IE 534/CS 547 Deep Learning

University of Illinois at Urbana-Champaign

Fall 2019

Lecture 8

- Recurrent neural networks are widely used to model sequential or temporal dependence.
- Prominent examples include speech and text recognition.
- Sequence of data $(X_1, Y_1), \ldots, (X_t, Y_t), \ldots$.
- The goal is to estimate a model $F_t(X_{s \leq t}; \theta)$ to predict Y_t for all times t.
- The model is allowed to depend upon the history of the data sequence X_t.

At a high level, a recurrent neural network is of the form

$$
\begin{aligned}
\hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta), \\
S_t &= f_S(X_t, S_{t-1}; \theta),
\end{aligned} \tag{1}
$$

where $X_t \in \mathbb{R}^d$ is the input data at time t, $\hat{Y}_t \in \mathbb{R}^K$ is the prediction for Y_t, $S_t \in \mathbb{R}^{d_S}$ is the "internal state" of the model, $f_Y : \mathbb{R}^d \times \mathbb{R}^{d_S} \times \Theta \to \mathbb{R}^K$, and $f_S : \mathbb{R}^d \times \mathbb{R}^{d_S} \times \Theta \to \mathbb{R}^{d_S}$.

S_{t-1} is a **nonlinear representation** of the previous data $X_{t-1}, X_{t-2}, \dots, X_1$.

The objective function for estimating θ is

$$
\mathcal{L}(\theta) = \sum_{t=1}^{T} \rho(Y_t, \hat{Y}_t), \tag{2}
$$

Advantages of recurrent networks:

- A naive implementation $F_t(X_1, \ldots, X_t; \theta)$ is impractical since the input size changes at each time t.
- For long sequences, it is impractical to include such a long input vector due to computational constraints.
- The size of the model will drastically increase as $t \to \infty$, which could cause the model to overfit during training.

$$
\begin{aligned}
\hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta), \\
S_t &= f_S(X_t, S_{t-1}; \theta),
\end{aligned} \tag{3}
$$

- $f_S(\cdot; \theta)$ and $f_Y(\cdot; \theta)$ are neural networks.
- The parameters θ are **shared** across times $t = 1, 2, \ldots, T$.
- $S_1, \ldots, S_T$ depend upon θ in a complicated way.
- The backpropagation rule must be carefully derived.

Sharing parameters over time:

- Similar to convolution networks!
- Parsimonious model specification
- Reduces overfitting

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \rho(Y_t, \hat{Y}_t). \tag{4}$$

- The computational cost of the backpropagation algorithm is $\mathcal{O}(T)$.
- For long sequences, the computational cost is prohibitively large.
- To address this, an *ad hoc* method called "Truncated Backpropagation through Time" is used in practice.

The objective function at the k-th iteration is:

$$\mathcal{L}^k(\theta^{(k)}) = \sum_{t=\tau k+1}^{(k+1)\tau} \rho(Y_t, \hat{Y}_t), \tag{5}$$

The **truncation length** is $\tau \ll T$.

$$\begin{aligned} \hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta^{(k)}), \quad \tau k + 1 \le t \le (k+1)\tau, \\ S_t &= f_S(X_t, S_{t-1}; \theta^{(k)}), \quad \tau k + 1 \le t \le (k+1)\tau. \end{aligned} \tag{6}$$

Notice that

$$S_{\tau k} = g\left(\theta^{(k-1)}\right). \tag{7}$$

The SGD update is

$$\theta^{(k+1)} = \theta^{(k)} - \alpha^{(k)} \nabla_{\theta^{(k)}} \mathcal{L}^k(\theta^{(k)}). \tag{8}$$

$$
\begin{aligned}
\mathcal{L}^k(\theta^{(k)}) &= \sum_{t=\tau k+1}^{(k+1)\tau} \rho(Y_t, \hat{Y}_t), \\
\theta^{(k+1)} &= \theta^{(k)} - \alpha^{(k)} \nabla_{\theta^{(k)}} \mathcal{L}^k(\theta^{(k)}).
\end{aligned} \tag{9}
$$

Unlike traditional SGD, “Truncated Backpropagation through Time” is **biased**!

Nonetheless, it works well in practice.

The computational cost is $\mathcal{O}(\tau)$.

How do we implement mini-batch Truncated Backpropagation through Time?

