COMP551 Analysis and Modification of DenseNet

Kianoosh Ojand - 260894514, Dara Shahriari - 260715981, Negar Hassantabar - 260939318

December 2019

Abstract*—*The use of Convectional Neural Networks has gained ample traction in the realm of image classification. DenseNet is a new variant of CNN that boasts noticeable advantages over industry leaders, like ResNet. In this study we begin by giving an in depth description of DenseNet's architecture, pointing out its advantages. We then introduce our implementation of DenseNet and train it on the CIFAR10 data-set, comparing its results with those of the original[1]. Finally, in efforts to try and improve the accuracy of our implementation, we adjust certain parameters of the baseline model and discuss their effects.

1 INTRODUCTION

The use of Convolutional Neural Networks (CNNs) has become the norm for solving image related machine learning problems [2]. They learn by implementing various building blocks, such as convolution layers, pooling layers, and fully connected layers. Standard CNNs pass an input image through multiple layers of convolution, with later layers only receiving outputs from the layer directly behind them.

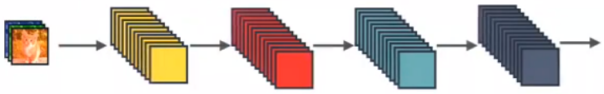

Figure 1: Standard CNN model

In recent years, improvements to the standard CNN have been made. The ResNet model expands by using element-wise addition to blend the output of layer i with that of layer $i-1$, feeding this additive blend as input for layer $i+1$. This form of identity mapping is said to promote gradient propagation, and can be viewed as an algorithm that passes state from one ResNet module to the next.

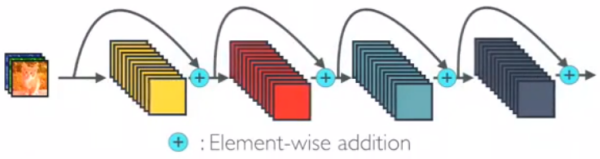

Figure 2: ResNet model

DenseNet, the model we will be analyzing in this paper, takes this feed forward variation one step further. It accumulates additional input from preceding layers via channel-wise concatenation, feeding that input to the next layer. This type of forward feeding means that each layer will receive a collective knowledge of all preceding layers.

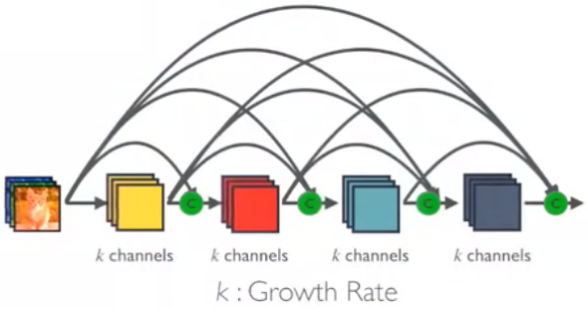

Figure 3: DenseNet model showing channel count and channel-wise concatenation

In DenseNet, every layer is provided with a feature map of all preceding layers, resulting in a thinner network with a lower channel count (width of each layer). The decrease in channel count allows the model to have high computational efficiency as well as low memory requirements. The model also allows for manual adjustment of channel count via changes in the hyper-parameter k.

2 RELATED WORKS

Convolutional Neural Networks (CNNs) are referred to as high level accuracy Machine Learning algorithms (ML) and are used for visual recognition tasks such as image classification. However, the connectivity structure of CNNs can lead to some redundancy [3][4]. In the last few decades, efforts have

been made to improve CNNs with respect to parameter usage and computing time, resulting in models like ImagNets, ResNets, DenseNets, MobileNets, and ShuffleNets[3].

The number of features used in DenseNet are dependent on the depth of the network. Limited GPU performance generally places a constraint the number of parameters and layers that can be used, but the usage of more parameters will typically lead to a better performing model[4][5][6].

