Session 3 & Assignment

Published Edit

Advanced Convolutions, Batch Normalization and Receptive Field

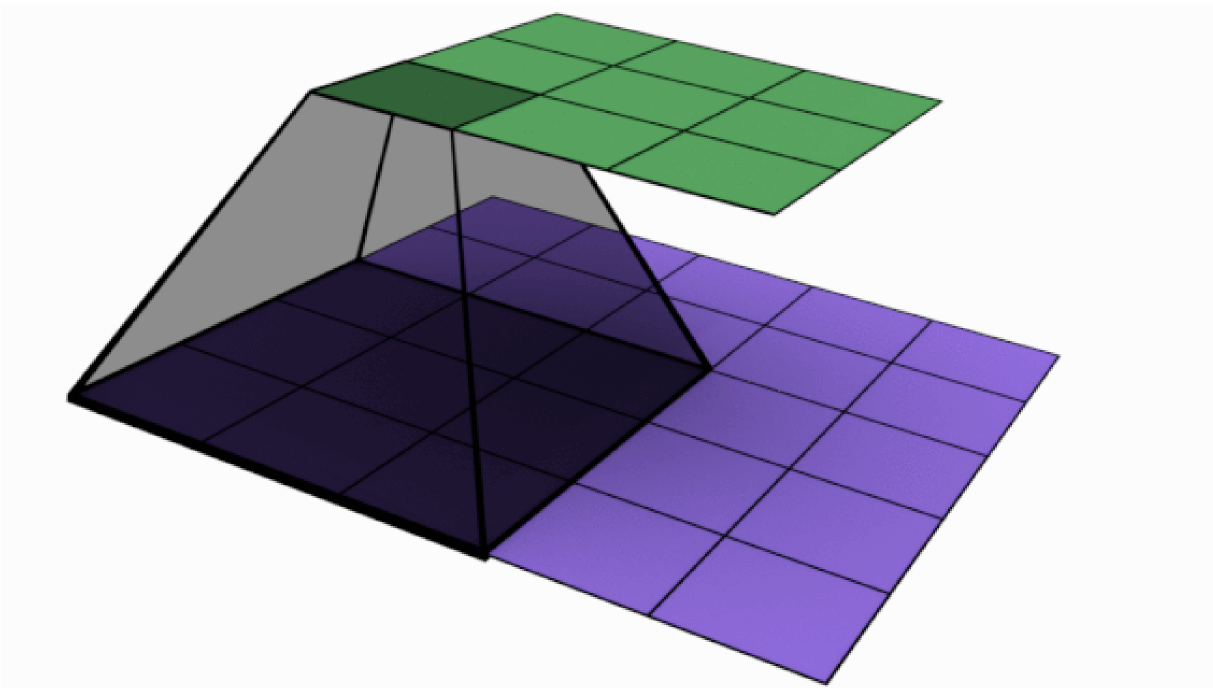
Session 3 Video

Advanced Convolutions

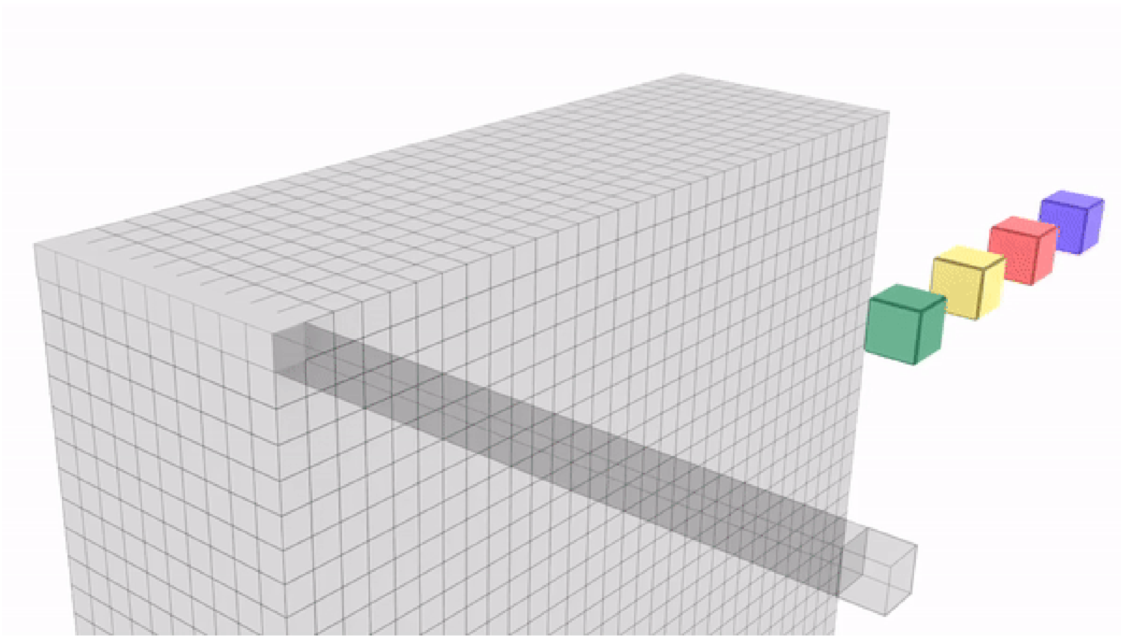
What is Convolution?

The process of extracting features from input data using kernels/filters. Filter moves discretely on top of data-plane/channel, scales the input data equal to the size of it's receptive field, sums this results and creates a new feature map.

