02/20/2019

Convolution Arithmetic
+
Batch Normalization

Baris Ozmen
AI Fellow | Insight

Introduction

convolution
padding
avg pooling
max pooling

Computing the output values of a convolution

Kernel (Filter)

0	1	2
2	2	0
0	1	2

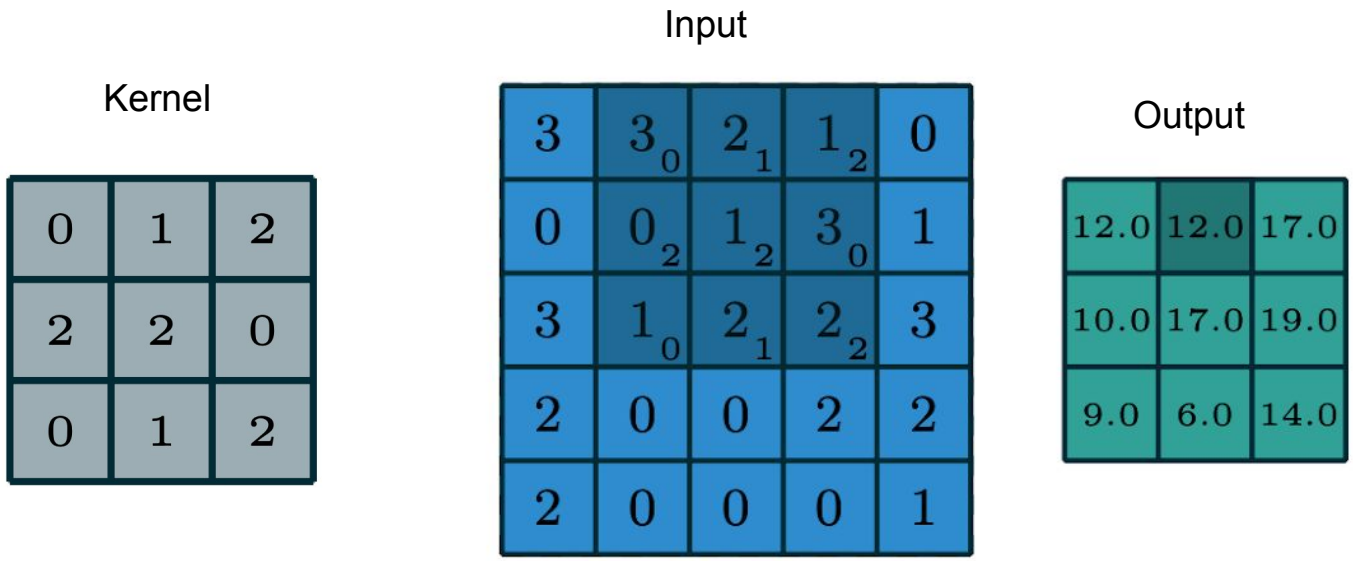
Input

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

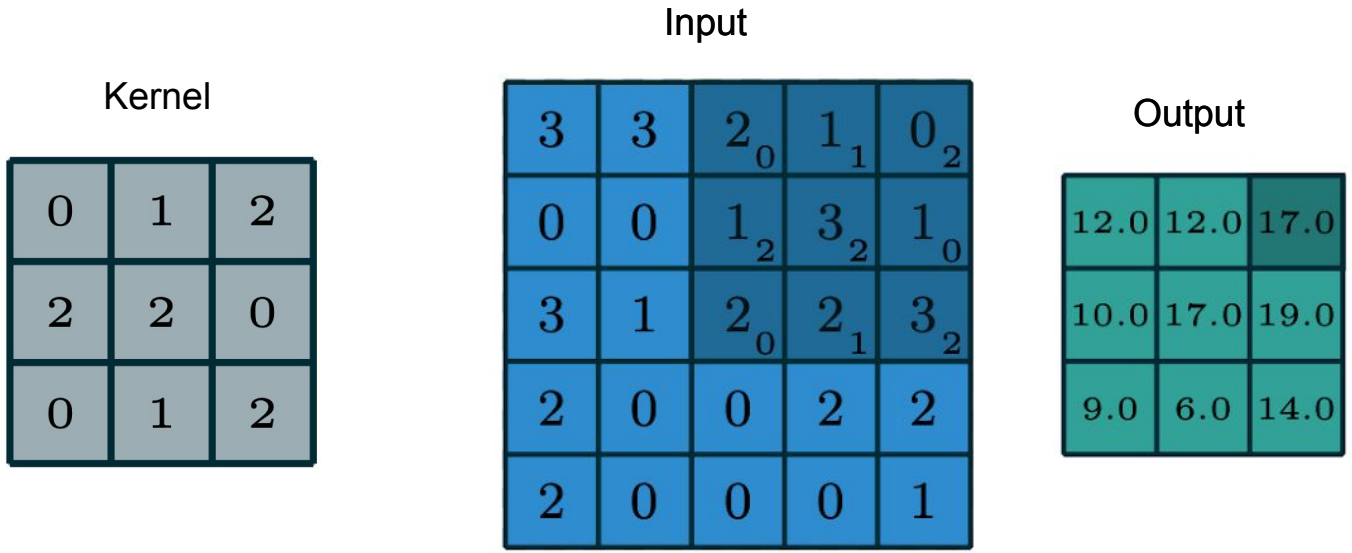
Output

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

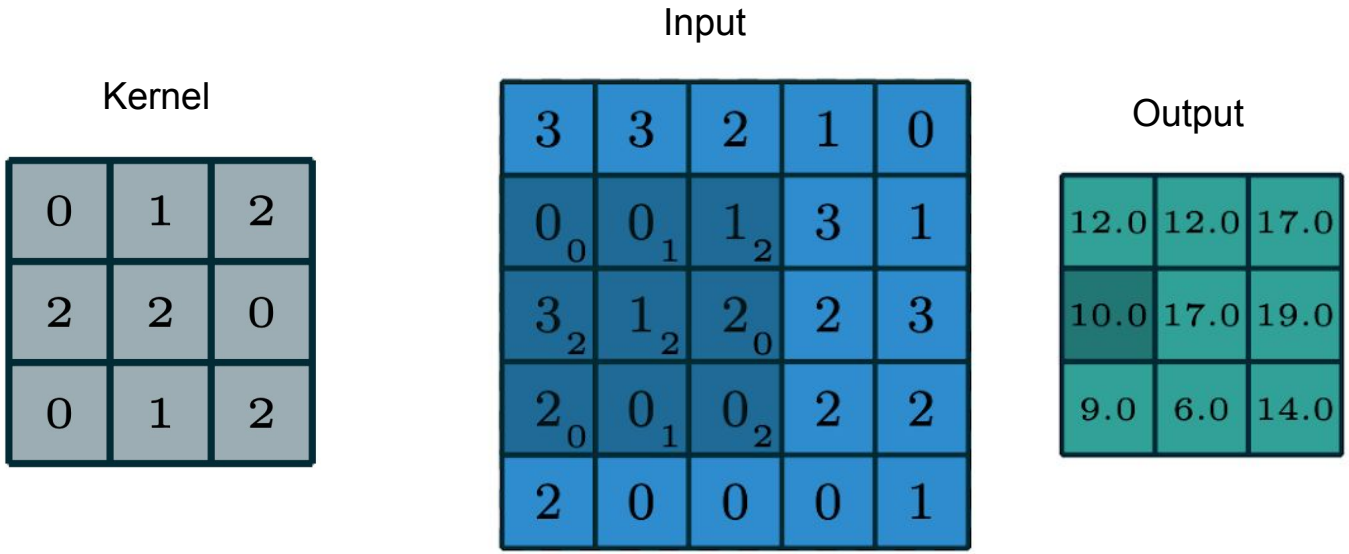
Computing the output values of a convolution



