# Study of Models for Image Classification

Fan Gao[*], Seung Hwan Lee[†], and Sneha Nagaraj Bangalore[‡]

Columbia University, New York, NY

## Abstract

*In this project, we examine various deep learning models used for Image Classification task. We base our models on DenseNet, RNN, CNN, ResNet and Inception architectures and explore potential new deep learning models that draw benefits from combinations of them. The goal of this project is to run experiments on simple yet challenging data sets, collect empirical results and analyze them to understand how each of our proposed models work based on their underlying architectures. Thus, our contribution to the image classification community would be the novel models we have come up with, their experimental results and our analysis on them.*

## 1. Introduction

In the last few years, our image recognition capabilities have dramatically improved thanks to advances in deep learning. Object-detection, segmentation, human pose estimation, video classification, object tracking, and super-resolution are only a few examples of all the computer vision applications to which deep neural networks have been successfully applied.

One encouraging fact is that most of this progress is not just the result of more powerful hardware, larger datasets and more complex models, but also a consequence of novel ideas, more efficient algorithms and improved network architectures. These novelties have given state of the art results in image classification task evaluated on multiple benchmark data sets such as CIFAR-10, CIFAR-100, SVHN and ImageNet.

Some of the new models are variants of Convolutional Neural Networks (CNNs) that rely on its ability to extract local features while others are based on Recurrent Neural Networks (RNNs) which leverages its ability to capture global features. Others try to maximize information flow while keeping a manageable amount of parameters. While each of these individual models has been studied, there is still room to explore how hybrid models that draw benefits from each of these models would perform. It is not always clear if such enhancements will result in better performance.

Thus, the goal of our project is to collect empirical results on how various models perform and analyze the results to understand how the components taken from different deep learning models interact with one another. We have chosen to create three hybrid models. Wide DenseNet is an attempt to apply the computational advantages of wide network to DenseNet. ReLeNet tries to combine global and local feature extraction from ReNet and CNNs. ResNet-Inception tries to draw the best of ResNet and Inception Network. We hope that we can gain insights from studying these novel architectures and applying them to the task of image classification.

## 2. Problem Formulation

Through the study of deep learning methods, we have learned that some architectures excel in modeling dependency through time, while others specializing in local feature extraction. Some models have better gradient flow, while others are more computationally efficient. In practice, there are multiple aspects one needs to consider when creating a model for image classification task. Understanding how different components of a model work and interaction is vital. This deeper understanding allows us to improve prevailing models and customize them to be better suited for a specific task. By exploring the benefits of existing architectures and understanding how their combinations perform, we hope to learn and contribute simultaneously.

## 3. Wide DenseNet (WDN)

### 3.1. Related Work

Recent work has shown that densely connected convolutional networks (DenseNet) obtain significant improvements over deep residual networks, whilst requiring less computation to achieve higher performance. Instead of a

[*]fg2432@columbia.edu
[†]sl3966@columbia.edu
[‡]sb3889@columbia.edu

chain structure adopted by traditional convolutional networks, [2] proposed a structure in which all layers have direct connections through concatenation of feature-maps of all preceding layers and current layer and feed it as input into all subsequent layers. DenseNet:

- Alleviate the vanishing-gradient problem
- Strengthen feature propagation
- Encourage feature reuse / avoid information loss
- Substantially reduce the number of layers and parameters

On the other hand, [7] showed that very deep networks have a problem of diminishing feature reuse, which makes these networks hard to train and proposed a shallow and wide architecture.

## 3.2. Methods

While DenseNet is a simple and elegant architecture, we argue that the architecture is not fully optimized and there's room to further improve it. Our attempt is to widen the network as proposed by [7]. While the number of parameters increases linearly with depth, it increases quadratically with width. However, it is more computationally effective to widen the model since GPU is much more efficient in parallel computations on large tensors. Table 1 and Figure 1 show the structure and architecture of WDN.