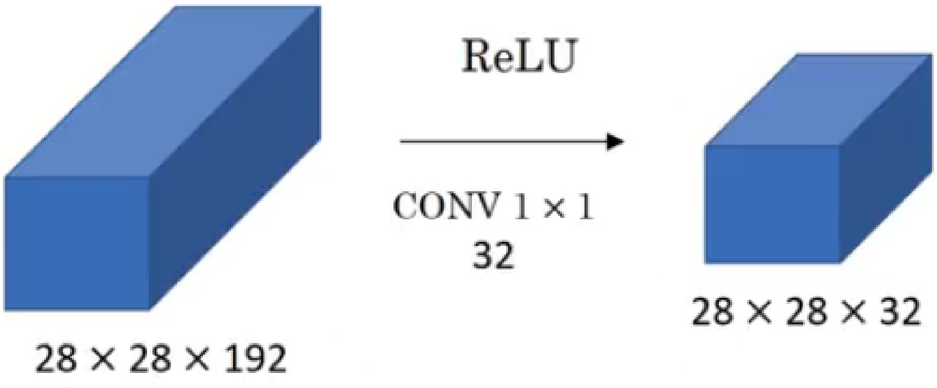
Normal Convolutions



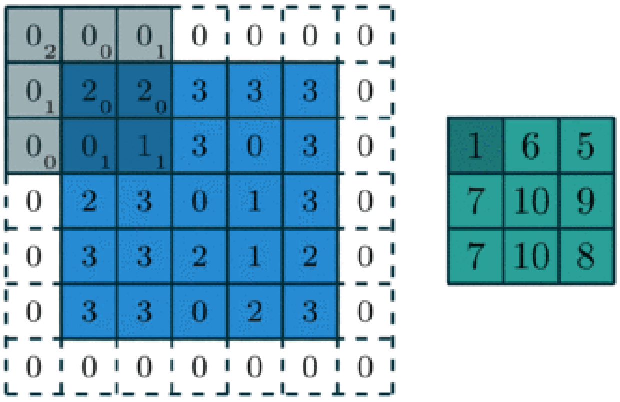
Pointwise Convolution



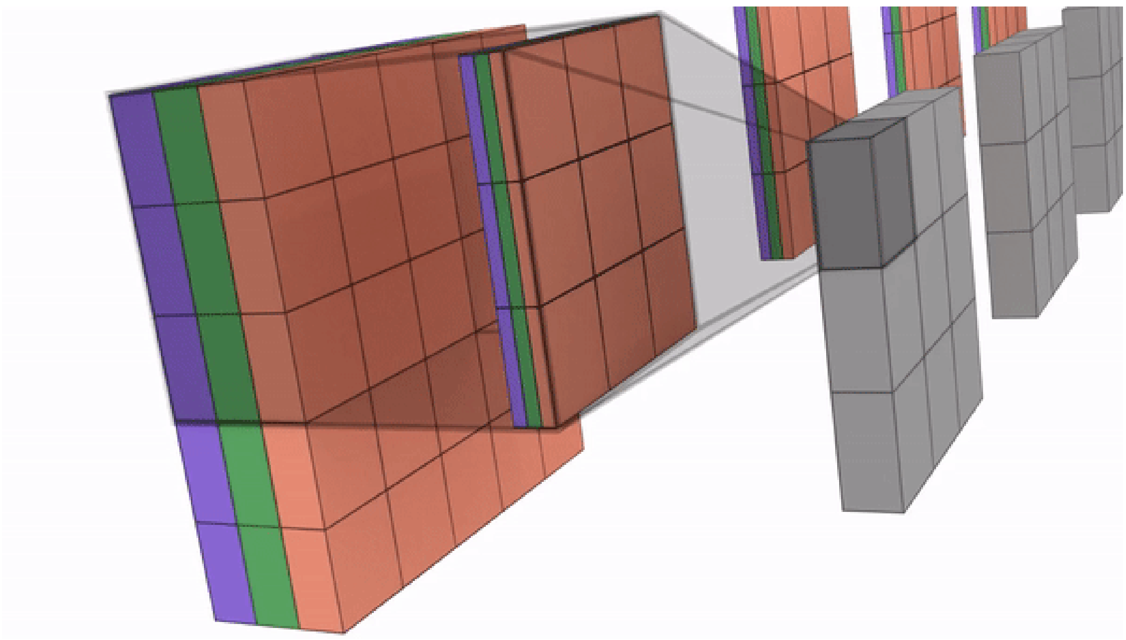
Basically



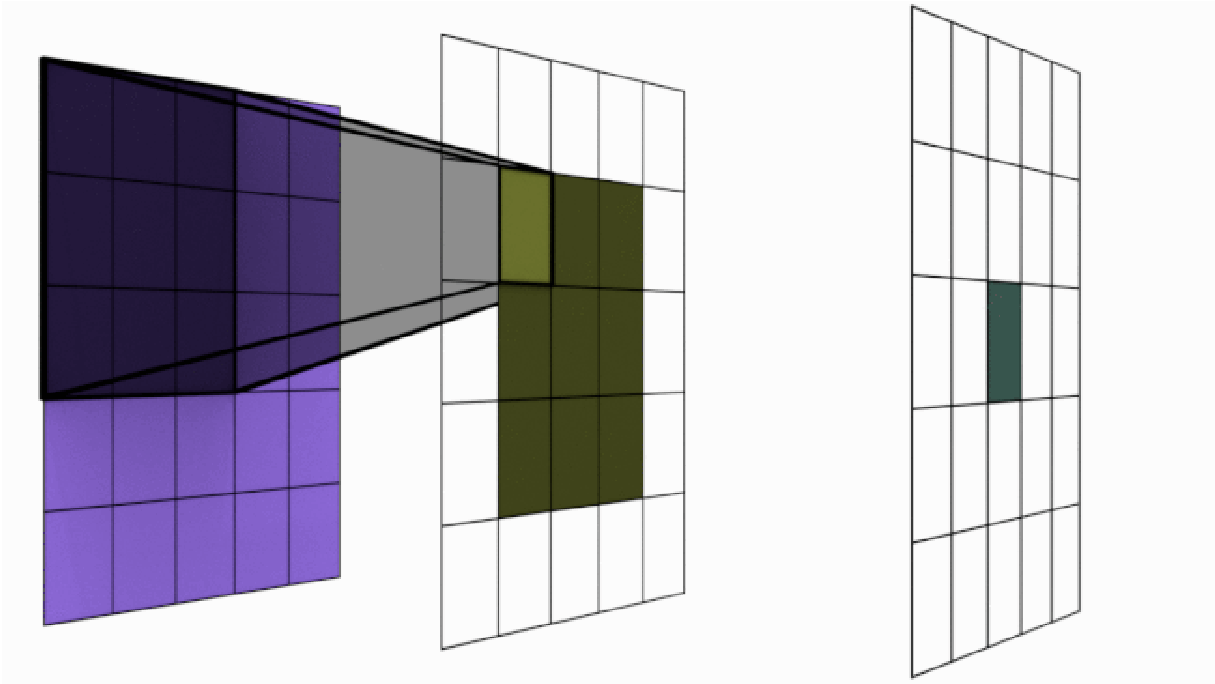
Behind the scenes:

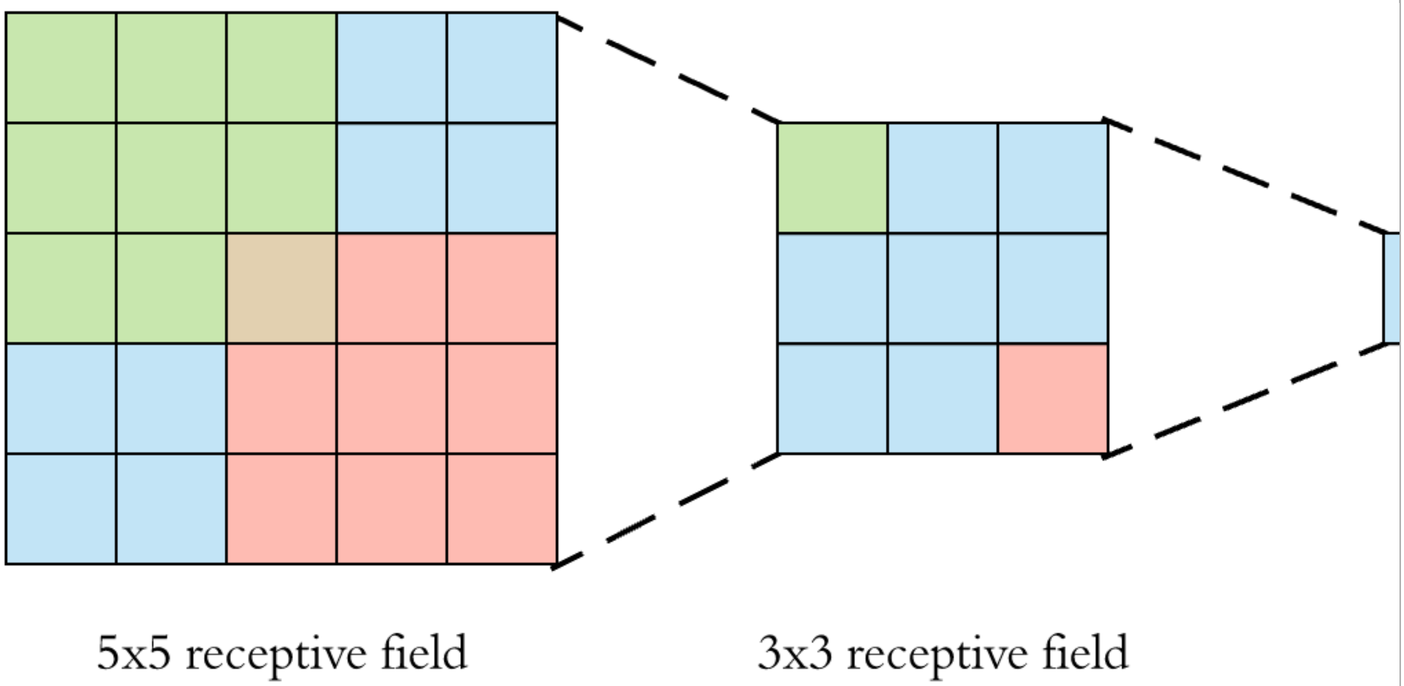


Concept of Channels

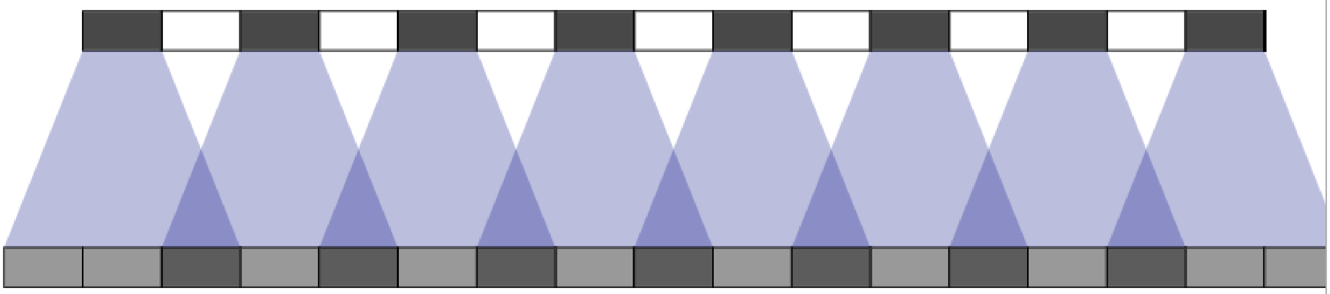


Receptive Field





Checkboard Issue



A Note on The Strides

- why should we not use strides?
- why should we use strides or when do we use them?

We get a receptive field of 3x3 when we convolve by 3x3. What should we do to get a receptive field of 5x5 when we convolve by 3x3?

Atrous Convolutions

Or

Dilated Convolution

Dilated convolution is a way of increasing receptive view (global view) of the network exponentially and linear parameter accretion. With this purpose, it finds usage in applications cares more about integrating the knowledge of the wider context with less cost.

The key application the dilated convolution authors have in mind is dense prediction: vision applications where the predicted object that has similar size and structure to the input image. For example, semantic segmentation with one label per pixel; image super-resolution, denoising, demosaicing, bottom-up saliency, keypoint detection, etc.

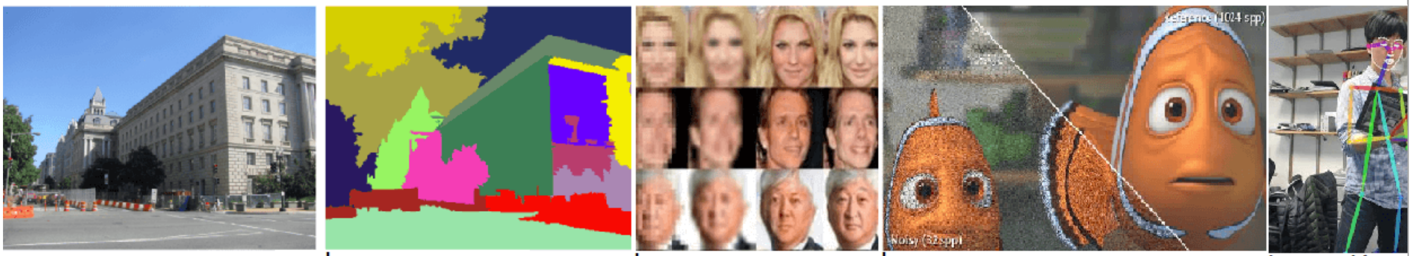
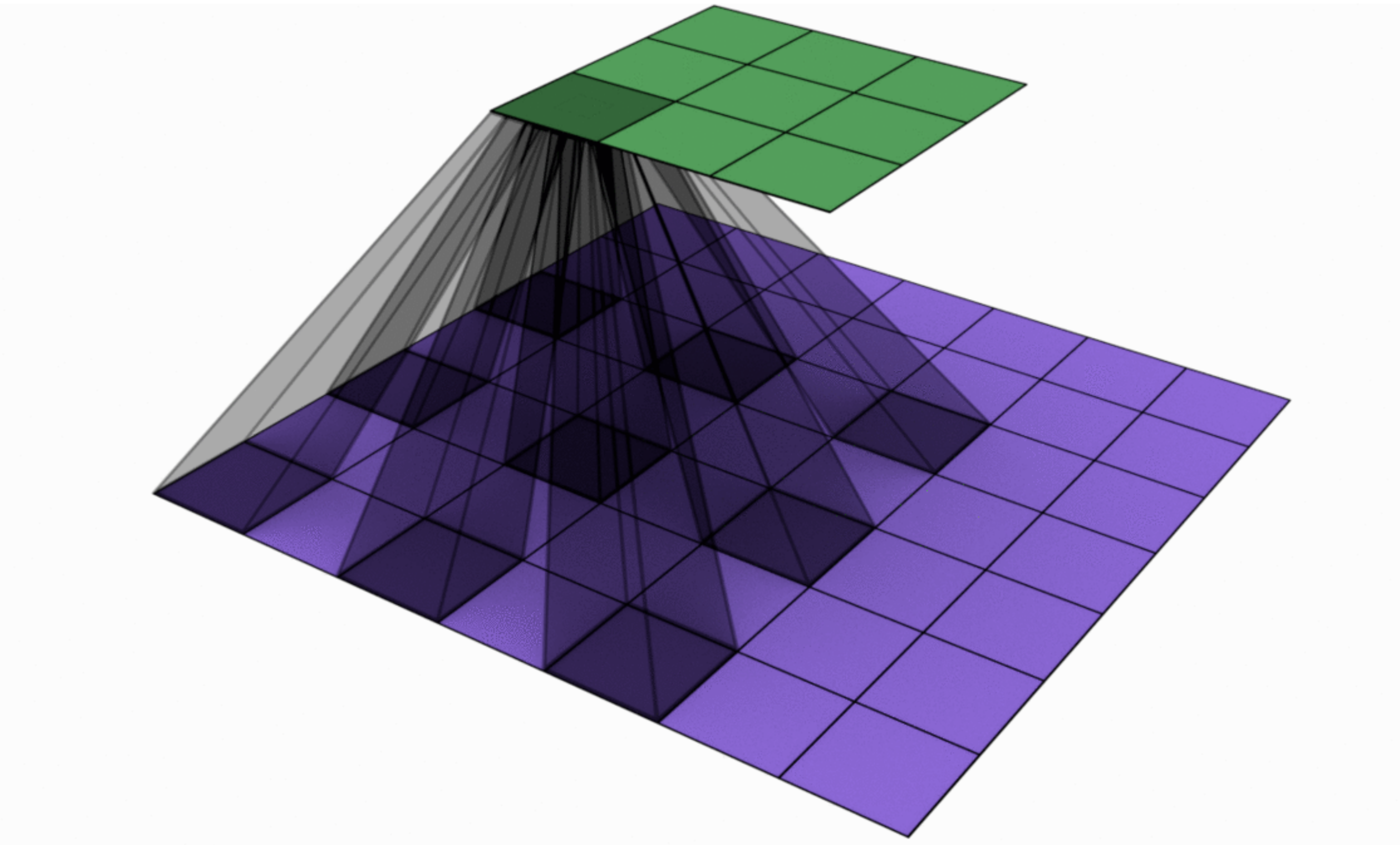