Computing the output values of a convolution

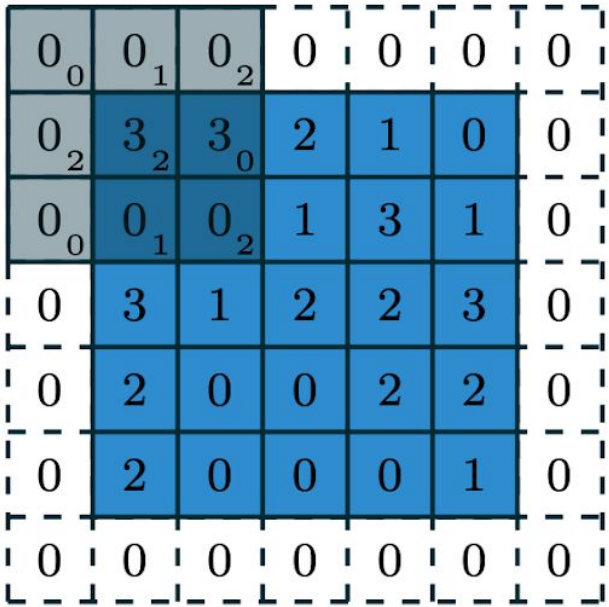


Computing the output values of a convolution



Computing the output values of a convolution

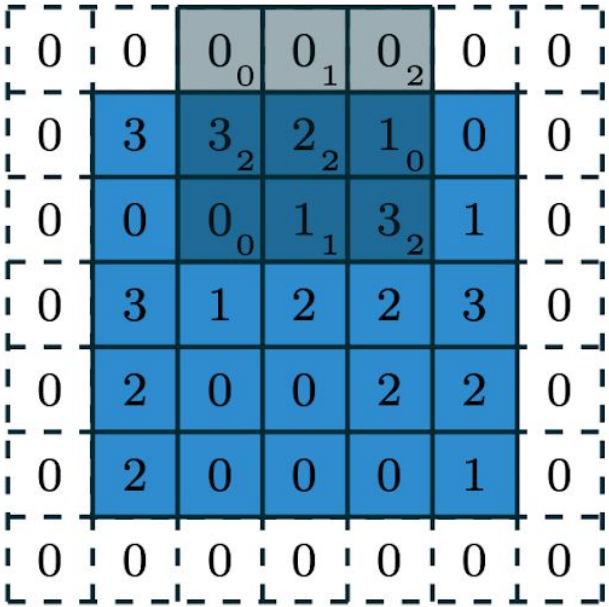
i	5
k	3
s	2
p	1
o	**3**

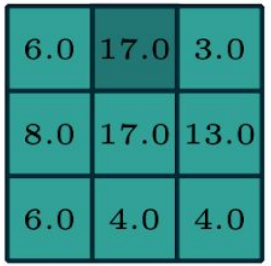


6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

Computing the output values of a convolution

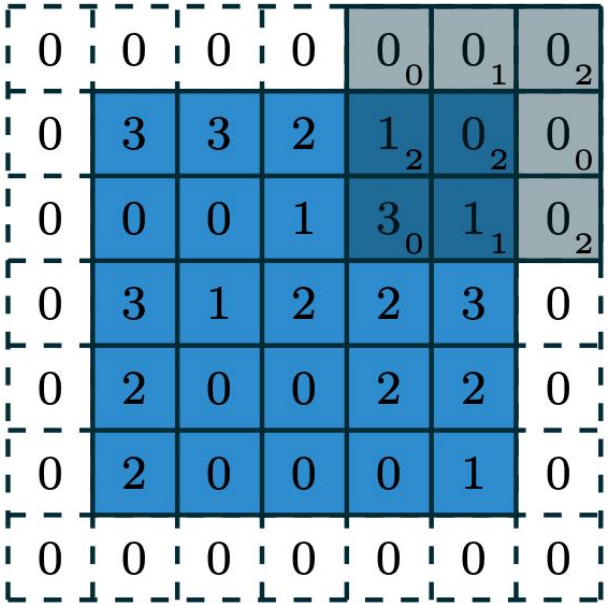
i	5
k	3
s	2
p	1
o	**3**





Computing the output values of a convolution

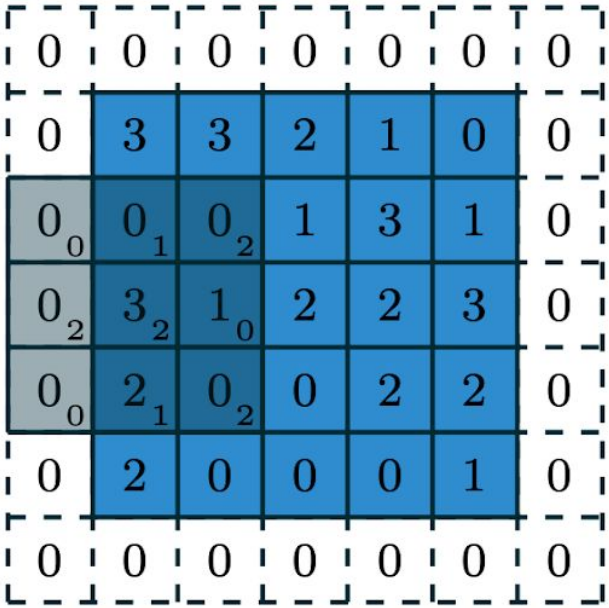
i	5
k	3
s	2
p	1
o	**3**



6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

Computing the output values of a convolution