Table 1. Structure of Wide Dense Networks

| Layer | Output Size | Block Type |
|---|---|---|
| initial conv | 32 x 32 | [3 x 3, k*2] |
| block1 | 32 x 32 | (bottleneck [1 x 1, k * 4], composite [3 x 3, k ]) x N/2 |
| transition1 | 16 x 16 | composite [1 x 1, k * c] 2 x 2 average pool, stride 2 |
| block2 | 16 x 16 | (bottleneck [1 x 1, k * 4], composite [3 x 3, k ]) x N/2 |
| transition2 | 8 x 8 | composite [1 x 1, k * c] 2 x 2 average pool, stride 2 |
| block3 | 8 x 8 | (bottleneck [1 x 1, k * 4], composite [3 x 3, k ]) x N/2 |
| transition3 | 1 x 1 | composite [1 x 1, k * c] |

Number of layer within each block N = (depth - 4) / number of blocks. Growth rate k = number of feature maps. Compression ratio c = 0.5. Note that each conv layer corresponds to the sequence BN-ReLU-Conv2d.

There are multiple advantages to this architecture:

- More computationally effective.
- Considerable reduction in training time and inference time. Reduction in training time helps speeding up development cycles. And reduction in inference time benefits end-to-end models. For example, a shorter latency gives a self-driving car more time to stop for a pedestrian crossing the street, it improves safety of the pedestrian as well as comfortability for the driver.
- Given the same amount of time, wide networks can learn with more parameters than thin ones, which would require considerably more layers, making them unfeasibly expensive to train.
- Wide networks have better scalability given more computing power, which is critical in training large scale networks.

## 3.3. Optimal Depth/Width ratio

However, a network cannot be completely flattened. In fact, performance starts degrading when network depth falls below a certain level. It'd better to find the optimal depth/width ratio so that we can:

- Take full advantage of parallel computing.
- Avoid loading too many parameters into the model.
- Preserve feature extraction and feature reuse.

Based on prior research, expanding either the depth or width of a model will likely increase its representational power. With limited time budgeted for training or inference, however, theres a trade-off between model size and training time. We found that 1:1 depth/width ratio is best for CIFAR-10 dataset.

Note that optimal ratio is unique for every problem, as it's determined by many factor, i.e., the dimension of underlying images, the size of mini batches, the architecture of a model, and the computing power of GPUs, etc. We encourage readers to explore a few configurations to find the optimal ratio for their problems.

## 3.4. Results

We ran a comprehensive set of evaluations of our proposed architecture on CIFAR-10. Standard data augmentation was applied to the images (color normalization, horizontal flip and random crop). CUDA 8.0, cudnn V6 and NVidia Tesla P100 GPUs were used for training and measurement. Results are shown in Table 2.

In Row 1, we reproduced a long and thin model from the original paper as benchmark. Comparing row 1 and row 2, we see a WDN with similar number of parameters as the baseline model achieves lower accuracy, but significant reduction in both training time and inference time. In row 3, we see a WDN, when given roughly the same amount of training time as the baseline model, scores higher accuracy.

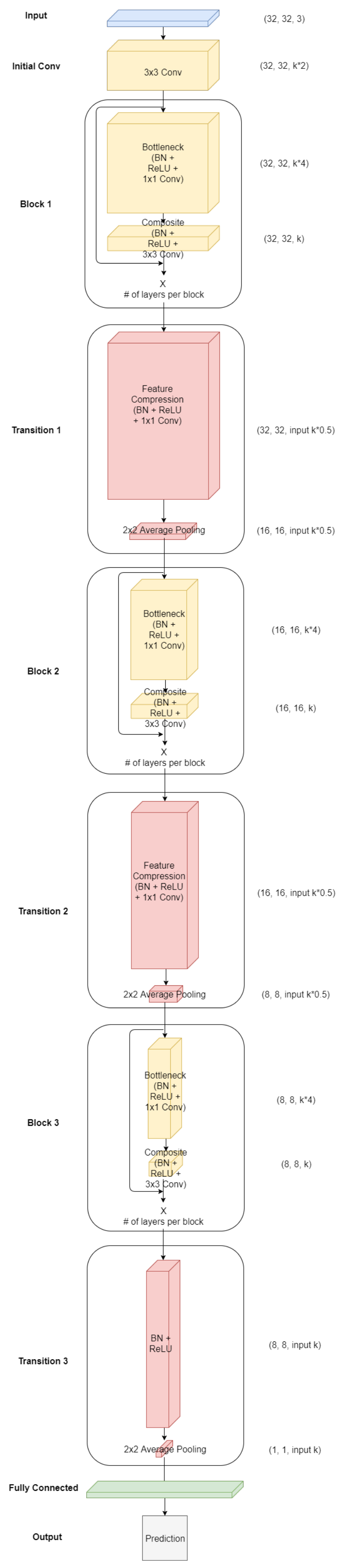

Figure 1. Wide DenseNet Architecture

Comparing row 3 and 5, we see that further flattening the model is not beneficial, resulting in longer training time and lower accuracy. In addition, note that wider models benefit more from an extra GPU in terms of both training and inference time.