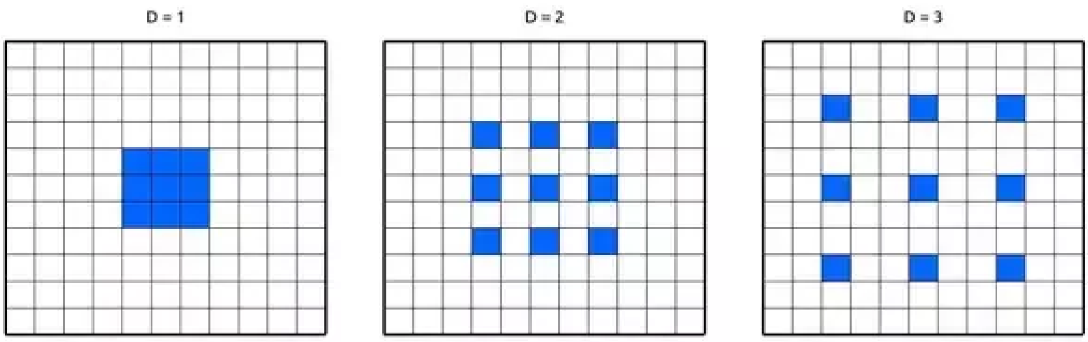
Image Segmentation | Image Superresolution | Image Denoising | Image Keyp Detection

In many such applications one wants to integrate information from different spatial scales and balance two properties:

1. local, pixel-level accuracy, such as precise detection of edges, and
2. integrating the knowledge of the wider, global context



Dilation:



The Problem: "convolution is a rather local operation"

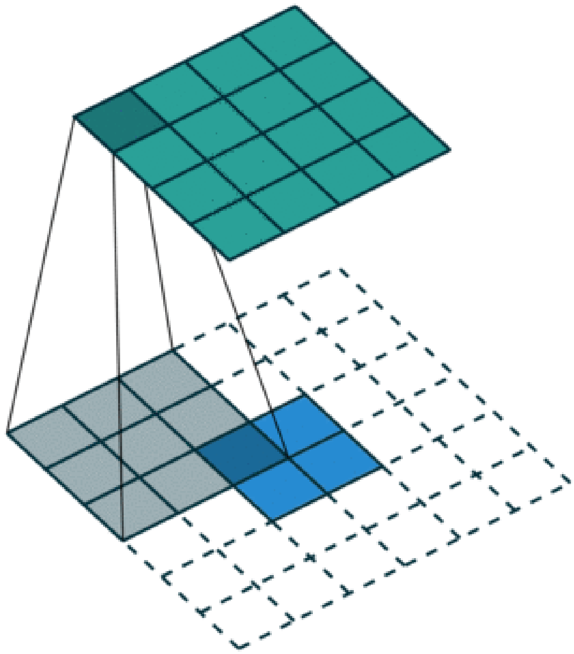
Improving the resolution of segmentation results

Dilated convolutions or atrous convolutions present a potential solution to this problem: they are capable of capturing global context without reducing the resolution of the segmentation map. Normal convolutional neural networks (CNNs) are based on the fact that each convolutional layer collects information from a larger neighborhood around each pixel/voxel. This means that in the upper layers of the network, each pixel of a feature map could potentially hold information about a large region of the image. However, experiments show that during learning this is usually not the case. Rather, the **information in each pixel stays localized** [source (https://arxiv.org/abs/1412.6856)] and the network "does not live up to its full potential". Dilated convolutions change the rules of the game by using kernels of the same size as normal convolutional layers but spread out over a larger area on the image (see animation).

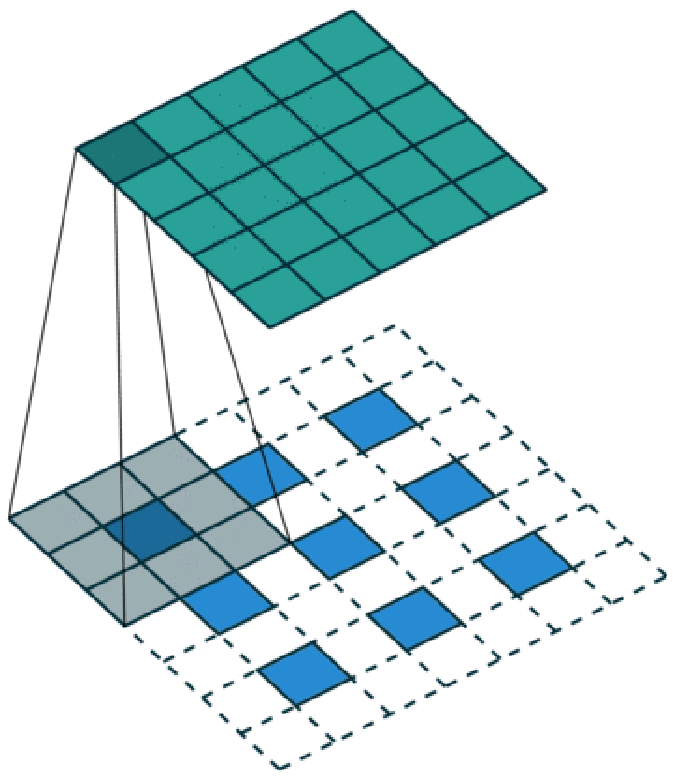
DECONVOLUTION or Fractionally Strided OR Transpose Convolution

We have a kernel of size 3x3. If we convolve on 5x5 we get 3x3. What do we do to get 7x7? Or get a larger channel size than we started with?

A simple approach, but inefficient.



