

Cutout is a special type of data augmentation which randomly masks square regions of input images during training. Cutout has been shown to significantly enhance the performance of many models tested and also perform specifically better than dropout for model regularization [2]. We use cutout with number of holes set to 1 and length of squares set to 8 pixels which were shown to give the best performance for CIFAR 100 in [2].

4.2.3 Dense2Net Scale

As described previously, one of the hyperparameters of the Res2Net block is the scale. The scale parameter represents the number of hierarchical levels of convolutions performed inside of the Res2Net block. In theory, a higher scale should provide a higher level of granularity and thus the ability to extract details at many more scale intervals. Our default scale is set to 4. We also show results for scales of 2 and 8. Scale dimension is visualized in figures 2.2 and 3.1 (right).

4.2.4 Cardinality

Cardinality was first introduced as a new dimension in neural network models by Xie et. al [18] and is the basis for the state of the art and widely popular ResNeXt model. Cardinality uses the idea of grouped convolutions which has been shown to increase model performance. This is due to the effect of the filter groups to learn with a block-diagonal structure sparsity on the channel dimension. This enforces the filters to be learned in a more structured way and therefore induces a type of regularization on the network (which in turn should reduce over-fitting). We experiment

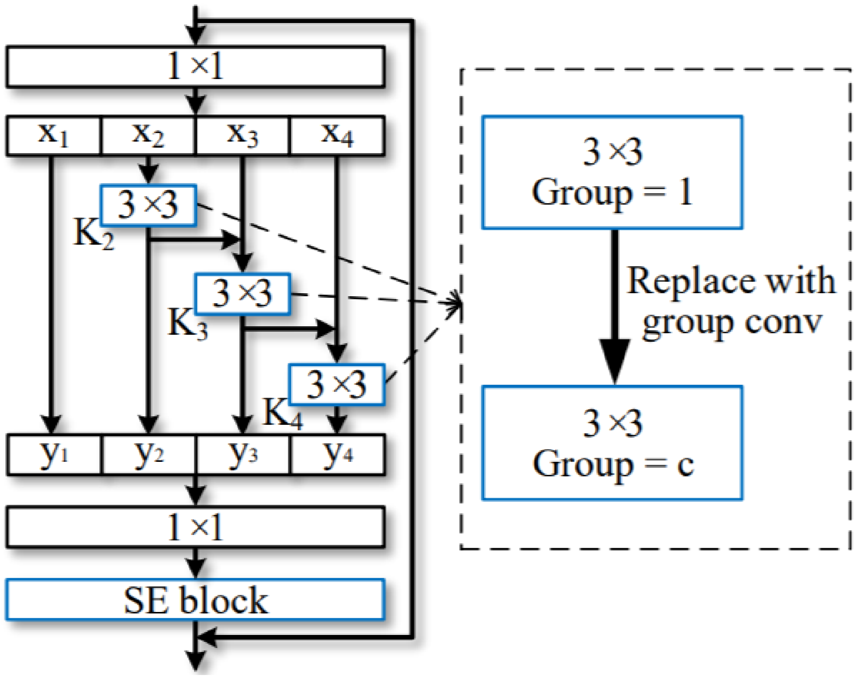

Figure 7: Res2Net block including group convolutions (cardinality) and the SE block.

with enforcing cardinality inside of the Dense2Net bottleneck block for the convolutions at each scale. We evaluate the effect of setting the cardinality to 32 (which is equal to our growth rate). The implementation of cardinality is visualized in Figure 4.2.4.

4.2.5 Squeeze and Excitation Block

We experiment with introducing a squeeze and excitation (SE) block [6] into our network. The SE block is inserted at the very end of our Dense2Net bottleneck block as shown in figure 4.2.4. The SE block enables feature calibration so that the network can emphasize informative features while suppressing less useful ones. The SE layer essentially weights the various filters produced by our convolutions at the various scales. We hypothesize that the SE layer will be particularly useful for our application because, due to the increase in scale representation, there is a good chance we are creating even more less useful information. A weighting scheme can help the network to learn the more important features/scales by weighting the output filters of the Res2Net block appropriately.