Results for ImageNet are shown in Table 3. Due to time and resource limits, we ran one image through the model to measure training and inference time. Time reduction is not as significant as before, but still meaningful. As stated earlier, the optimal depth/width ratio is different for every problem.

## 4. ReNet with CNN

### 4.1. Related Work

Unlike many other image classification deep learning models that rely on Convolutional Neural Networks (CNNs), ReNet relies on Recurrent Neural Networks (RNNs). Thus, unlike CNN based models that rely on extracting local features, ReNet tries to extract features that rely on the whole image information due to its dependence on Recurrent Neural Networks [6]. The input image of the feature map gets divided into small patches. The architecture makes vertical sweeps on each of these patches by feeding them into RNNs, where each sweep also requires a forward and a reverse scan. The hidden units for vertical forward RNN and vertical reverse RNN get concatenated. Now, the architecture makes horizontal sweeps on each of these hidden units, where each sweep again requires a forward and a reverse scan. Again, the hidden units for forward and reverse RNNs get concatenated. The concatenated hidden units are used as features that get fed into a softmax classifier to make predictions.

### 4.2. Methods

The authors of ReNet paper hypothesize that the features that ReNet and LeNet learn are quite different, and suggest a future work as an ensemble method of RNN and CNN. The proposed model takes a similar path, by adding convolutional neural networks to perform another feature extraction. Rather than only feeding in hidden units of RNN into a classifier, the proposed model also create features using CNN, concatenates features from RNN and CNN and feed them into a softmax classifier. This in a way provides both local and global features and allows the model to construct useful relationships out of those features.

General architecture of ReNet is described and defined in the original paper. Some of them are number or ReNet layers( $N_{re}$), their corresponding receptive field sizes ($w_p$ x $h_p$) and feature dimensionality ($d_{RE}$).

For LeNet, they are channel size ($C_{in}$), out channel size ($C_{out}$), conv filter size ($F$), strides ($S$), kernel size $K$.

Table 2. CIFAR-10 Results for Wide DenseNet

| | | | | | 1-GPU | | 2-GPU | |
|---|---|---|---|---|---|---|---|---|
| Model | Depth | Growth Rate(k) | # Parameters | Error(%) | Training Time (sec/epoch) | Inference Time (sec/epoch) | Training Time (sec/epoch) | Inference Time (sec/epoch) |
| DenseNet-BC | 100 | 12 | 0.8M | 4.54 | 75.07 | 4.08 | 50.83 | 3.30 |
| DenseNet-BC | 19 | 48 | 0.7M | 5.99 | 21.11 | 1.41 | 14.86 | 1.10 |
| DenseNet-BC | 40 | 48 | 2.7M | 4.14 | 78.20 | 4.71 | 46.92 | 2.98 |
| DenseNet-BC | 19 | 96 | 2.7M | 5.03 | 50.83 | 3.30 | 31.28 | 1.96 |
| DenseNet-BC | 22 | 96 | 4.4M | 4.38 | 81.83 | 5.10 | 53.96 | 2.98 |
| DenseNet-BC | 40 | 96 | 10.9M | 3.91 | 197.06 | 12.25 | 114.17 | 6.75 |

Table 3. ImageNet Results for Wide DenseNet

| | | | | | 1-GPU | |
|---|---|---|---|---|---|---|
| Model | Depth | Growth Rate(k) | # Parameters | Layers/Dense Block | Training Time (sec/epoch) | Inference Time (sec/epoch) |
| DenseNet-BC | 34 | 64 | 10.2M | 3+6+12+8 | 0.030 | 0.021 |
| DenseNet-BC | 45 | 64 | 16.4M | 4+8+16+12 | 0.039 | 0.024 |
| DenseNet-BC | 63 | 64 | 31.2M | 6+12+24+16 | 0.055 | 0.055 |
| DenseNet-BC | 121 | 32 | 8.0M | 12+24+48+32 | 0.047 | 0.026 |
| DenseNet-BC | 121 | 48 | 17.1M | 12+24+48+32 | 0.057 | 0.028 |
| DenseNet-BC | 121 | 64 | 26.2M | 12+24+48+32 | 0.056 | 0.051 |
| DenseNet-BC | 121 | 120 | 100.7M | 12+24+48+32 | 0.072 | 0.185 |
| DenseNet-BC | 201 | 12 | 3.3M | 12+24+96+64 | 0.076 | 0.037 |
| DenseNet-BC | 201 | 20 | 8.3M | 12+24+96+64 | 0.082 | 0.034 |