DenseNet has various hyper-parameters: Normalization and augmentation, optimizers, and schedulers. Normalization and augmentation seem to be necessary for a better model because they provide richer training data. According to recent works, there is no optimal optimizer for all data-sets. So each data-set may require a specific optimizer that works best. J. Yang obtained faster and more accurate performance while using the SDG optimizer[7]. Their research also shows the importance of learning rate range for better performance[7].

3 DENSENET STRUCTURE

3.1 Composition Function

For each convolution layer, a composition function H_l of three consecutive operations is applied. These operations are batch normalization (BN), followed by a rectified linear unit (ReLU), and a 3 x 3 convolution (Conv).

3.2 Dense Connectivity

As mentioned earlier, each layer of DenseNet receives a concatenation of all preceding feature maps $x_0, ..., x_{l-1}$, as input:

$x_l = H_l([x_0, x_1, ..., x_{l-1}])$ where x_l is the output of the lth layer.

3.3 Growth Rate

The growth rate k for DenseNet represents the number of channels in the model or the width of each layer. The output of H_l, x_l, is actually a vector of feature maps with size k. Inputs to H_l receives a vector of feature maps with size $k_0 + k(l-1)$, where k_0 is the number of channels in the input layer. It is worth noting that DenseNet can operate with very narrow layers, e.g. small k.

3.4 Pooling Layers

To facilitate down-sampling, DenseNet segments the network into multiple densely connected dense blocks. Layers between dense blocks are referred to as transition layers, responsible for convolution and pooling. A 1x1 convolution followed by 2x2 average pooling represent the transition layers between two contiguous dense blocks. Feature-map dimensions are shared within a respective dense block, allowing them to be concatenated with ease.

3.5 Bottleneck Layers

Even though the output from each layer in DenseNet will only produce k feature-maps, the input usually has many more. DenseNet introduces a 1x1 convolution as bottleneck layer before each 3x3 convolution, reducing the number of input feature-maps and improving efficiency. So **BN-ReLU-1x1 Conv** is done before **BN-ReLU-3x3 Conv**.

3.6 Further Compression

For a dense block containing m feature-maps, its transition layer will produce θm output feature-maps, where $0 < \theta \leq 1$ is the compression factor. When $\theta = 0$, the number of feature-maps across transition layers will remain unchanged. We refer to DenseNet with $\theta < 1$ as **DenseNet-C**. When both compression with $\theta < 1$ and bottleneck layers are implemented, we refer to the model as **DenseNet-BC**.

4 ADVANTAGES

4.1 Strong Gradient Flow

DenseNet's improved information and gradient flow throughout the network allows the model to train easily. Each layer has direct access to loss function and original input signal gradients, leading to an implicit deep supervision [8].

4.2 Parameter and Computational Efficiency

The number of parameters in each DenseNet layer is directly proportional to $l \times k \times k$. K, which represents the width of channels, can be very small, allowing the parameter count to be much lower than that of other models.

4.3 More Diversified Features

DenseNet layers receive the feature-maps from all preceding layers as their input. As a result, DenseNet boasts a higher frequency of diversified features and tends to have richer patterns than others.

5 PROPOSED APPROACH

After verifying that our baseline performs similarly to the original proposed DenseNet[1], we will try to change parameters of our baseline, analyzing the effects. For each test, only the parameter in question will change and everything else will remain constant.

5.1 Data Set

All of the comparisons and tests performed in this analysis will be trained using the same subset of the CIFAR10 data-set, as seen in the original paper[1]. The CIFAR-10 data-set consists of 32x32 color images in 10 classes.

Even though our code will be run on Colab, using google's cloud computing services, we still have concerns regarding the models' run-times. Therefore, we will only use half of the CIFAR10 data-set for training and validation splits, minimizing validation error instead of train error to prevent over-fitting. Finally, we'll use the fully trained models to obtain a test error rate.