- Truncated Backpropagation through Time does not fully optimize over the full sequence of the data.
- However, information does flow forward from time t to **all** future times $t' > t$ via the internal state $S_{t'}$.
- Recurrent neural networks (RNN) can be viewed as very deep networks, where the number of layers is T (or τ).
- Vanishing gradient problem

PyTorch

$$\text{model.hidden} = (\ \text{model.hidden}[0].\text{detach}(), \text{model.hidden}[1].\text{detach}())$$

"Z.Detach()" breaks the computational chain, i.e. it treats Z as a constant.

PyTorch, with its define-by-run framework, allows for breaking computational graph at any time during training.

Backpropagation.

$$\begin{aligned}
\hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta), \\
S_t &= f_S(X_t, S_{t-1}; \theta), \\
\mathcal{L}(\theta) &= \sum_{t=1}^{T} \rho(Y_t, \hat{Y}_t).
\end{aligned} \tag{11}$$

The parameters θ are **shared** across times $t = 1, 2, \ldots, T$. In particular, $S_1, \ldots, S_T$ depend upon θ in a complicated way.

Let us begin by considering a generic function $G : \Theta^T \to \mathbb{R}$. That is, G is a function T inputs, each taking values in Θ. Then,

$$\begin{aligned} G(\theta, \ldots, \theta) &= G(\theta_1, \ldots, \theta_T), \\ \theta_t &= g(\theta), \quad t = 1, 2, \ldots, T, \end{aligned} \tag{12}$$

where g is the identity function $g(z) = z$. Then, from chain rule, we have that

$$\begin{aligned} \nabla_\theta G(\theta, \ldots, \theta) &= \sum_{t=1}^{T} \nabla_{\theta_t} G(\theta_1, \ldots, \theta_T) \frac{dg}{d\theta}(\theta) \\ &= \sum_{t=1}^{T} \nabla_{\theta_t} G(\theta_1, \ldots, \theta_T). \end{aligned} \tag{13}$$

Therefore, it is equivalent to derive the backpropagation rule for

$$\begin{aligned}\hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta_t), \\ S_t &= f_S(X_t, S_{t-1}; \theta_t), \\ \tilde{\mathcal{L}}(\theta_1, \ldots, \theta_T) &= \sum_{t=1}^{T} \rho(Y_t, \hat{Y}_t), \\ \theta_t &= g(\theta). \end{aligned} \tag{14}$$

This can be viewed as a multi-layer network where θ_t is the parameter vector for the t-th layer!

We therefore have that

$$\begin{aligned}
\nabla_\theta \mathcal{L}(\theta) &= \nabla_\theta \sum_{t=1}^{T} \rho\Big(Y_t, f_Y(X_t, S_{t-1}; \theta_t)\Big) \\
&= \sum_{t=1}^{T} \nabla_{\theta_t} \tilde{\mathcal{L}}(\theta_1, \ldots, \theta_T). \qquad (15)
\end{aligned}$$

Let's begin by considering $\nabla_{\theta_\tau} \tilde{\mathcal{L}}(\theta_1, \ldots, \theta_T)$. If $\tau = T$,

$$\begin{aligned}
\nabla_{\theta_\tau} \tilde{\mathcal{L}}(\theta_1, \ldots, \theta_T) &= \nabla_{\theta_\tau} \sum_{t=1}^{T} \rho(Y_t, \hat{Y}_t) \\
&= \nabla_{\theta_T} \rho(Y_T, \hat{Y}_T) \\
&= \frac{\partial \rho}{\partial \hat{Y}}(Y_T, \hat{Y}_T) \frac{\partial f_Y}{\partial \theta}(X_T, S_{T-1}; \theta_T).
\end{aligned}$$

(16)

If $\tau < T$,

$$
\begin{aligned}
& \nabla_{\theta_\tau} \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T) = \nabla_{\theta_\tau} \sum_{t=1}^{T} \rho(Y_t, \hat{Y}_t) \\
= \quad & \nabla_{\theta_\tau} \sum_{t=\tau}^{T} \rho(Y_t, \hat{Y}_t) \\
= \quad & \frac{\partial \rho}{\partial \hat{Y}}(Y_\tau, \hat{Y}_\tau) \frac{\partial f_Y}{\partial \theta}(X_\tau, S_{\tau-1}; \theta_\tau) + \nabla_{\theta_\tau} \sum_{t=\tau+1}^{T} \rho(Y_t, \hat{Y}_t) \\
= \quad & \frac{\partial \rho}{\partial \hat{Y}}(Y_\tau, \hat{Y}_\tau) \frac{\partial f_Y}{\partial \theta}(X_\tau, S_{\tau-1}; \theta_\tau) + \nabla_{S_\tau} \tilde{\mathcal{L}}(\theta_1, \dots, \theta_T) \frac{\partial S_\tau}{\partial \theta_\tau}.
\end{aligned}
\tag{17}
$$