5 Results and Analysis

The final results of each of the model configurations are given in Table 1. It is clear that the two biggest factors in performance improvement from DenseNet to Dense2Net are data augmentation and the replacement of the standard 3x3 convolution of each dense layer with the Res2Net Module. Our Dense2Net implementation with data augmentation has a 5.83% relative increase in validation accuracy over the standard DenseNet with augmentation. The use of data augmentation gives a 9.99% relative increase in accuracy in our Dense2Net model. It also seems as though scale may have a small impact on performance accuracy with higher scale correlating to higher validation accuracy. We hypothesize that larger images would see even more of a benefit from increasing scale (CIFAR-100 is fairly low resolution at 32x32 pixels and therefore contains less feature scales). Figures 5 and 5 show the accuracy and loss curves for the DenseNet and Dense2Net models respectively. Clearly the Dense2Net model narrows the gap between the training and validation accuracies and losses. Surprisingly cutout, cardinality, and SE did not have a significant impact on the performance of our models. This is contrary to our initial hypothesis that these regularization techniques should decrease validation error, especially since our performance is mainly being limited by overfitting. Clearly overfitting is the issue because we are getting very high training accuracy while the validation accuracy stagnates.

One glaringly obvious aspect of our results is the fact that our baseline results are quite a bit worse than the results reported in both the DenseNet and Res2Net papers [7], [3]. According to the results presented by DenseNet in figure 2.1 our baseline DenseNet model with augmentation should be achieving error rates of about 19.25%, however we are seeing about 28% (see table 1). We are unsure of why there is a discrepancy between our results and the paper's results. Our baseline models are complete replicates of everything the two papers did. There could be some difference in weight initialization, however both the default PyTorch method and the DenseNet paper use the weight initialization introduced in [4], so this can be ruled out. Further examination of the baseline models should be carried out to determine the cause of these differences.

Despite the discrepancies in the performance of the baseline models, we show that incorporating the Res2Net block in DenseNet improves the performance against our baseline DenseNet model. We propose that if the discrepancies in our baseline models can be worked out, we should be able to achieve performance higher than that given in [7].

Table 1: Results of the experiments on CIFAR-100 (Top-1 accuracy)

Model Config	**Valid Acc (%)**	**Train Acc (%)**
DenseNet No Augmentation	65.41	99.98
DenseNet	71.91	98.7
Dense2Net No Augmentation	68.96	99.98
Dense2Net	75.85	99.95
Dense2Net $scale = 2$	75.26	99.35
Dense2Net $scale = 8$	76.1	98.77
Dense2Net with Cutout	75.23	98.33
Dense2Net with Grouped Convs	75.96	98.67
Dense2Net with SE	74.77	98.68

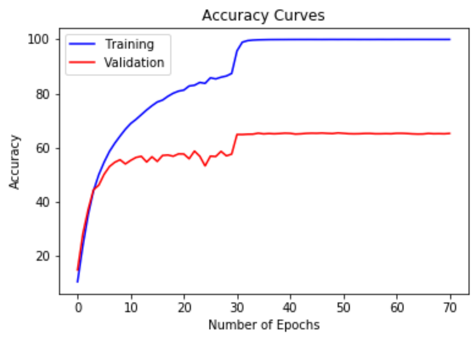

Figure 8: Accuracy curves of the baseline DenseNet model without data augmentation that we will compare our implementations against.

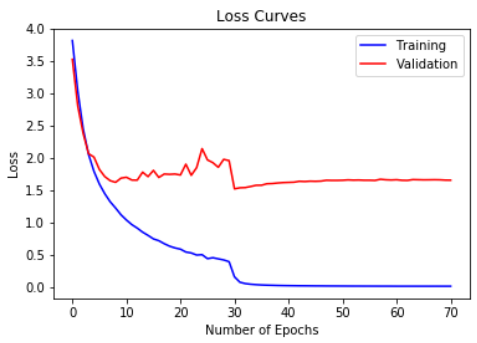

Figure 9: Loss curves of the baseline DenseNet model without data augmentation that we will compare our implementation against.