5.2 Baseline DenseNet and ResNets

For our first test, we will create a baseline implementation of **DenseNet-BC**. We choose to enable bottleneck layers and compression because the original paper's results[1] deem this type of model as the best performer. We define our BC baseline with the following settings: normalization and augmentation of data, SGD optimizer with learning rate = .1, MultiStepLR scheduler, depth = 100, and k = 12. This model will be referred to as **"Baseline"** from here on. We will then use this baseline to ensure that our DenseNet implementation does in fact perform better than ResNet on the same CIFAR10 data-set.

The ResNet variants we will be testing our baseline against are ResNet18, ResNet34, and ResNet50, as seen in the original paper[1]. We should mention that due to hardware limitation we run all the algorithms for only 100 epochs.

5.3 Increasing K and Depth

We touched on k and its representation of the channel width in earlier sections. In our testing, we will observe how increasing k to 24 and 40 affect the model's accuracy. We will also try changing the depth, number of layers, to 55, 190, and 250 to see if that improves anything.

5.4 Normalization and Augmentation

Next we'll want to compare that same baseline against itself but with normalization and augmentation selectively disabled. This test will give us some insight on the importance of implementing augmentation and normalization.

Normalization scales feature magnitudes so that they are consistent among themselves and sets the mean of the data close to zero. It generally speeds up learning and leads to faster convergence.

Data augmentation is a technique aimed to reduce over-fitting and to improve the quality of train data. In this experiment it randomly augments each image by shifting and mirroring. This prevents a model from learning the wrong kind of information.

5.5 Optimizers

In this section, we will examine different optimizers and observe if their usage can improve the error rate. We'll compare two optimization methods, Adadelta and Adam, to the baseline, stochastic gradient descent (SGD) with learning rates: .1, .05, and .01.

Optimizers work by updating parameters in a way that minimize the loss function. Both Adadelta and Adam are adaptive learning rate methods because they allow learning rates to behave as parameters, eliminating the need for manual tuning[9]. While both are very common in practice, Adam is generally the more popular option.

The learning rate for SGD is also an important factor to consider, as it is responsible for the behavior of the model's convergence. Small learning rates reduce the momentum of a model during gradient descent, reducing the chances of passing over a minimum. Larger learning rates add more momentum, preventing the model from getting stuck in plateaus or local minimums.

5.6 Schedulers

Another factor we'll explore is the implementation of different schedulers. Schedulers add addi-

tional complexity and control to the adaptive process of changing learning rates.

We will compare the baseline scheduler, MultiStepLR, against CyclicLR, OneCycleLR, and the baseline with no scheduler.

6 RESULTS

As seen in Table 1, our lowest test error is achieved by the baseline with (k = 24, 40), and the highest is achieved by the baseline using (Adam). According to Table 2, our fasted model is the baseline with (depth = 55) and the slowest model is baseline with (depth = 250). The baseline model with (k = 24) is a good choice if a combination of fast run-time and low error values are deemed equally important.

In the rest of this section we discuss results in the same fashion as described in our proposed approach.

Table 1: Sorted test errors of all models

Models	Test Error
Baseline (k = 24)	8.00%
Baseline (k = 40)	8.00%
Baseline (depth = 250)	8.16%
Baseline (depth = 190)	8.19%
ResNet34	8.70%
Baseline	9.00%
Baseline (No Normalization)	9.00%
ResNet18	9.83%
Baseline (depth = 55)	10.06%
ResNet50	10.25%
Baseline (SGD: lr = .05)	10.56%
Baseline (No Scheduler)	12.00%
Baseline (No Norm. and Aug.)	12.46%
Baseline (No Augmentation)	12.98%
Baseline (SGD: lr = .01)	14.91%
Baseline (CyclicLR)	15.00%
Baseline (OneCycleLR)	17.00%
Baseline (Adadelta)	19.00%
Baseline (Adam)	38.00%

6.1 Baseline DenseNet and ResNets

Our implementation of DenseNet is consistent with that of the original paper[1]. As shown in figure 4, our DenseNet baseline maintains a lower validation error than ResNet18 across the majority of epochs. Looking at table 1, we also note that our DenseNet