Deconvolution/Transpose Convolution

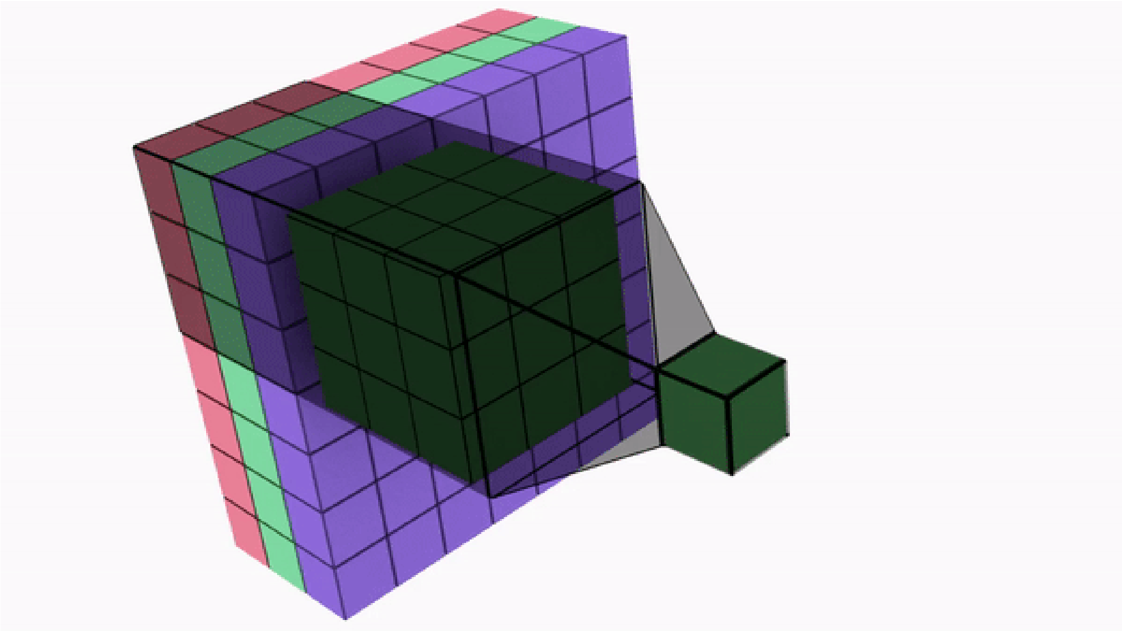


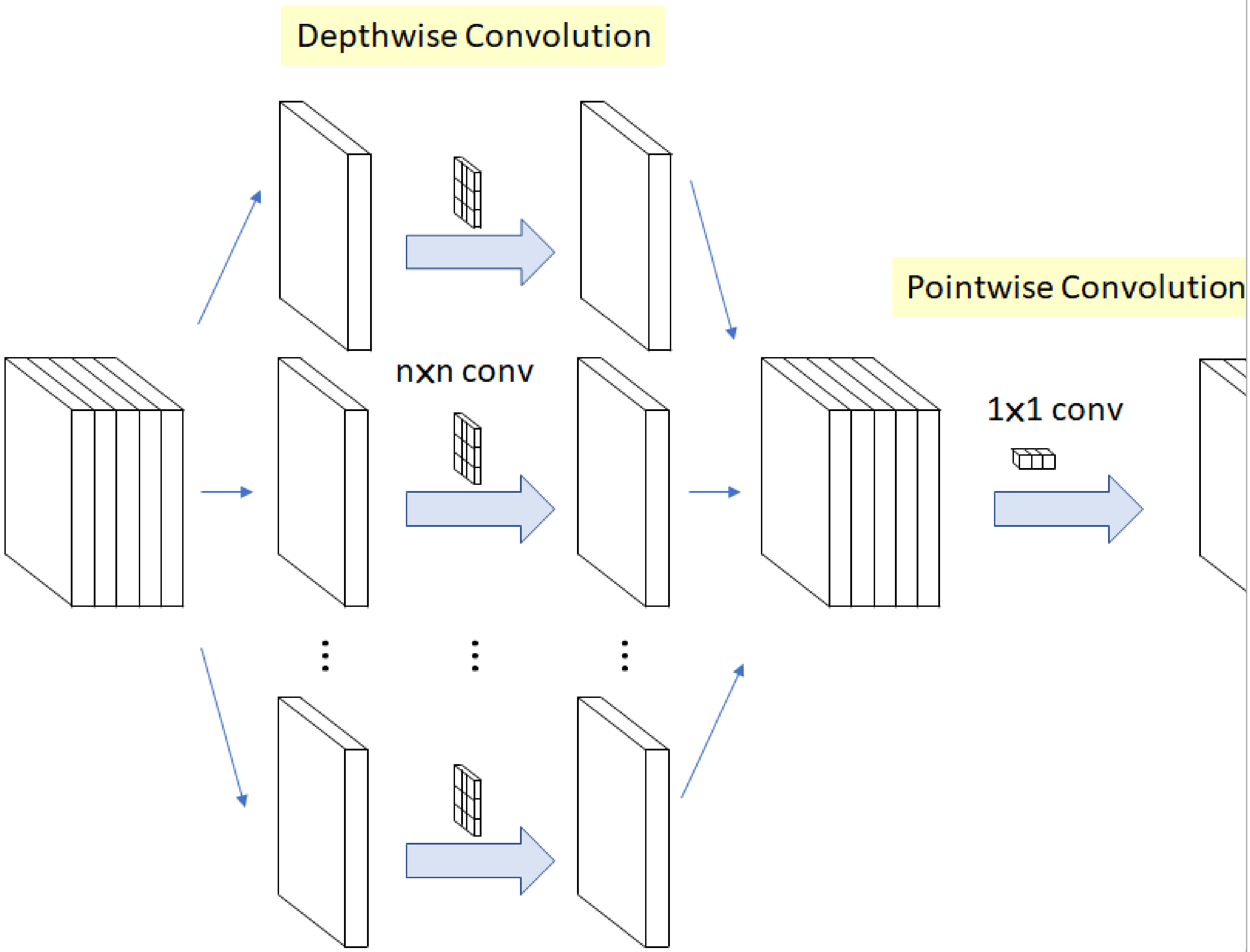
it introduces the checker board issue!

Perceptual Losses for Real-Time Style Transfer and Super-Resolutio

Depthwise Separable Convolution

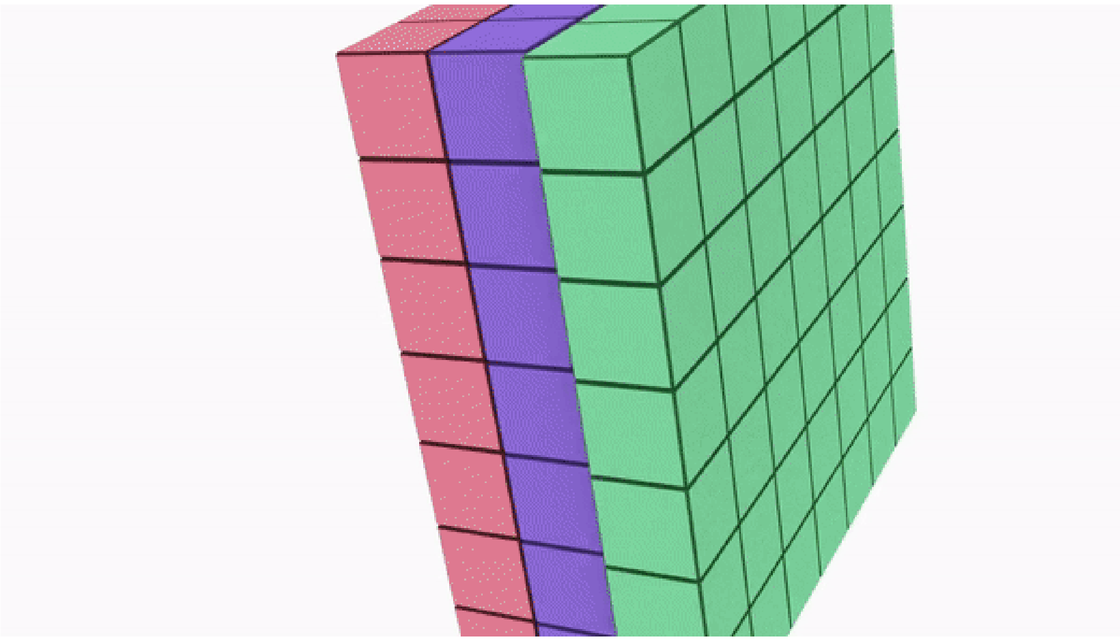
What is wrong in this convolution?