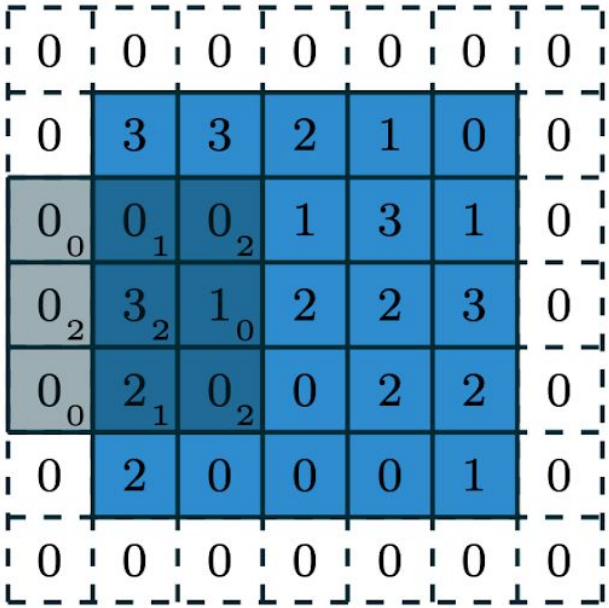
i	5
k	3
s	2
p	1
o	**3**



6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

Computing the output values of a convolution

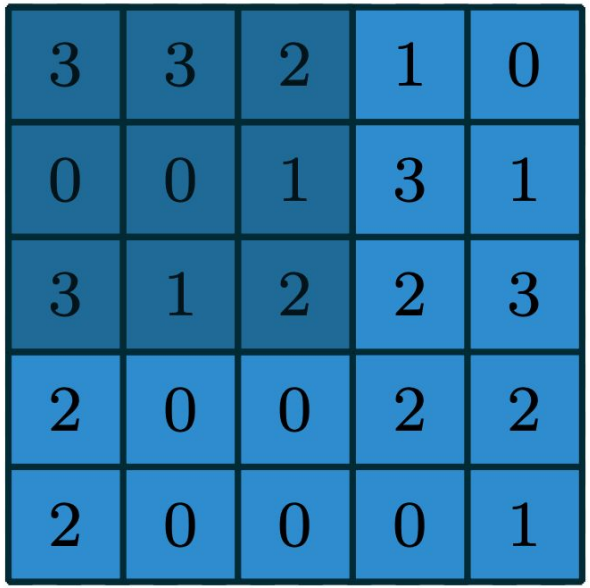
i	5
k	3
s	2
p	1
o	**3**

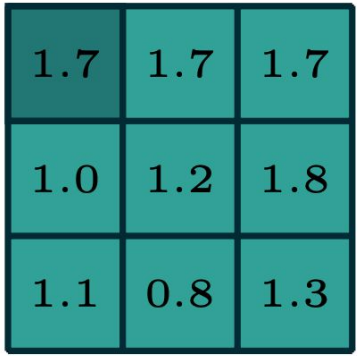


6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

Average Pooling

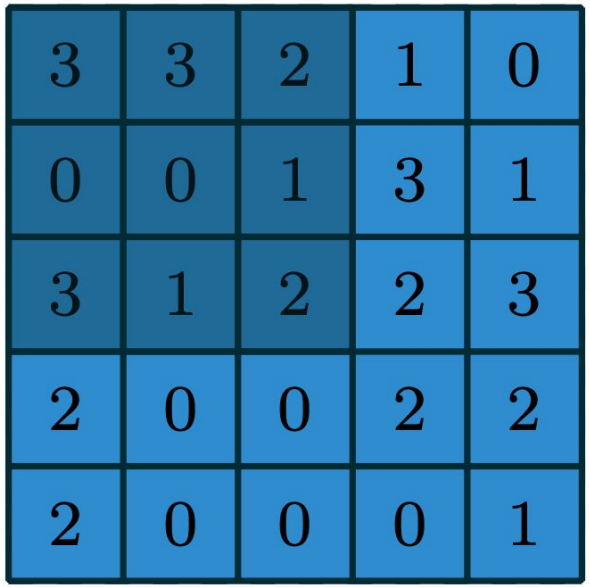
i	5
k	3
s	1
p	0
o	**3**

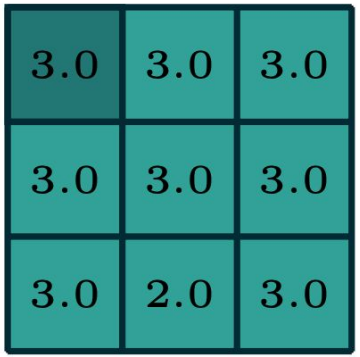




Max Pooling

i	5
k	3
s	1
p	0
o	**3**





Convolution Arithmetic

Padding (valid, half, full),
strides
Source code of Conv2D

Some terminology

VALID padding. The easiest case, means no padding at all. Just leave your data the same it was.