Table 2: Sorted average run-times across all epochs for all models

Models	Avg Train Run-time
Baseline (depth = 55)	0.084s
ResNet18	0.109s
Baseline (OneCycleLR)	0.148s
Baseline (CyclicLR)	0.151s
Baseline	0.152s
Baseline (SGD: lr = .05)	0.152s
Baseline (SGD: lr = .01)	0.154s
Baseline (No Scheduler)	0.167s
ResNet34	0.172s
Baseline (No Normalization)	0.179s
Baseline (Adadelta)	0.183s
Baseline (Adam)	0.199s
Baseline (k = 24)	0.255s
Baseline (No Norm. & Aug.)	0.285s
Baseline (No Augmentation)	0.289s
ResNet50	0.322s
Baseline (depth = 190)	0.385s
Baseline (k = 40)	0.461s
Baseline (depth = 250)	0.600s

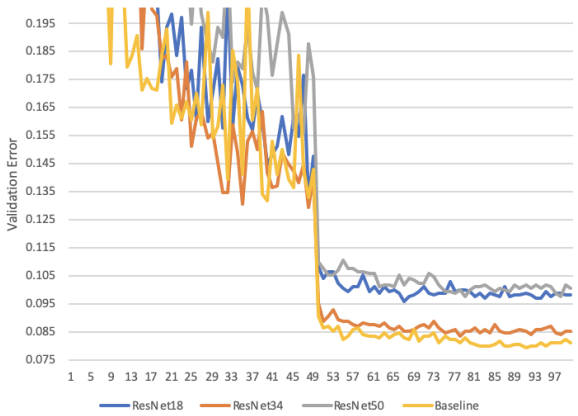

Figure 4: Validation errors of ResNet implementations and the Baseline on CIFAR10 per epoch (cropping in top left is intentional to reduce congestion at higher epochs)

baseline's test error of 9% is lower than ResNet18's test error of 9.83%. These scores are both higher than those achieved by the paper, however, this is likely due to our training data being half as large and also limitation on number of epochs.

As it can be seen in Table 1 and also in Figure 4, ResNet34 out performs both ResNet18 and ResNet50 with a test accuracy of 8.7% and validation error. We will be using this as our ResNet for comparison as we explore k and depth.

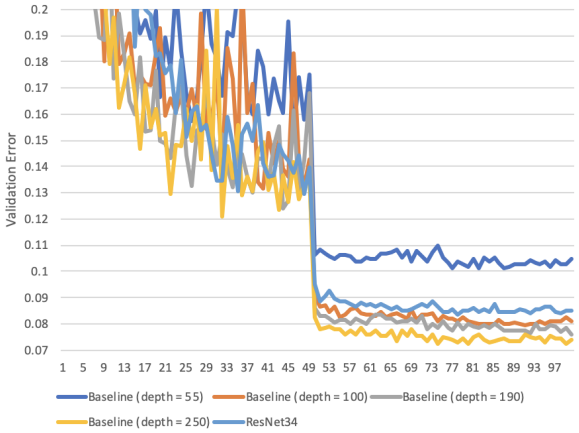

Figure 5: Validation errors per epoch of ResNet34 and Baselines with varying depth on CIFAR10 (cropping in top left is intentional to reduce congestion at higher epochs)

6.2 Increasing K and Depth

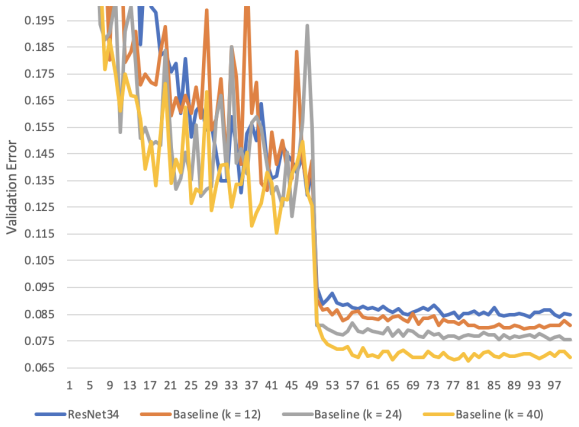

Figure 6: Validation errors per epoch of ResNet34 and Baselines with varying k on CIFAR10 (cropping in top left is intentional to reduce congestion at higher epochs)