Used by XCEPTION and MOBILENET a lot

Depthwise Convolution



[Reference (https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/)](https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/)

For a depthwise separable convolution on the same example, we traverse the 16 channels with 1 3x3 kernel each, giving us 16 feature maps. Now, before merging anything, we traverse these 16 feature maps with 32 1x1 convolutions each and only then start to them add together. This results in 656 (16x3x3 + 16x32x1x1) parameters opposed to the 4608 (16x32x3x3) parameters from above.

Other convolutions you should learn about:

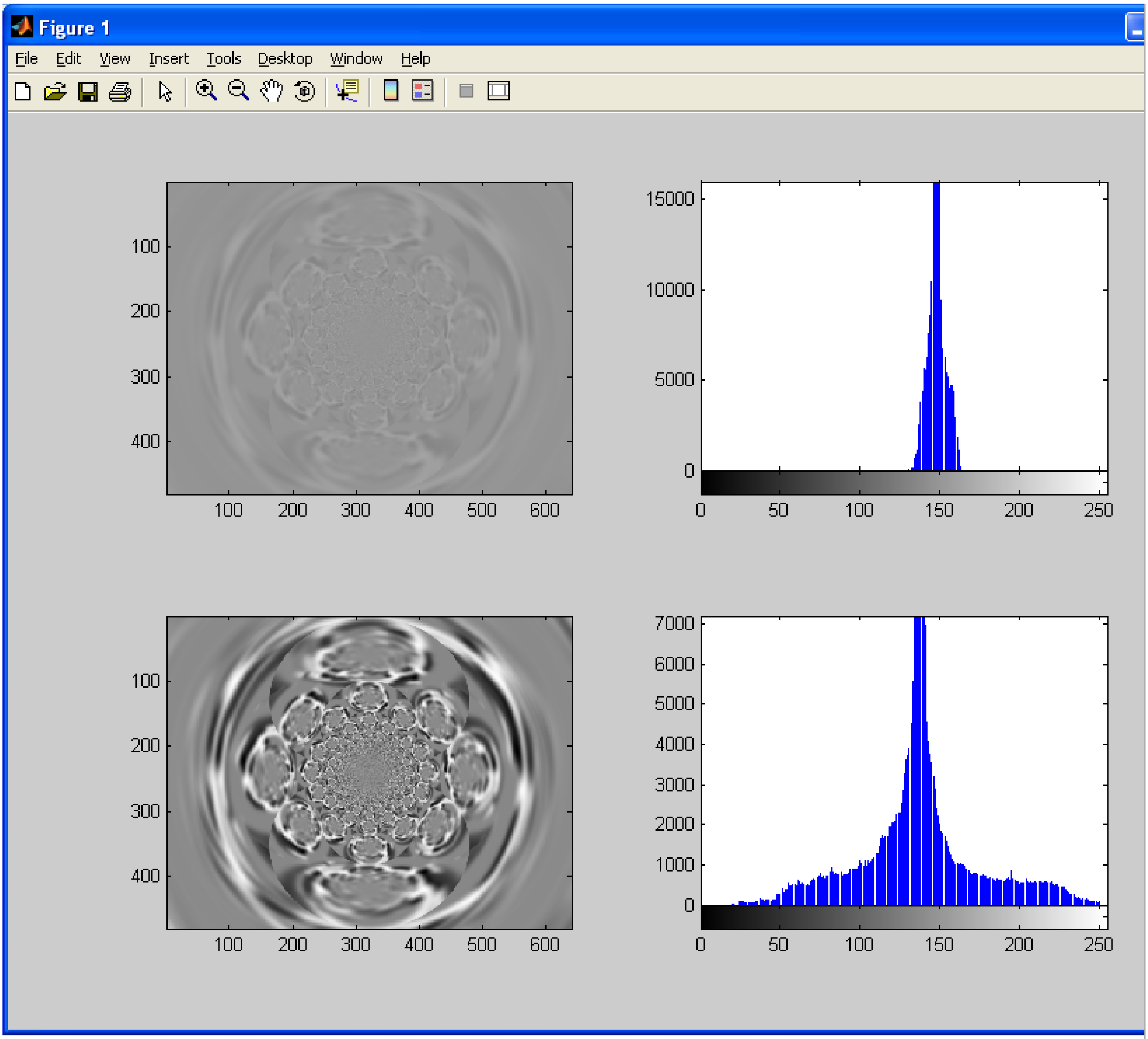
- spacially separable convolution
- grouped convolutions
- roots concept in convolutions