SAME padding sometimes called **HALF padding**. It is called *SAME* because for a convolution with a stride=1, (or for pooling) it should produce output of the same size as the input. It is called *HALF* because for a kernel of size `k`

$$p = \left\lfloor \frac{k}{2} \right\rfloor$$

FULL padding is the maximum padding which does not result in a convolution over just padded elements. For a kernel of size `k`, this padding is equal to `k - 1`.

Source: https://stackoverflow.com/a/44102413

We will assume ...

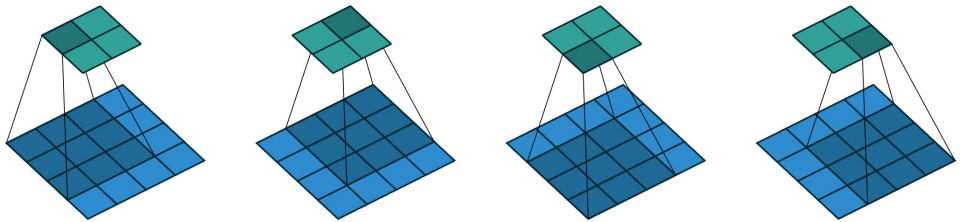
- 2-D discrete convolutions (N = 2)
- square [i]nputs (i1 = i2 = i)
- square [k]ernel size (k1 = k2 = k)
- same [s]trides along both axes (s1 = s2 = s)
- same zero [p]adding along both axes (p1 = p2 = p)
- [o]utput (o = f(i,k,s,p))

No padding, unit strides

Relationship 1. *For any i and k, and for $s = 1$ and $p = 0$,*

$$o = (i - k) + 1.$$

i	4
k	3
s	1
p	0
o	**2**

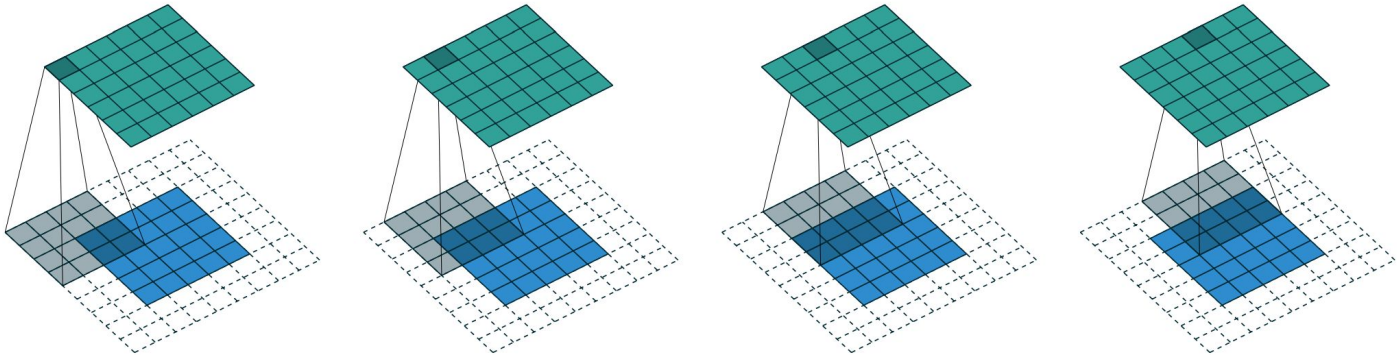


Arbitrary padding, unit strides

Relationship 2. *For any i, k and p, and for $s = 1$,*

$$o = (i - k) + 2p + 1.$$

i	5
k	4
s	1
p	2
o	**6**

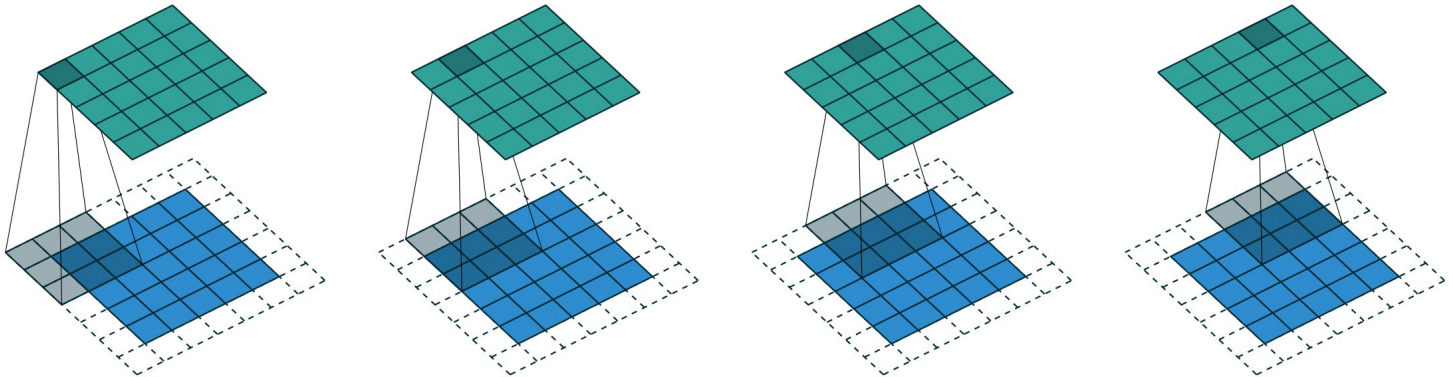


Half padding, unit strides

Relationship 3. *For any i and for k odd ($k = 2n + 1, \quad n \in \mathbb{N}$), $s = 1$ and $p = \lfloor k/2 \rfloor = n$,*

$$\begin{aligned} o &= i + 2\lfloor k/2 \rfloor - (k - 1) \\ &= i + 2n - 2n \\ &= i. \end{aligned}$$