Our best scoring models, regarding test error, are Baselines k = 24, k = 40, and depth = 250 with test errors of 8%, 8%, and 8.16% respectively (table 1, figure 5, figure 6). In accordance with the original paper[1], our results demonstrate a decrease in test error with the increase of k and of depth. High k and depth values increase average run-time when compared to other modifications (table 2). As a result, our hardware limitations prevented us from combining these key takeaways into a mixed model ex. baseline with k = 40 and depth = 250. Intuition seems to indicate that these two parameters would behave constructively, improving accuracy beyond 8%.

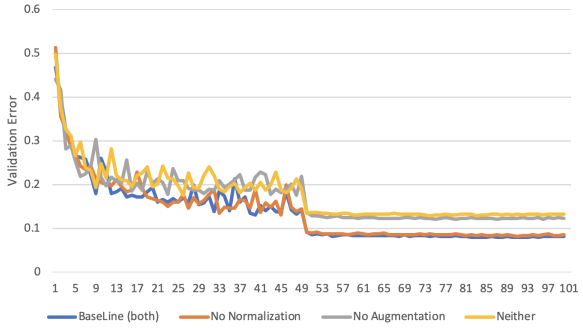

Figure 7: Validation errors per epoch of the Baseline, with both augmentation and normalization enabled, and Baseline variants with them disabled on CIFAR10

6.3 Normalization and Augmentation

DenseNet seems to work best when both augmentation and normalization are enabled (table 1). The inclusion of normalization appears to make very little difference in the validation error of the baseline model (figure 7). This lack of change actually makes sense if we consider the fact that the data we're learning consists of positive pixel values. If our data had some negative values, centering the mean at zero would probably have a more noticeable effect. Augmentation, however, has a more drastic impact because it effectively adds variation to our training data, improving its quality. For example, consider the situation where all pictures of mice are facing right to left. The model might learn how to classify mice only facing this direction. Mirroring of the images would counter this bias, giving DenseNet a better chance of genuinely learning the geometry of a mouse.

6.4 Optimizers

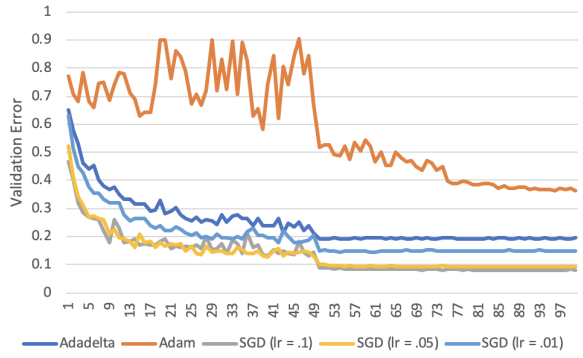

Figure 8: Validation errors per epoch of Adadelta and Adam compared against SGD with different learning rates (lr = .1 is the baseline).

It is clear that the SGD variants outperform both Adadelta and Adam by a significant margin (figure

8). The implementation of DenseNet with SGD and learning rate of .1 achieves the best results with a test error of 9%. SGD, using learning rates .05 and .01 respectively, earns test errors of 10.56% and 14.91%. Reducing the learning rate past .1 seems to have a negative outcome on accuracy. It is interesting to note that learning rates .1 and .05 perform relatively evenly with respect to validation error (figure 8), a trend that is not reciprocated in the test results in table 1. Perhaps learning rate of .05, while able to converge to a global minimum in the validation set, doesn't have enough momentum to pass over a local minimum in the test set.