Combining results yields

$$\begin{aligned}\nabla_\theta \mathcal{L}(\theta) &= \sum_{\tau=1}^{T} \nabla_{\theta_\tau} \tilde{\mathcal{L}}(\theta_1, \ldots, \theta_T) \\ &= \sum_{\tau=1}^{T} \frac{\partial \rho}{\partial \hat{Y}}(Y_\tau, \hat{Y}_\tau) \frac{\partial f_Y}{\partial \theta}(X_\tau, S_{\tau-1}; \theta_\tau) \\ &+ \sum_{\tau=1}^{T-1} \nabla_{S_\tau} \tilde{\mathcal{L}}(\theta_1, \ldots, \theta_T) \frac{\partial S_\tau}{\partial \theta_\tau}. \end{aligned} \tag{18}$$

Define

$$\delta^t = \nabla_{S_t} \tilde{\mathcal{L}}(\theta_1, \ldots, \theta_T). \tag{19}$$

Recall that

$$\begin{aligned}
\hat{Y}_t &= f_Y(X_t, S_{t-1}; \theta_t), \\
S_t &= f_S(X_t, S_{t-1}; \theta_t), \\
\tilde{\mathcal{L}}(\theta_1, \ldots, \theta_T) &= \sum_{t=1}^{T} \rho(Y_t, \hat{Y}_t), \\
\theta_t &= g(\theta).
\end{aligned} \tag{20}$$

By chain rule, for $t = 1, \ldots, T-1$,

$$\begin{aligned}
\delta^t &= \frac{\partial f_S}{\partial S}(X_{t+1}, S_t; \theta_{t+1}) \delta^{t+1} \\
&+ \frac{\partial f_Y}{\partial S}(X_{t+1}, S_t; \theta_{t+1}) \frac{\partial \rho}{\partial \hat{Y}}(Y_{t+1}, \hat{Y}_{t+1}),
\end{aligned} \tag{21}$$

and $\delta^T = 0$.

Therefore,

$$\begin{aligned} \nabla_\theta \mathcal{L}(\theta) &= \sum_{t=1}^{T} \frac{\partial \rho}{\partial \hat{Y}}(Y_t, \hat{Y}_t) \frac{\partial f_Y}{\partial \theta}(X_t, S_{t-1}; \theta_t) \\ &+ \sum_{t=1}^{T-1} \delta^t \frac{\partial S_t}{\partial \theta_t}, \end{aligned} \tag{22}$$

where

$$\begin{aligned} \delta^t &= \frac{\partial f_S}{\partial S}(X_{t+1}, S_t; \theta_{t+1}) \delta^{t+1} \\ &+ \frac{\partial f_Y}{\partial S}(X_{t+1}, S_t; \theta_{t+1}) \frac{\partial \rho}{\partial \hat{Y}}(Y_{t+1}, \hat{Y}_{t+1}), \end{aligned} \tag{23}$$

and $\delta^T = 0$.

$$\begin{aligned}\nabla_\theta \mathcal{L}(\theta) &= \sum_{t=1}^{T} \frac{\partial \rho}{\partial \hat{Y}}(Y_t, \hat{Y}_t) \frac{\partial f_Y}{\partial \theta}(X_t, S_{t-1}; \theta_t) \\ &+ \sum_{t=1}^{T-1} \delta^t \frac{\partial f_S}{\partial \theta}(X_t, S_{t-1}; \theta_t), \end{aligned} \tag{24}$$

since

$$\frac{\partial S_t}{\partial \theta_t} = \frac{\partial f_S}{\partial \theta}(X_t, S_{t-1}; \theta_t). \tag{25}$$

Therefore, the backpropagation algorithm is:

- For $t = 1, 2, \ldots, T$:
  - Calculate $(\hat{Y}_t, S_t)$
- $\delta^T = 0$ and set

$$G = \frac{\partial \rho}{\partial \hat{Y}}(Y_T, \hat{Y}_T)\frac{\partial f_Y}{\partial \theta}(X_T, S_{T-1}; \theta_T). \tag{26}$$

- For $t = T-1, T-2, \ldots, 1$:
  - Calculate

$$\begin{aligned} \delta^t &= \frac{\partial f_S}{\partial S}(X_{t+1}, S_t; \theta_{t+1})\delta^{t+1} \\ &+ \frac{\partial f_Y}{\partial S}(X_{t+1}, S_t; \theta_{t+1})\frac{\partial \rho}{\partial \hat{Y}}(Y_{t+1}, \hat{Y}_{t+1}). \end{aligned} \tag{27}$$