i	5
k	3
s	1
p	1
o	**5**

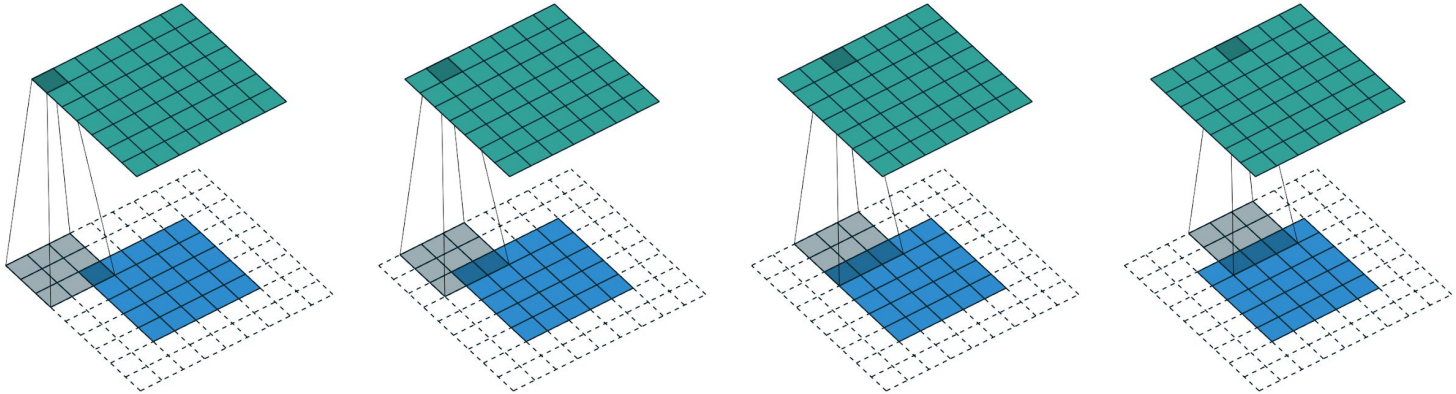


Full padding, unit strides

Relationship 4. *For any i and k, and for $p = k - 1$ and $s = 1$,*

$$o = i + 2(k-1) - (k-1)$$
$$= i + (k-1).$$

i	5
k	3
s	1
p	2
o	**7**

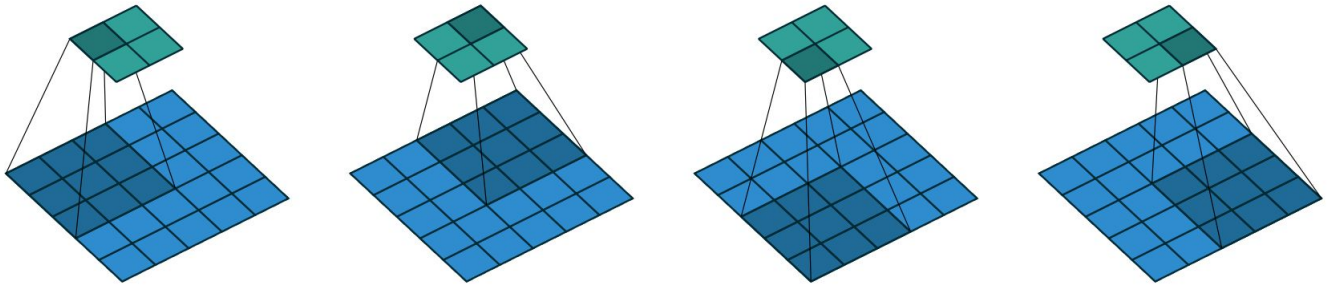


Zero padding, non-unit strides

Relationship 5. *For any i, k and s, and for $p = 0$,*

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1.$$

i	5
k	3
s	2
p	0
o	**2**

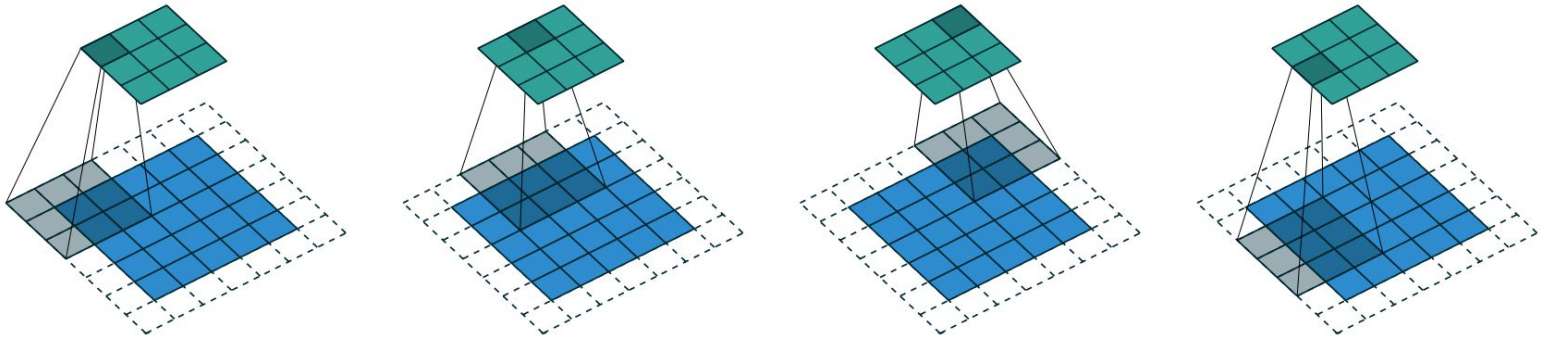


Non-zero padding, non-unit strides

Relationship 6. *For any i, k, p and s,*

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1.$$

i	5
k	3
s	2
p	1
o	**3**



Conv2D function in Keras

```
# arguments with default values
keras.layers.Conv2D(
      filters,
      kernel_size,
      strides=(1, 1),
      padding='valid',
      data_format=None,
      dilation_rate=(1, 1),
      activation=None,
      use_bias=True,
      kernel_initializer='glorot_uniform',
      bias_initializer='zeros',
      kernel_regularizer=None,
      bias_regularizer=None,
      activity_regularizer=None,
      kernel_constraint=None,
      bias_constraint=None
)
```

'Valid' or 'same'

Another way of saying 'stride'
dont use it if you set stride=1
probably no need to use this at all

I guess anything you can do with dilation_rate, can be done with 'stride' as well.
Any idea??