6.5 Schedulers

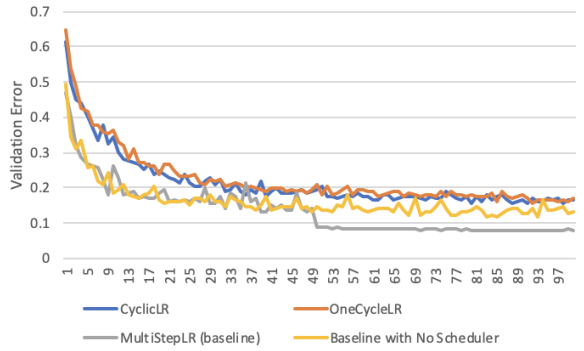

Figure 9: Validation errors per epoch of CyclicLR and OneCycleLR compared against the baseline scheduler (MultiStepLR) and the baseline with no scheduler.

Unfortunately, the inclusion of OneCyclic and CyclicLR does not improve accuracy either. The baseline actually performs better when no schedulers are used at all, (figure 9, table 1). MultiStepLR, the original scheduler, proves to be the best scoring with accuracy of 9% (table 1).

7 CONCLUSION

In this work, we investigated the convolutional neural network architecture: DenseNet. We demonstrated that our implementation of the model did indeed replicate the original implementation's superiority over ResNet[1]. Various parameters were tuned, including augmentation and normalization, growth rate and depth, type of scheduler, and choice of optimizer. Depth and growth rate proved to be the most useful with respect to test accuracy improvement, indicating that an ideal version of DenseNet would likely need high values for both to achieve optimal accuracy.

DenseNet boasts parameter efficiency, feature diversification, and strong gradient flow as advantages. DenseNet is noticeable faster than ResNet, all while achieving similar or better accuracy. This model could very well become a commonly used model in the field of image classification. We look forward to exploring more of DenseNet has to offer and furthering its performance in the future.

8 STATEMENT OF CONTRIBUTIONS

Kianoosh worked on the coding part and running them and helped a little bit on the report, Dara worked on writing report and Negar worked on literature review and helped a little bit on running the codes.

References

[1] Laurens van der Maaten Gao Huang, Zhuang Liu and Kilian Q. Weinberger. Densely connected convolutional networks. 2018.

[2] J. S. Denker D. Henderson R. E. Howard W. Hubbard Y. LeCun, B. Boser and L. D. Jackel. An efficient three-stage classifier for handwritten digit recognition. In *Neural computation*, page 541–551. IEEE, 1989.

[3] Gao huang, shichen liu, laurens van der maaten, kilian q. weinberger;"condensenet: An efficient densenet using learned group convolutions" the ieee conference on computer vision and pattern recognition (cvpr), 2018, pp. 2752-2761.

[4] Geoff pleiss, danlu chen, gao huang, tongcheng li, laurens van der maaten, kilian q. weinberger,"memory-efficient implementation of densenets", 21 jul 2017.

[5] Yulin wang, xuran pan, shiji song, hong zhang, gao huang, cheng wu, "implicit semantic data augmentation for deep networks", advances in neural information processing systems 32 (nips 2019) pre-proceedings.

[6] Y. zhu and s. newsam, "densenet for dense flow," 2017 ieee international conference on image processing (icip), beijing, 2017, pp. 790-794. , doi: 10.1109/icip.2017.8296389.

[7] Yang, j.; yang, g. modified convolutional neural network based on dropout and the stochastic gradient descent optimizer. algorithms 2018, 11, 28.

[8] P. Gallagher Z. Zhang C.-Y. Lee, S. Xie and Z. Tu. Deeplysupervised nets. In *AISTATS*, pages 2, 3, 5, 7, 2015.

[9] Renu Khandelwal. Overview of different optimizers for neural networks. In *Medium*. Data Driven Investor, Feb 3, 2019.