Table 4. Structure of ReNet of ReLeNet

| | |
|---|---|
| $N_{re}$ | 1 |
| $w_p$ x $h_p$ | 4 x 4 |
| $d_{RE}$ | 320 |

Table 5. Structure of LeNet of ReLeNet

| | |
|---|---|
| conv1 | $C_{in}$ : 3, $C_{out}$: 128, $F$: 3, $S$: 1 |
| pool1 | $K$: 2, $S$: 2 |
| batch norm1 | yes |
| conv2 | $C_{in}$ : 128, $C_{out}$: 48, $F$: 4, $S$: 1 |
| batch norm2 | yes |

## 4.3. Results

During all training, one Nvidia P100 GPU was used and CIFAR-10 data set was used.

### 4.3.1 ReNet base Model

ReNet base model achieved a test accuracy of around 60 %. This is around 27.5 % lower than what the paper was able to achieve. The differences can be attributed to using fewer number of layers and bigger receptive filter size, which as a result creates fewer image patches. Also, lack of applying data augmentation, flipping and rotating potentially contributed to lower test accuracy. Also note that the goal of this study wasn't to achieve the highest accuracy, but rather collect empirical evidence and provide analysis, thus not much effort was put into optimizing the base model.

### 4.3.2 ReNet with Convolution

Adding a convolutional layer on the image input increased the test accuracy to around 66%. The effect was basically increasing the image channel from 3 to 128. It is widely known that adding a convolutional layer on top of RNN input usually enhances the overall model thus it is very likely that the authors of the paper already knew about the effect of adding a convolutional layer. The paper likely just focused on RNNs only as the goal was to send a message that RNNs can also potentially be used to directly extract features from images rather than just used as a tool to sequentially process images for visual attention methods. ReNet with Convolution was performed just to gather an empirical evidence that CNN can help to augment the base ReNet model.