Batch Normalization & Regularization

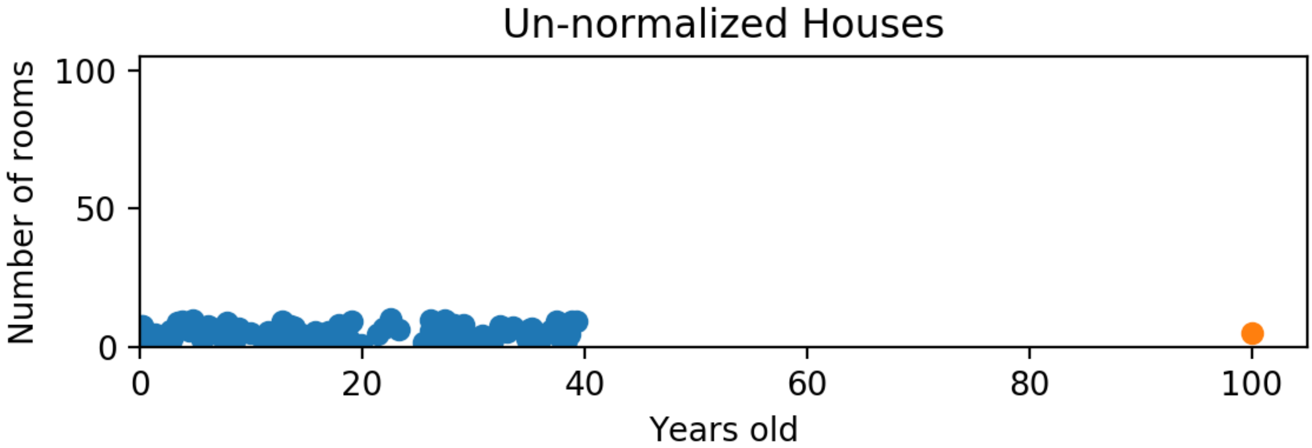
Image Normalization

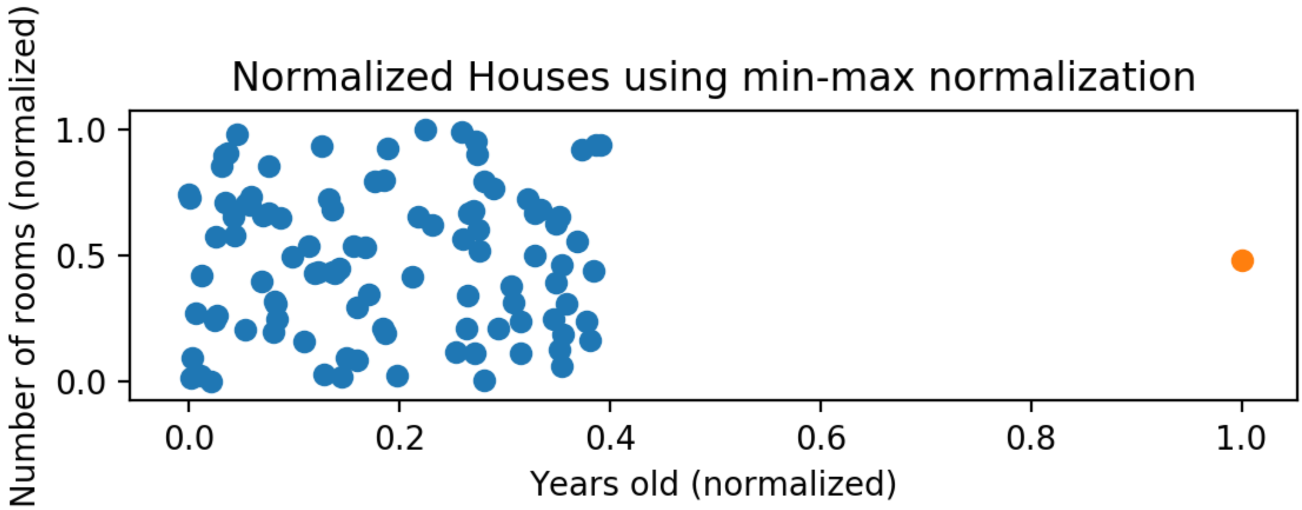
Normalizing input (LeCun et al 1998 “Efficient Backprop”)

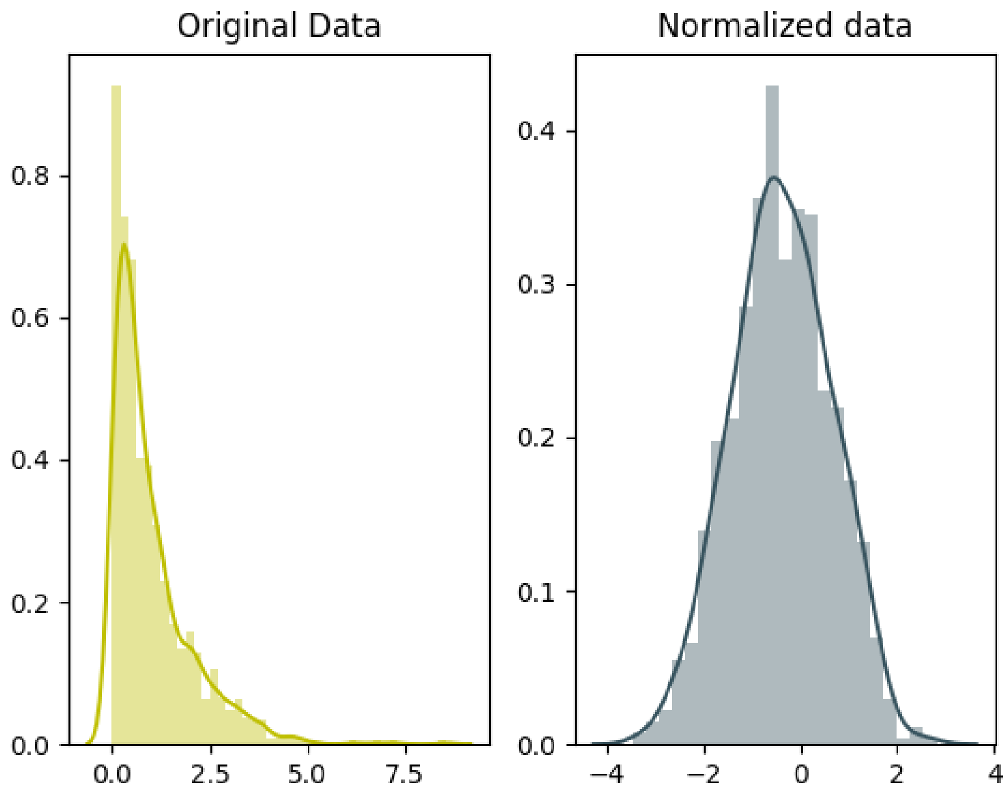
Let's understand through some examples:



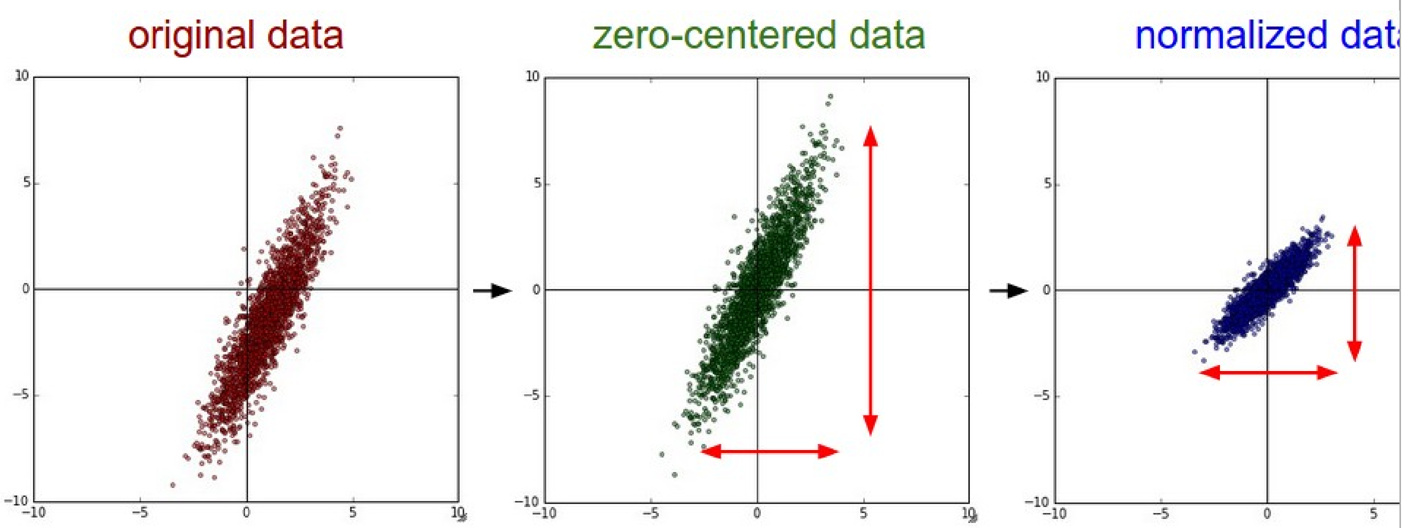
Let's look at normalization working on other kinds of data.