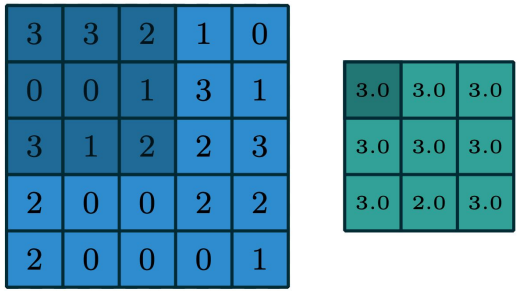
docu source

Pooling arithmetic

Relationship 7. *For any i, k and s,*

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1.$$

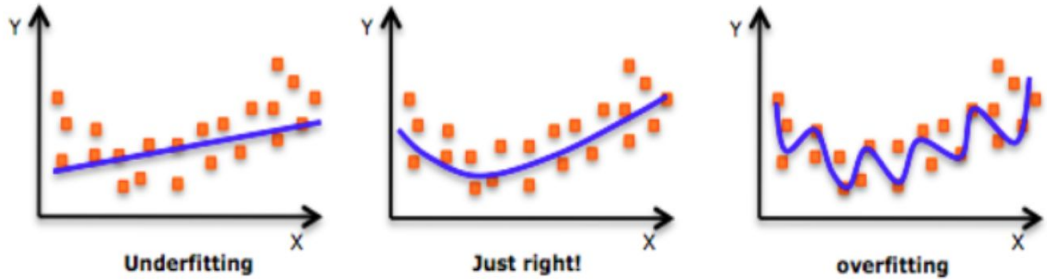
i	5
k	3
s	1
p	0
o	**3**

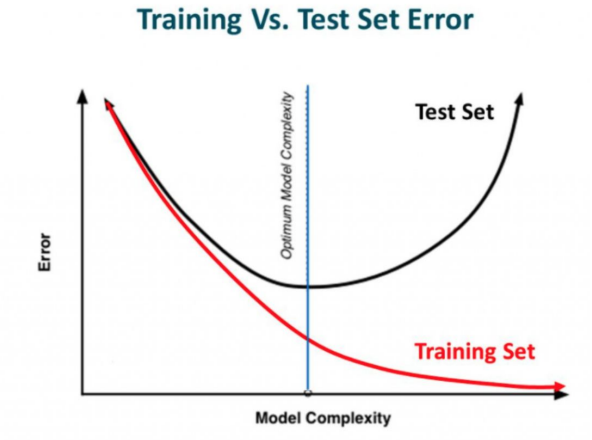


Note: Pooling doesn't include padding

Overview of overfitting

What is overfitting?





How to combat overfitting?

By data

1. More data
2. Data augmentation
3. Data synthesis

By architecture

1. Simplify model
2. Dropout
3. L1/L2/../L_inf
4. Batch normalization

By training decisions

1. Early stopping

How to combat overfitting?

By data

1. More data
2. Data augmentation
3. Data synthesis

By training decisions

1. Early stopping

By architecture

1. Simplify model
2. Dropout
3. L1/L2/../L_inf
4. **Batch normalization**

We'll discuss this approach today

Batch Normalization

When invented? = 2015

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

1 Introduction

Deep learning has dramatically advanced the state of the art in vision, speech, and many other areas. Stochastic gradient descent (SGD) has proved to be an effective way of training deep networks, and SGD variants such as momentum (Sutskever et al., 2013) and Adagrad

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience *covariate shift* (Shimodaira, 2000). This is typically handled via domain adaptation (Jiang, 2008). However, the notion of covariate shift can be extended beyond the learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider a network computing

$$\ell = F_2(F_1(\mathrm{u}, \Theta_1), \Theta_2)$$

where F_1 and F_2 are arbitrary transformations, and the parameters Θ_1, Θ_2 are to be learned so as to minimize the loss ℓ. Learning Θ_2 can be viewed as if the inputs $\mathrm{x} = F_1(\mathrm{u}, \Theta_1)$ are fed into the sub-network

What uses batch normalization?

Inception v2, v3

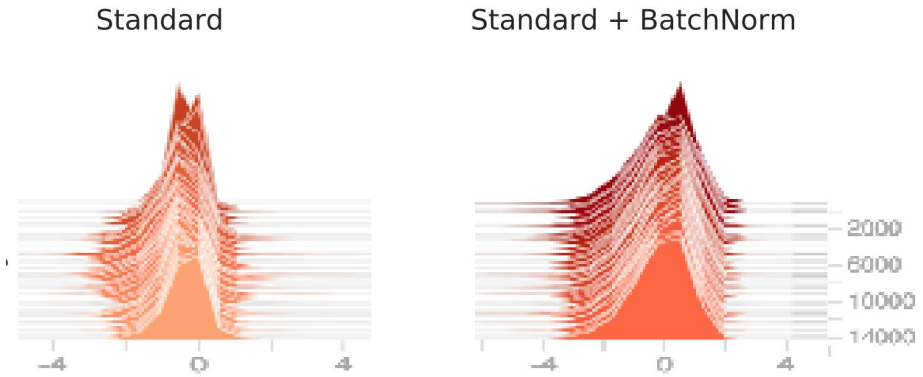
And most CNN architectures designed after 2015

It became very popular

What it does?

Normalizes the activations of the previous layer at each batch

i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1



How it works?

1. Calculate mean and variance of previous layer's activation

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{Batch mean}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \qquad \text{Batch variance}$$

2. Normalize activations by their mean and variance

$$\overline{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

3. Scale and shift them by variables **γ** and **β** (important: **γ** & **β** are trainable)

$$y_i = \gamma \overline{x_i} + \beta$$

Why it works?
(An academic discussion)

Original paper and many others think/thought that it helps overfitting because **reduces internal covariate shift (ICS)**

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

1 Introduction

Deep learning has dramatically advanced the state of the art in vision, speech, and many other areas. Stochastic gradient descent (SGD) has proved to be an effective way of training deep networks, and SGD variants such as momentum (Sutskever et al., 2013) and Adagrad

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience *covariate shift* (Shimodaira, 2000). This is typically handled via domain adaptation (Jiang, 2008). However, the notion of covariate shift can be extended beyond the learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider a network computing

$$\ell = F_2(F_1(u, \Theta_1), \Theta_2)$$

where F_1 and F_2 are arbitrary transformations, and the parameters Θ_1, Θ_2 are to be learned so as to minimize the loss ℓ. Learning Θ_2 can be viewed as if the inputs $x = F_1(u, \Theta_1)$ are fed into the sub-network

Why it works?
(An academic discussion)

A recent paper claims that:

- The reason it helps <u>not</u> that it reduces ICS
- It does not even reduce ICS (showed that)
- It works because **makes optimization landscape smoother**
    - induces predictive and stable behaviour of gradients, allowing faster training

How Does Batch Normalization Help Optimization?