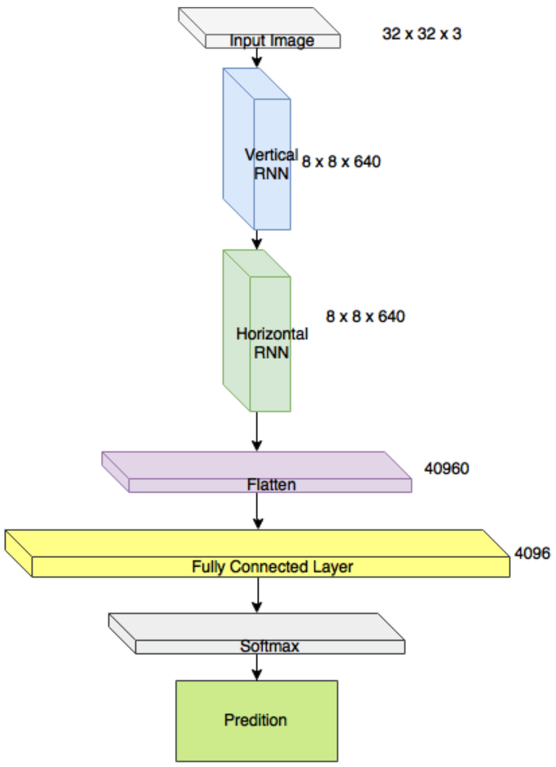

Figure 2. ReNet Base Model Architecture

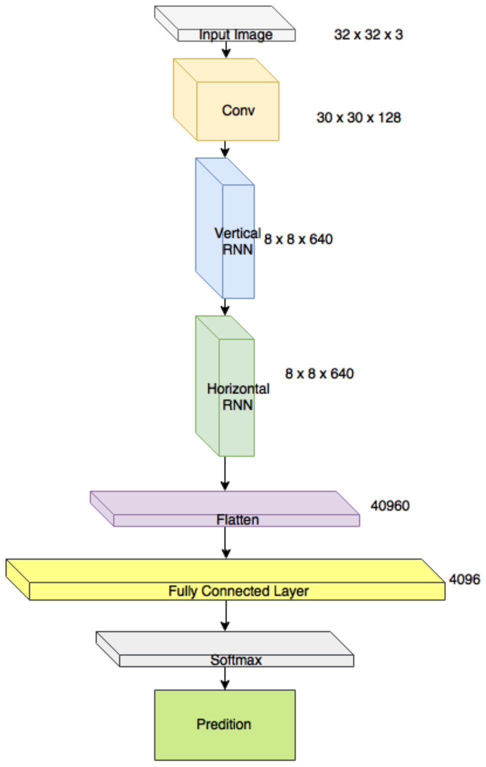

Figure 3. ReNet with Convolution Architecture

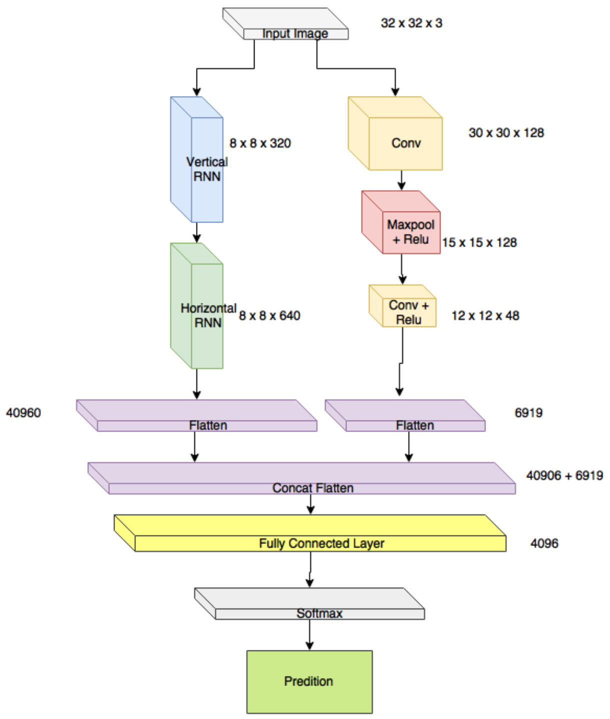

Figure 4. ReLeNet Architecture

#### 4.3.3 ReLeNet

For ReLeNet, the test accuracy reached around 77%. However, for LeNet(CNN) only, the test accuracy also reached 77%. Though ReNet and LeNet have learned quite different features, it appears like as CNN training time is much faster than RNN, during the training phrase, most of the gradients actually end up flowing back into CNN, as optimizing the CNN part alone can drastically reduce the overall loss. Thus, it appears that to use both CNN and RNN feature to enhance the image classification, one needs to come up with a architecture where gradients can be equally distributed among RNN and CNN during the training phase.

One conceptual architecture that could potentially work is to come up with an objective function that will initially place more emphasis on the RNN features, which will give a motivation for the architecture to learn features extracted from RNN equally well.

## 5. ResNet-Inception

### 5.1. Related Work

The use of residual connections improves the training speed greatly since, reformulating the layers as learning residual functions ($F(x) = H(x) - x$) with reference to the layer inputs ($x$), instead of learning unreferenced functions ($H(x)$) is easier to optimize for the solver (like SGD). The introduction of residual connections in conjunction with a more traditional architecture has yielded state-of-the-art performance.

Inception architecture has been shown to achieve very good performance under strict constraints on memory and computational budget. It increases the representational power of neural networks.

This raises the question of whether there is any benefit in combining the Inception architecture with residual connections.

This question is not new and has already been studied in [3]. They compare the two pure Inception variants, Inception-v3 and v4, with similarly expensive hybrid Inception-ResNet versions which replace filter concatenation with residual connections and show two important findings. One, that Inception with residual trains faster than just Inception. Another, that Inception with residual performs better than just Inception.

Motivated by these results, we came up with a simpler ResNet-Inception network. We measure the accuracy and inference time of our simple new architecture. We want Inception to reap all the benefits of the residual approach while retaining its computational efficiency.

## 5.2. Methods

The focus here is on the behavior of the network, but not on pushing the state-of-the-art results, so we intentionally used simple architectures and tried several versions of the Residual version of Inception. Only one of them is detailed here.

The first layer is $3 \times 3$ convolutions. [3] recommends that the first pooling layer must be removed in order to make the classification work for lower resolution images. Hence we do not have a pooling layer after the first convolution layer. Since the computational budget should be distributed in a balanced way between the depth and width of the network, we chose 4 Groups ($n$) with different number of $3 \times 3$ convolution blocks within them, based on [2].