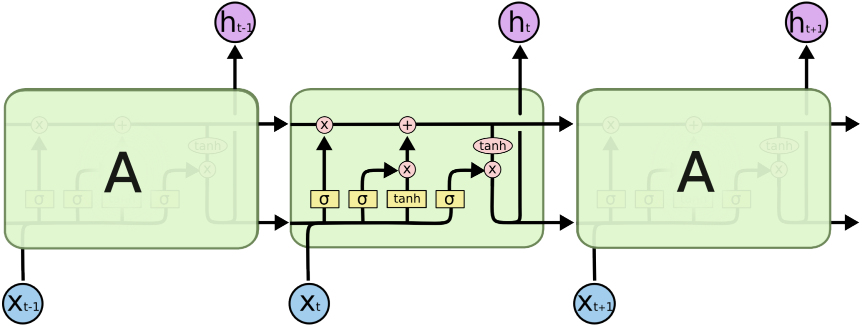
  - Calculate

$$G \to G + \frac{\partial \rho}{\partial \hat{Y}}(Y_t, \hat{Y}_t)\frac{\partial f_Y}{\partial \theta}(X_t, S_{t-1}; \theta_t) + \delta^t \frac{\partial f_S}{\partial \theta}(X_t, S_{t-1}; \theta_t). \tag{28}$$

- Take a gradient descent step in the direction $-G$.

The computational cost of the backpropagation through time (BPTT) algorithm is $\mathcal{O}(T)$.

Long short-term memory (LSTM) networks.



Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

The LSTM network architecture is:

$$
\begin{aligned}
f_t &= \sigma(b^f + U^f X_t + W^f h_{t-1}), \\
S_t &= f_t \odot S_{t-1} + g_t \odot \sigma(b + UX_t + Wh_{t-1}), \\
g_t &= \sigma(b^g + U^g X_t + W^g h_{t-1}), \\
h_t &= \tanh(S_t) \odot q_t, \\
q_t &= \sigma(b^o + U^o X_t + W^o h_{t-1}).
\end{aligned}
\tag{29}
$$

- LSTM networks
- Gated Recurrent Unit (GRU) networks

- Mini-batch version of Truncated Backpropagation through Time.

- Skip/shortcut connections

- Gradient clipping

$$\tilde{G} = \max\Big(-c, \min(G, c) \Big), \tag{30}$$

where $c > 0$.

Implementing LSTMs in PyTorch..

Residual Networks.

- Deep networks more challenging to train.
- Vanishing gradient problem
- Introduce skip/shortcut connections to facilitate the “flow of information” from lower layers to higher layers.

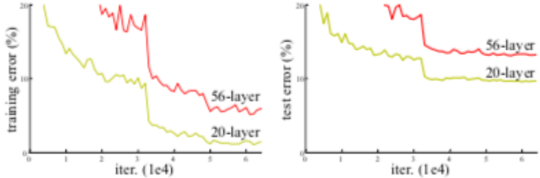


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Source: "Deep Residual Learning for Image Recognition" by He et al. (2015)

- If dimensions match, no extra parameters involved in the residual connection.

- If dimensions do not match, can use a linear connection with parameter $W^{s,\ell}$.

- Lower computational complexity than VGG network: 3.6 versus 19.6 billion FLOPs.

- Capable of building very deep networks (e.g., 150 layers).

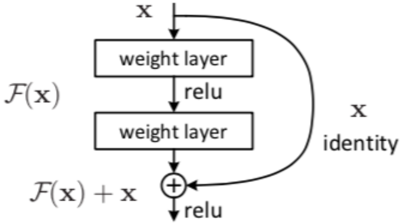


Figure 2. Residual learning: a building block.