Shibani Santurkar* MIT shibani@mit.edu
Dimitris Tsipras* MIT tsipras@mit.edu
Andrew Ilyas* MIT ailyas@mit.edu
Aleksander Mądry MIT madry@mit.edu

Abstract

Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift". In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

1 Introduction

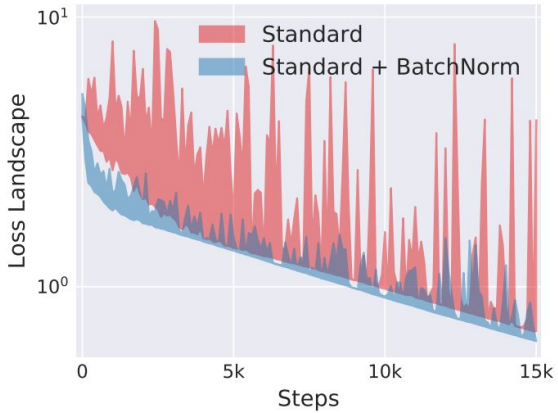
Over the last decade, deep learning has made impressive progress on a variety of notoriously difficult tasks in computer vision [16, 7], speech recognition [5], machine translation [29], and game-playing [18, 25]. This progress hinged on a number of major advances in terms of hardware, datasets [15, 23], and algorithmic and architectural techniques [27, 12, 20, 28]. One of the most prominent examples of such advances was batch normalization (BatchNorm) [10].

At a high level, BatchNorm is a technique that aims to improve the training of neural networks by stabilizing the distributions of layer inputs. This is achieved by introducing additional network layers that control the first two moments (mean and variance) of these distributions.

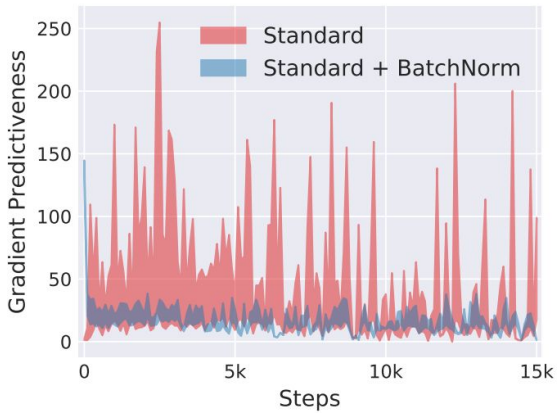
The practical success of BatchNorm is indisputable. By now, it is used by default in most deep learning models, both in research (more than 6,000 citations) and real-world settings. Somewhat shockingly, however, despite its prominence, we still have a poor understanding of what the effectiveness of BatchNorm is stemming from. In fact, there are now a number of works that provide alternatives to BatchNorm [1, 3, 13, 31], but none of them seem to bring us any closer to understanding this issue. (A similar point was also raised recently in [22].)

Currently, the most widely accepted explanation of BatchNorm's success, as well as its original motivation, relates to so-called *internal covariate shift* (ICS). Informally, ICS refers to the change in

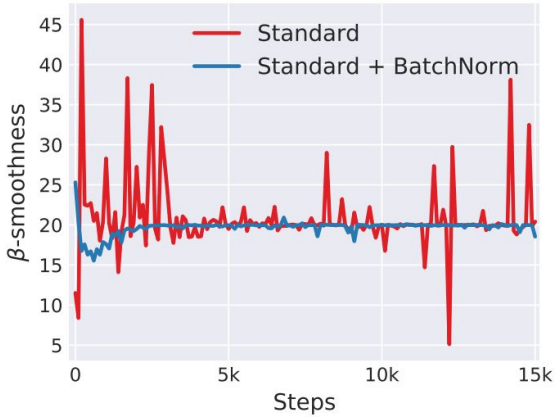
Batch normalization makes optimization landscape smoother



(a) loss landscape

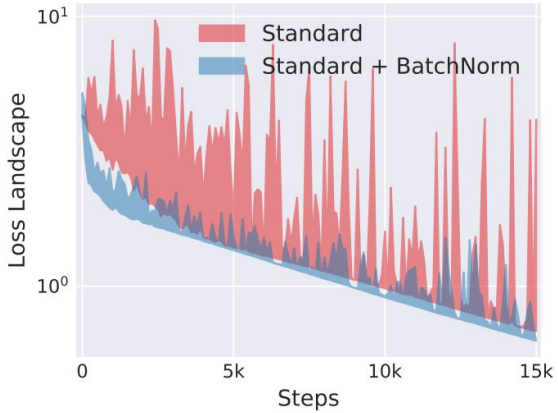


(b) gradient predictiveness

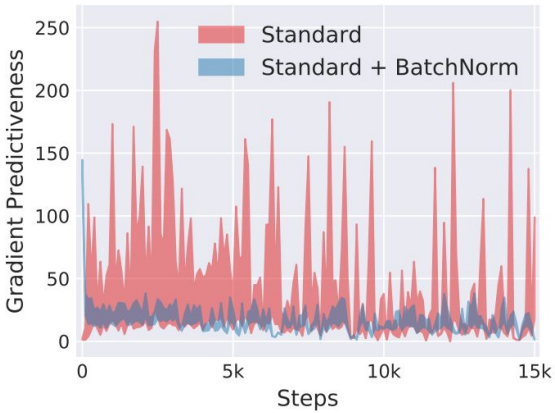


(c) "effective" β-smoothness

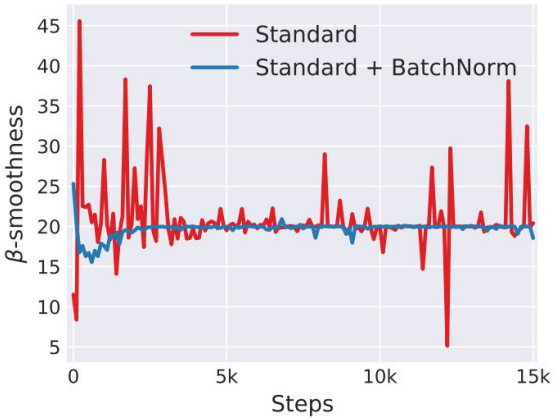
Batch normalization makes optimization landscape smoother