In order to perform the shortcut connection akin to residual networks, the input and the Group output must be of the same size. For this, we add Conv Project (blue block in Figure 6 which is a $1 \times 1$ convolution with no activation.

The Conv Project (blue block) is the $x$ from residual network notation and the Group (white block) along with the Conv Reduce (yellow block) learns the residual $F(x) = H(x) - x$ making the next Group receive $F(x) + x$ as its input through the Conv Project (blue block).

The Groups consist of $3 \times 3$ convolutions with the same residual concept applied within. The idea is that, since Inception increases the representational power and residual connections make training faster, we expect this architecture to learn faster and better. Table 6 describes the number of filters applied in each of the models we tried.

[4] reports that a move from fully connected layers to average pooling improves the accuracy. Hence the network ends with a global average pooling, and softmax.

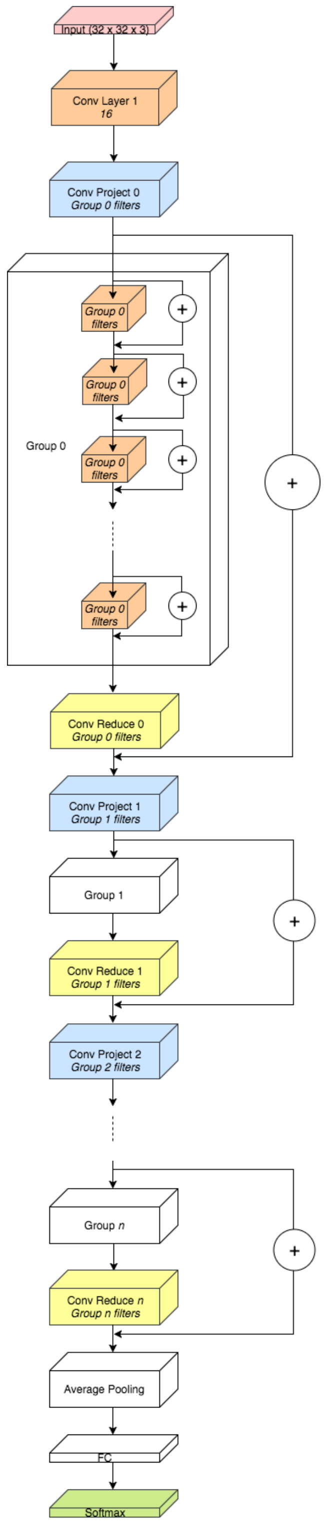

Figure 5. ResNet-Inception architecture

Figure 5 shows the overall architecture of the ResNet-Inception network where the residual connections between the Groups represent the outer layer of residual connections and the Group 0 block shows the inception of residual connections. Figure 6 zooms into the description of convolution blocks used in Figure 5.

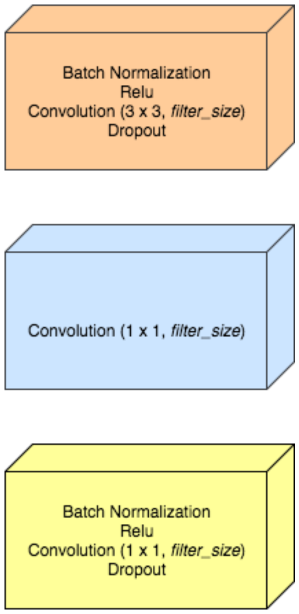
Batch Normalization
Relu
Convolution (3 x 3, *filter_size*)
Dropout
Convolution (1 x 1, *filter_size*)
Batch Normalization
Relu
Convolution (1 x 1, *filter_size*)
Dropout

Figure 6. Zoom in view of the convolution blocks from Figure 5

Table 6. ResNet-Inception Models

| Attribute | Model 1 | Model 2 | Model 3 | Model 4 |
|---|---|---|---|---|
| Group 0 block count | 6 | 6 | 6 | 24 |
| Group 0 filter size | 32 | 64 | 32 | 32 |
| Group 1 block count | 12 | 12 | 12 | 16 |
| Group 1 filter size | 32 | 64 | 64 | 32 |
| Group 2 block count | 24 | 24 | 24 | 12 |
| Group 2 filter size | 32 | 64 | 64 | 32 |
| Group 3 block count | 16 | 16 | 16 | 6 |
| Group 3 filter size | 32 | 64 | 128 | 32 |

### 5.3. Results

We trained our networks with stochastic gradient descent utilizing the TensorFlow framework. We used an instance running an NVIDIA Tesla K80 GPU, a batch size of 50 or 100 and a training period of 100 or more epochs. Our experiments used a learning rate of 0.1, decayed every 10000 steps using an exponential rate of 0.8. We follow the BN-Relu-Conv-Dropout template for every convolution except for the projection convolution.

