A Comparison of DenseNet and ResNet

groveh

February 2019

1 Introduction

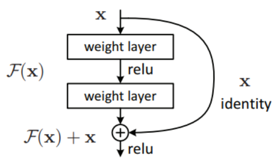
Both ResNet and DenseNet adopt a deep technique to neural networks. A deep network is one that stacks multiple feature extraction layers that feed into one another in order to gain a highly abstract, non-linear representation of an image. In 2015, ResNet presented a residual learning framework aimed to ease training of highly layered networks that arises from difficulty in gradient calculation because of non-convex error surfaces (due to numerous non-linear activations), and a vanishing gradient that becomes exponentially smaller during each subsequent layer, thus forcing the learning rate towards zero and therefore considerably slowing the learning process. Generally, the draw towards deeper networks stems from their tendency to generalize, while shallower networks with more parameters (wider networks) have a habit of memorizing a training set. A significant number of stacked layers can generalize because they have the capacity to learn intermediate features between raw data and high level classification. With an increase in parameters per layer, a network will analyze an image at a level too detailed and specific to maintain consistent classification on an unknown dataset.

2 ResNet

With a large number of layers, computational burden becomes increasingly difficult. A non-convex error coupled with vanishing gradient strains the network's ability to train at a practical rate. ResNet introduces residual learning every few stacked layers with an identity function. Specifically, residual networks address a degradation problem where networks may have difficulty approximating identity mappings with multiple non-linear layers. While identity mappings are rarely optimal, it provides reasonable preconditioning to improve common cases where the target function is more similar to an identity mapping than to a zero mapping. As a baseline network, ResNet uses convolutional layers that have 3×3 filters with two cases that dictate the number of filters: (i) the same number of filters for layers with the same feature map size as the previous layer; or (ii) double the number of filters for layers with half the feature map size as the previous layer. Down-sampling is done using a stride of 2, and the baseline network ends with a global average pooling layer and a 1000-way fully connected layer with a softmax activation. By inserting shortcut connections, ResNet creates a residual version of the baseline network, defined as:

$$y = F(x, \{W_i\}) + x \tag{1}$$

Where x and y are the input and output vectors, and $F(x, \{W_i\})$ represents the residual mapping to be learned. Using the following network as an example:

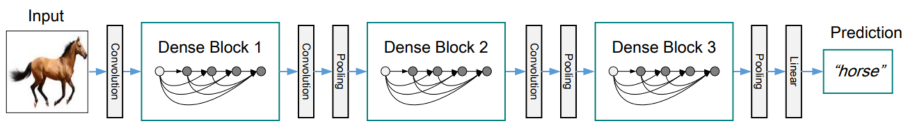

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

We can see that $F = W_2\sigma(W_1 x)$ where σ denotes ReLU activation, and element wise addition computes $F(x)+x$ from the shortcut connection. For connections where the dimensions change, two options are considered: an identity shortcut with padded zero entries, or a projection shortcut utilizing an additional parameter W_s to match dimensions:

$$y = F(x, \{W_i\}) + W_s x \tag{2}$$

3 DenseNet

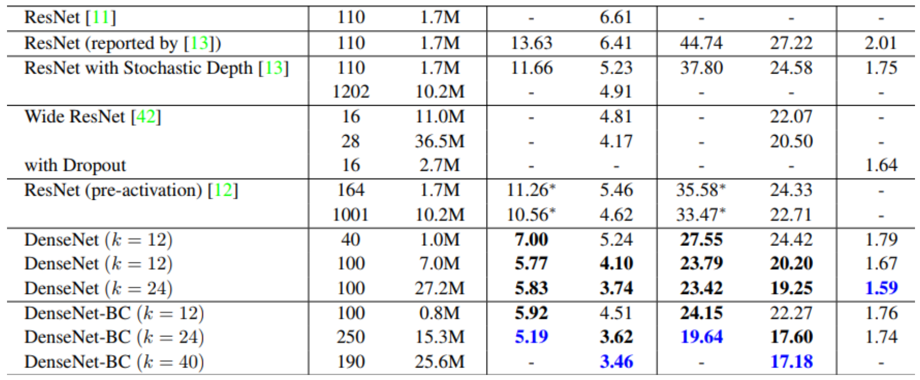
DenseNet introduces a connectivity pattern to increase information flow within a network by connecting layers to all subsequent layers within a "Dense Block", leading to an implicit deep supervision. These dense block are defined by matching feature map sizes, meaning that if the dimension changes the current dense block ends and a new one begins. Contrary to ResNet where features are combined through summation before being passed into another layer, features are combined by concatenation and therefore keep the feature maps generated by previous layers within the dense block unchanged. Below is an example of a 3 block DenseNet:



Each layer within a dense block uses a composite function consisting of three consecutive operations: batch normalization, a ReLU activation, and a $1x1$ convolution. The growth rate of the network is defined as the number of feature maps generated in each layer of a dense block. Layers are expected to be rather narrow due to a small growth rate, 12 filters for example. In order to use the concatenation operation, the feature maps generated in each dense block layer must be down-sampled. Layers in between each dense block are called pooling layers, and they are responsible for changing the feature map sizes through convolution and pooling. Each pooling layer consists of batch normalization, a $1x1$ convolutional layer, and a $2x2$ average pooling layer. In an implementation labeled DenseNet-B, a Bottleneck layer is introduced that adds a $1x1$ convolution before each $3x3$ convolution in the composite function. In a further implementation labeled DenseNet-C, a compression factor is added to reduce the number of feature maps by a factor $0 < \theta \leq 1$

4 Comparison

DenseNet utilizes feature reuse, which means the model is easy to train and highly parameter efficient. ResNet has a significantly larger amount of parameters because each layer has its own weights. DenseNet layers can be much narrower because they add only a small set of feature maps to the “collective knowledge” of the network. DenseNet is known to be rather memory hungry and takes a longer time to train, but has shown higher accuracy rates on standard datasets.



ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	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	-	-	-	-	1.64
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	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

DenseNet’s implicit deep supervision allows individual layers to receive additional supervision from the loss function through shorter connections. On top of higher accuracy, DenseNet offers a less complex model that is easy to train because of its feature reuse. It will be a good framework to use for initial testing, and should new research show that ResNet or another network will turn out to be better suited, switching the network is a brief process.

5 Implementation

With regard to the orientation problem, DenseNet can either be modeled after classification or regression.

Regression

Estimation is considered at three levels: $\pm30^\circ$ in the narrowest setting, $\pm45^\circ$ at the next level, and at the worst case of estimation a full $\pm180^\circ$ rotation of the image is explored. This ensemble of inferences is potentially easier to estimate because the course estimation of a full rotation is not expected to result in the optimal output, but rather be at least within the range of the the next level down so that the next estimation can be more accurate. In practice, this technique was shown to end with an accuracy of around 3° after the final estimation from the $\pm30^\circ$ layer on both easy and hard images, which are defined in proportion to their angle deviation, so images that are more rotated are labeled as hard.

Task	Net-30	Net-45	Net-360	Net-rough+45	Hough-var	Hough-pow	Fourier
±30°-all	**3.00**	4.00	19.74	19.64	11.41	10.62	10.66
±30°-easy	**2.17**	2.83	19.48	17.90	8.44	7.04	8.64
±30°-hard	**4.26**	5.75	20.12	22.24	15.88	15.99	13.69
±45°-all	-	**4.63**	20.64	19.24	16.92	13.06	16.51
±45°-easy	-	**3.24**	21.26	19.29	14.08	9.01	13.32
±45°-hard	-	**6.71**	19.70	19.15	21.16	19.13	21.28
±180°-all	-	-	**20.97**	**18.68**	-	-	-
±180°-easy	-	-	**20.29**	**18.70**	-	-	-
±180°-hard	-	-	**21.98**	**18.65**	-	-	-

It was noted that fine tuning the estimation by feeding it through the separate network stages resulted in the best results, rather than using a single estimator (±30°, ±45°, or ±180°).

Classification

A classification solution would produce a vector of 360 values instead of a single answer. Each value represents a probability between zero and one of this angle class being correct. The optimal angle would be the argmax of this vector. The loss in this implementation will be a multi-class cross entropy loss. The Pytorch implementation of cross entropy loss utilizes the output of a softmax operation to convert outputs to probabilities that sum to one.