Source: "Deep Residual Learning for Image Recognition" by He et al. (2015)


VGG-19
34-layer plain
34-layer residual

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Source: "Deep Residual Learning for Image Recognition" by He et al. (2015)

method	top-1 err.	top-5 err.
VGG [40] (ILSVRC'14)	-	8.43†
GoogLeNet [43] (ILSVRC'14)	-	7.89
VGG [40] (v5)	24.4	7.1
PReLU-net [12]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Source: "Deep Residual Learning for Image Recognition" by He et al. (2015)

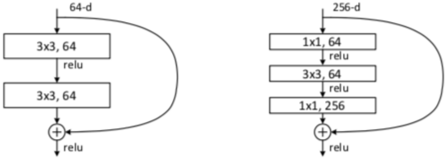
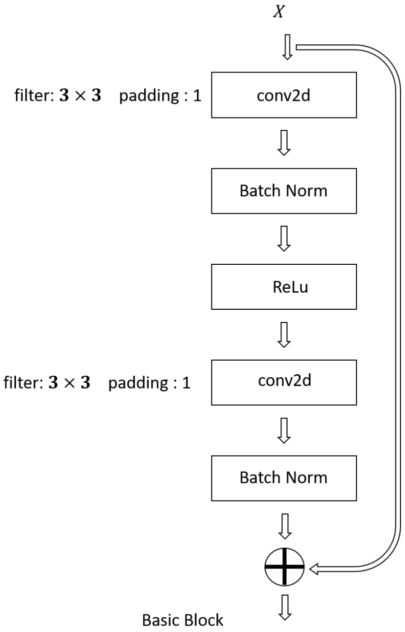


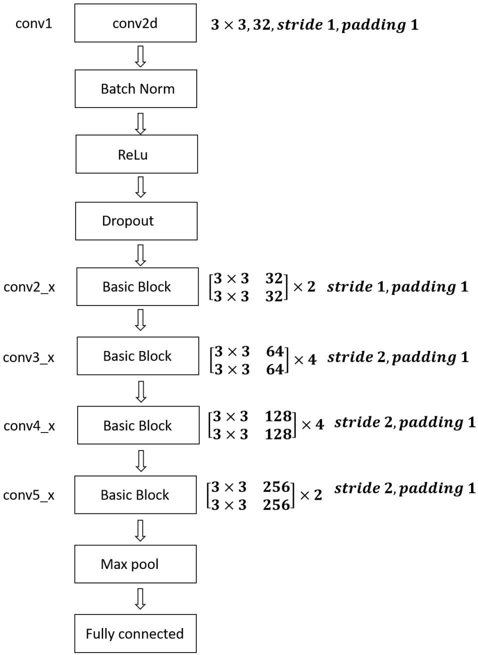
Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

Source: "Deep Residual Learning for Image Recognition" by He et al. (2015)

Homework 4.



X
filter: 3 × 3 padding : 1
conv2d
Batch Norm
ReLu
filter: 3 × 3 padding : 1
conv2d
Batch Norm
Basic Block


conv1
conv2d
3 × 3, 32, stride 1, padding 1
Batch Norm
ReLu
Dropout
conv2_x
Basic Block
[3 × 3 32; 3 × 3 32] × 2 stride 1, padding 1
conv3_x
Basic Block
[3 × 3 64; 3 × 3 64] × 4 stride 2, padding 1
conv4_x
Basic Block
[3 × 3 128; 3 × 3 128] × 4 stride 2, padding 1
conv5_x
Basic Block
[3 × 3 256; 3 × 3 256] × 2 stride 2, padding 1
Max pool
Fully connected

- CIFAR100 (60,000 images and 100 classes)
- Train a Residual Network from scratch
- Fine-tune a pre-trained residual network on CIFAR100
- Residual network is pre-trained on ImageNet (14 million images)

- "Densely Connected Networks" by Huang et al. (2016)
- Concatenate hidden layers $\ell - \tau, \ldots, \ell - 1$ as inputs to hidden layer ℓ.
- Memory intensive
- Small number of channels

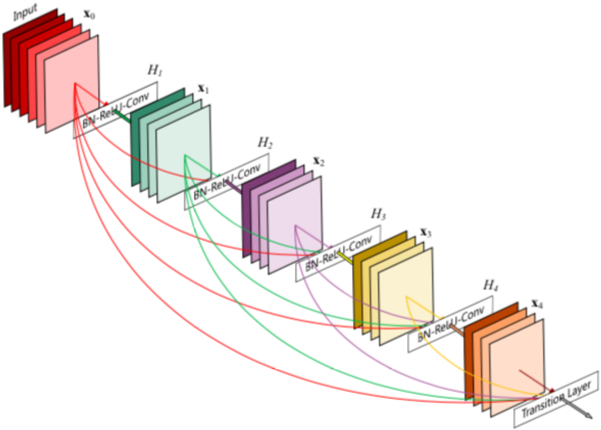


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Source: "Densely Connected Networks" by Huang et al. (2016)

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [31]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [33]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
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 [41]	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**	-

Source: "Densely Connected Networks" by Huang et al. (2016)