We evaluate our method on the CIFAR10 classification dataset that consists of 10 classes. The models are trained on the 50k training images, and evaluated on the 10k validation images. The network inputs are $32 \times 32$ images (and 3 RGB channels), with the per-image mean subtracted. Both for training and testing, we only evaluate the single view of the original $32 \times 32$ image.

The results for Models 1, 2 and 3 are shown in Table 7. [5] suggests that the representation size should gently decrease from the inputs to the outputs before reaching the final representation used for the task at hand. However, when we implemented this suggestion in ResNet-Inception Model 4 with Group block counts going from 24 to 6, we observed that the performance deteriorated.

Figure 7 shows the average pool (last but one layer) weights for the best performing (both in terms of accuracy and speed) ResNet-Inception model. (Model 1)

Table 7. ResNet-Inception Results

| Attribute | Model 1 | Model 2 | Model 3 |
|---|---|---|---|
| Train Accuracy | 95% | 96% | 100% |
| Test Accuracy | 82% | 80% | 84% |
| Number of epochs | 300 | 100 | 270 |
| Batch size | 100 | 100 | 50 |
| Inference time (sec/epoch) | 27.78 | 44.47 | 72.1 |
| Number of parameters ($10^6$) | 1.1 | 4.37 | 7.6 |

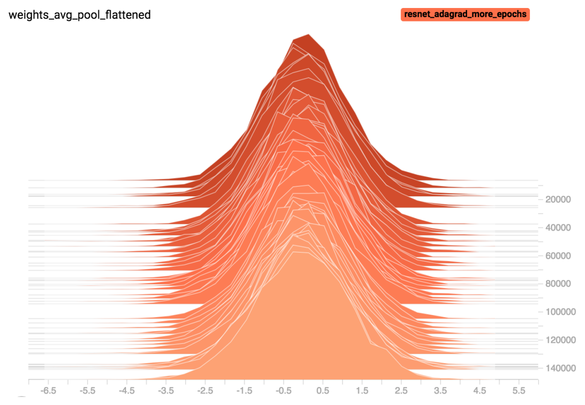

Figure 7. ResNet-Inception Average Pool layer weights

### 5.4. Conclusion

In this exercise, we found that balancing the depth and width of DenseNet is beneficial. An optimal ratio will improve the computational efficiency as well as accuracy of DenseNet. This idea may apply to other architectures too. For future work, implementing an optimization algorithm to search for the best depth/width ratio may allow us to build models that fully exploit the computing power of our setup.

Using ReNet model, the interaction between CNN and RNN models was studied. One takeaway is that to come up with an efficient architecture that can effectively take advantages of the features learned from both RNN and CNN, one may need to focus on coming up with an objective function that can help to equally distribute gradients for learning both RNN based features and CNN based features during training.

For ResNet-Inception, [5] suggests that scaling down the residuals (say, between 0.1 and 0.3) before adding them to the previous layer activation stabilizes the training. This can be done to check if this is indeed going to help. One other experiment that proved to be fatal was removing the Conv Reduce altogether. Further investigation is needed to find out why a projection onto the same number of dimensions with non-linear activation helps. Another observation made was that Model 1 reached the peak accuracy value after longer training whereas Model 2 (with more number of parameters) reached the same (or slightly better) number earlier.

## References

[1] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *arXiv eprint arXiv:1512.03385* (2015).

[2] Gao Huang et al. “Densely connected convolutional networks”. In: *arXiv preprint arXiv:1608.06993* (2016).

[3] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In: *arXiv eprint arXiv:1602.07261* (2016).

[4] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *arXiv eprint arXiv:1409.4842* (2014).

[5] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *arXiv eprint arXiv:1512.00567* (2015).

[6] Francesco Visin et al. “Renet: A recurrent neural network based alternative to convolutional networks”. In: *arXiv preprint arXiv:1505.00393* (2015).

[7] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146* (2016).