(a) loss landscape



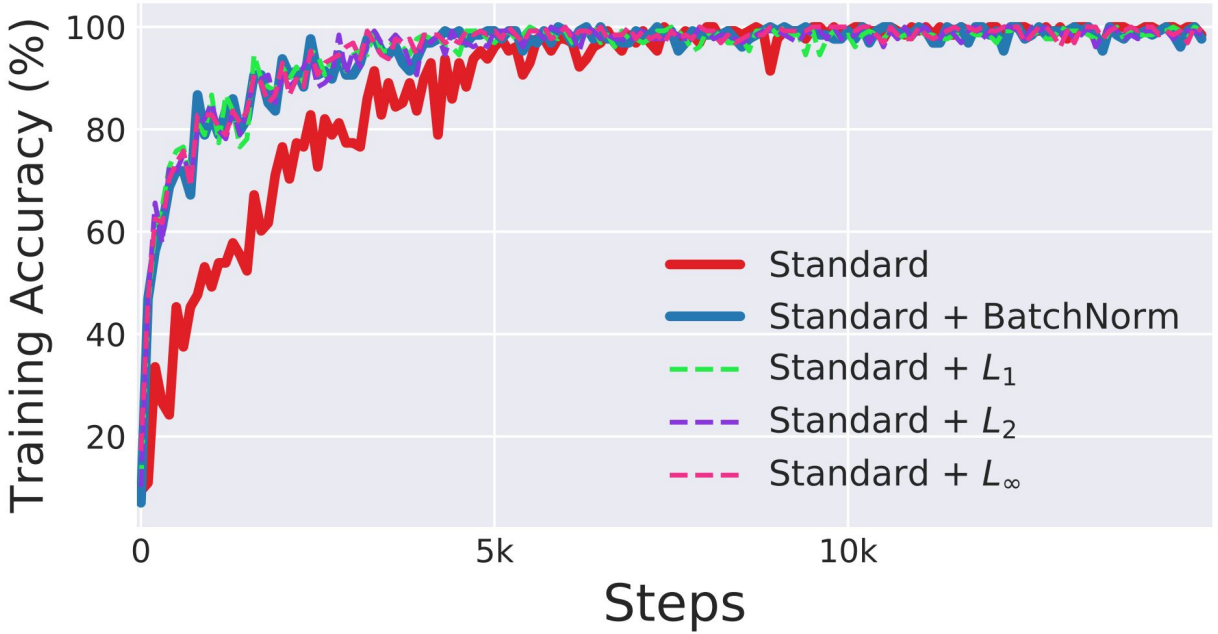
(b) gradient predictiveness



(c) "effective" β-smoothness

"… key impact that BatchNorm has on the training process: it reparametrizes the underlying optimization problem to make its landscape significantly more smooth. The first manifestation of this impact is improvement in the ***Lipschitzness*** ***of the loss function. That is, the loss changes at a smaller rate and the magnitudes of the gradients are smaller too****. .. BatchNorm's reparametrization makes gradients of the loss more Lipschitz too. In other words, the loss exhibits a significantly better "effective" β-smoothness"*

L_n regularizations performs similarly with (and sometimes even better than) BatchNorm



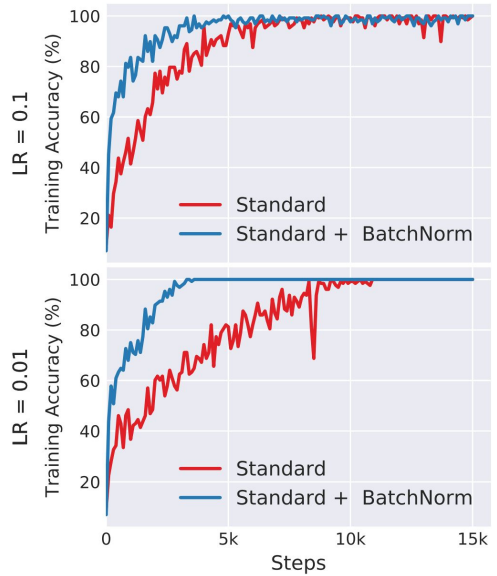
(a) VGG

Benefits of batch normalization

Reduce overfitting

Faster training

Make training more robust to different hyperparameters



BatchNormalization function in Keras

```
# arguments with default values
keras.layers.BatchNormalization(
      axis=-1,
      momentum=0.99,
      epsilon=0.001,
      center=True,
      scale=True,
      beta_initializer='zeros',
      gamma_initializer='ones',
      moving_mean_initializer='zeros',
      moving_variance_initializer='ones',
      beta_regularizer=None,
      gamma_regularizer=None,
      beta_constraint=None,
      gamma_constraint=None
)
```

docu
source

References

1. A guide to convolution arithmetic for deep learning
   (https://arxiv.org/pdf/1603.07285.pdf)

2. Ioeffe et al., 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
   (https://arxiv.org/abs/1502.03167)

3. Santurkar et al., 2018. How Does Batch Normalization Help Optimization?
   (https://arxiv.org/abs/1805.11604)

4. Same/Valid/Full padding
   (https://stackoverflow.com/a/44102413)

5. Batch normalization in neural networks
   (https://towardsdatascience.com/...)

6. Batch normalization: theory and how to use it with Tensorflow
   (https://towardsdatascience.com/...)