6 Conclusions and Future Work

In this paper we showed how the performance of the DenseNet architecture can be improved by implementing the new Res2Net block as described previously. We show that replacing the convolutions within the dense layers with the Res2Net block of hierarchical convolutions increases the classification performance of DenseNet on the CIFAR-100 dataset. The relative increase in performance is nearly 6% from the standard DensNet implementation. We also explore the effect of other techniques on performance of this new architecture including the cutout and the addition of an

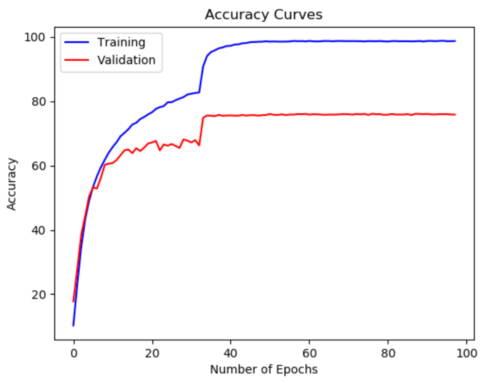

Figure 10: Accuracy curves of the best Dense2Net model with data augmentation and $scale = 8$.

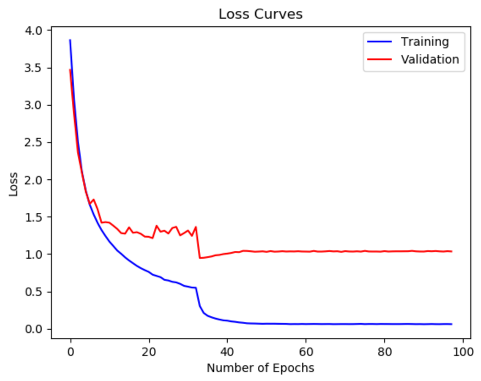

Figure 11: Loss curves of the best Dense2Net model with data augmentation and $scale = 8$.

SE layer. We also investigated the effect of changing the scale and cardinality of our Res2Net blocks and found that increasing scale lead to a small increase in performance. We found that the technique that caused the greatest increase in performance was data augmentation with a relative increase of 9.9%.

For future work, more rigorous hyperparameter tuning should be performed specifically on the learning rate, momentum, learning decay schedule, scale, cardinality, and growth rate of the dense blocks. Results can certainly be improved upon with more thorough tuning of these parameters. We also noticed that the baseline models were not performing as well as stated in the papers that they were replicated from. This discrepancy should be further investigated.

7 Appendix: Running the Code

We have provided the requirements.txt file in order to ease environment setup and installation of modules/libraries. This can easily be installed using the standard *pip install -r requirements.txt*. The main script to run the training and testing of the model is *dense2net.py*. The cutout function is provided in the *cutout.py script* however this does not need to be run and is simply referenced by the main script. The training and testing of the model can be run with all default parameters as *python3 dense2net.py*.

The command to run the training and validation of our model takes a few arguments as follows:

- --lr: Specify learning rate (default=0.1)
- --r: Resume training from checkpoint
- --a: Run without data augmentation
- --se: run without SE layer
- --c: Run without cutout
- --scale: Specify scale parameters (default=4)
- --groups: Specify cardinality (default=1)

Noticed how the boolean inputs actually remove the features, we want all of the features/techniques to be included by default. We recommend that you run the script with all default parameters.

References

[1] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. *CoRR*, abs/1707.01629, 2017. URL `http://arxiv.org/abs/1707.01629`.

[2] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. URL `http://arxiv.org/abs/1708.04552`.

[3] Shanghua Gao, Ming-Ming Cheng, Kai Zhao, Xinyu Zhang, Ming-Hsuan Yang, and Philip H. S. Torr. Res2net: A new multi-scale backbone architecture. *CoRR*, abs/1904.01169, 2019. URL `http://arxiv.org/abs/1904.01169`.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL `http://arxiv.org/abs/1502.01852`.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[6] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017. URL `http://arxiv.org/abs/1709.01507`.

[7] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL `http://arxiv.org/abs/1608.06993`.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL `http://dl.acm.org/citation.cfm?id=2999134.2999257`.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[12] Kuang Liu. Train cifar10 with pytorch, 2018. URL `https://github.com/kuangliu/pytorch-cifar`.

[13] Christos Matsoukas. Pytorch res2net, 2019. URL `https://github.com/ChrisMats/Res2Net`.

[14] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. URL `http://arxiv.org/abs/1211.5063`.

[15] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL `http://arxiv.org/abs/1505.00387`.

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL `http://arxiv.org/abs/1512.00567`.

[17] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016. URL `http://arxiv.org/abs/1602.07261`.

[18] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. URL `http://arxiv.org/abs/1611.05431`.

[19] Fisher Yu, Dequan Wang, and Trevor Darrell. Deep layer aggregation. *CoRR*, abs/1707.06484, 2017. URL `http://arxiv.org/abs/1707.06484`.