Un-normalized Houses
Number of rooms
100
50
0
0
20
40
60
80
100
Years old


Normalized Houses using min-max normalization
Number of rooms (normalized)
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Years old (normalized)



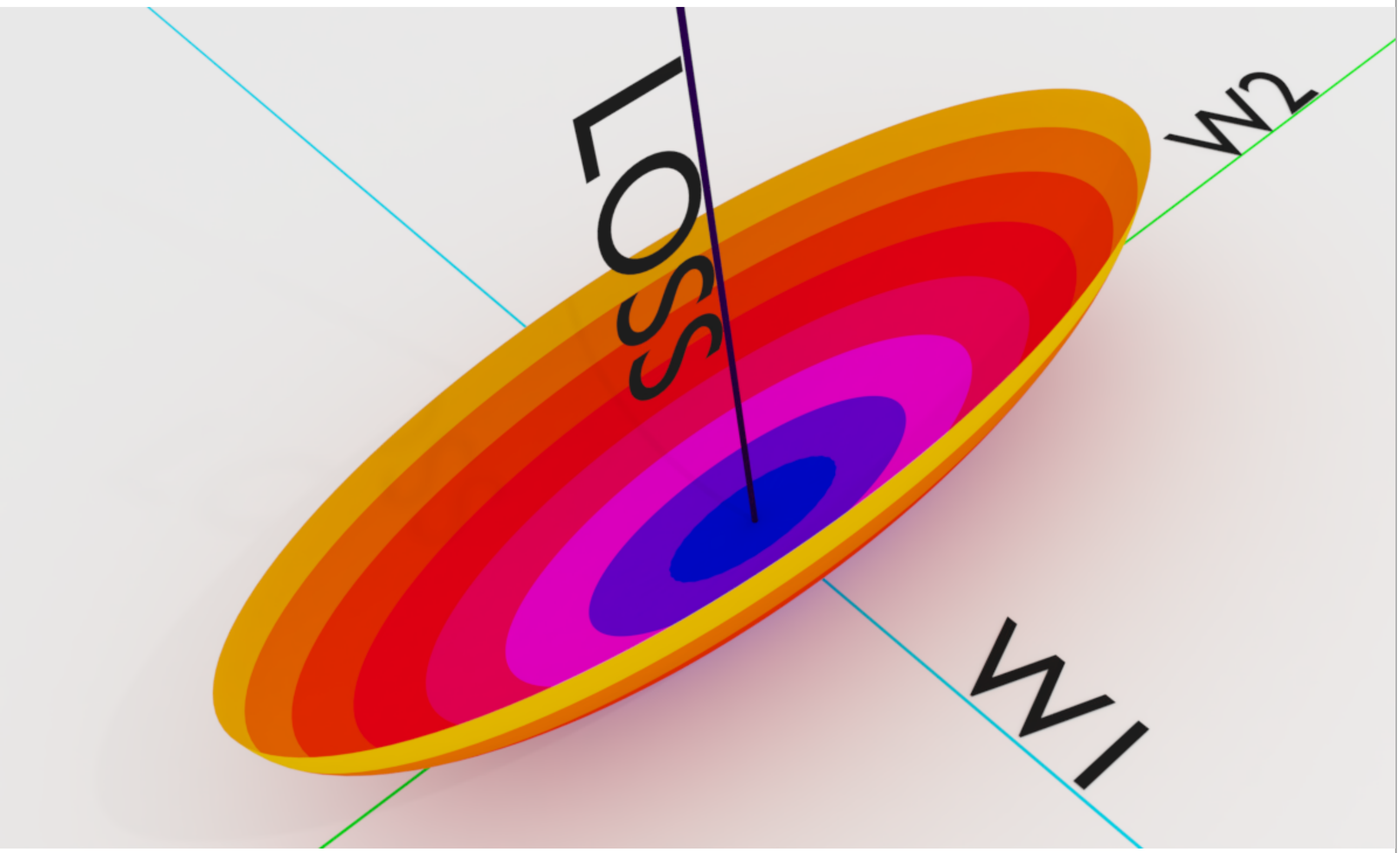
How do we achieve normalization?



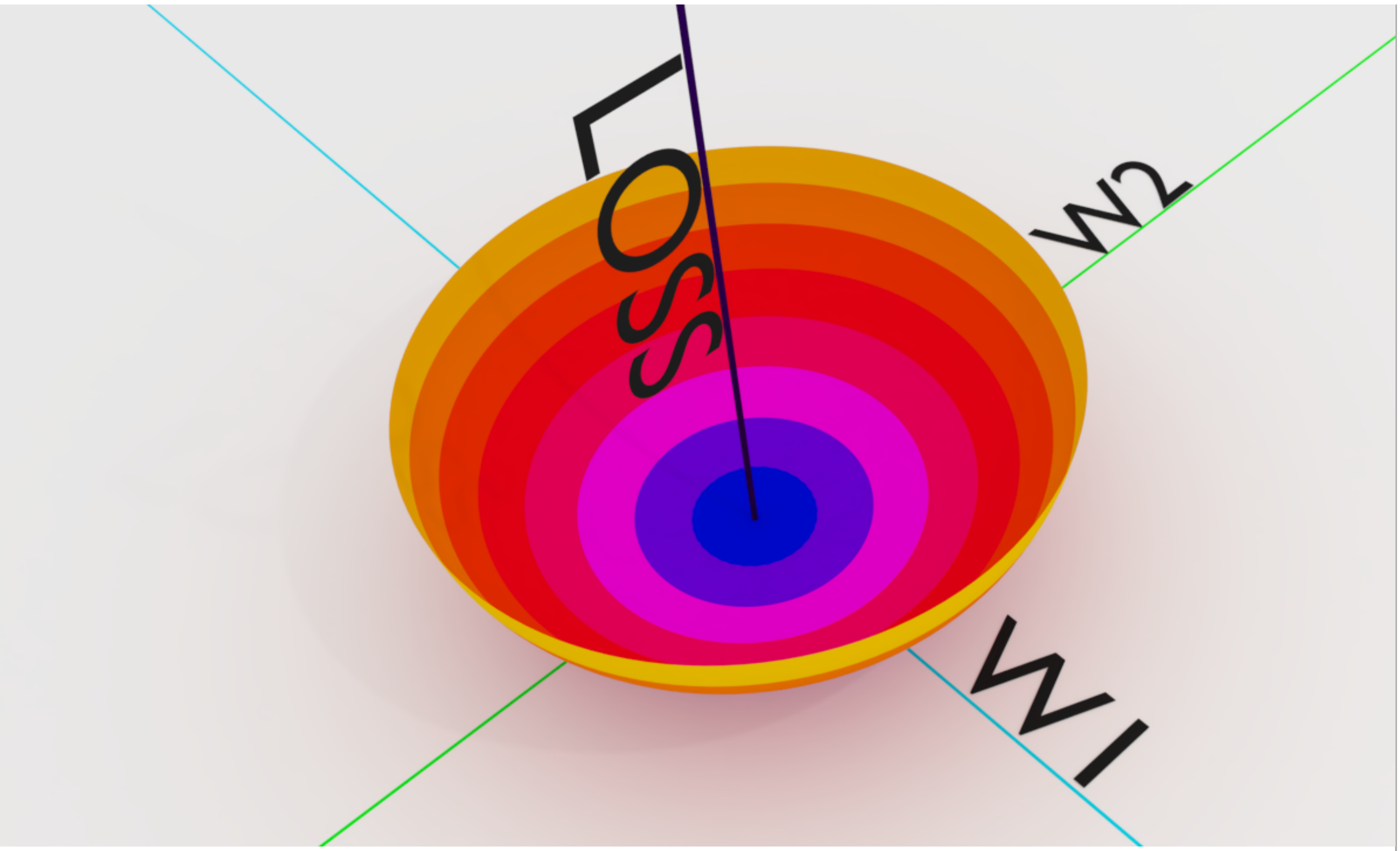
Why limit normalization to images only then?

Channel Normalization -aka Batch Normalization

Let's think about our un-normalized kernels and how loss function would look like:



If we had normalized our kernels (indirectly channels), this is how it would look like:



If features are found at a similar scale, then weights would be in a similar scale, and then backprop would actually make sense, ponder!

[**Batch Normalization**](http://jmlr.org/proceedings/papers/v37/ioffe15.pdf) **(http://jmlr.org/proceedings/papers/v37/ioffe15.pdf)** (BN), introduced in 2015– and is now the defacto standard for all CNNs and RNNs.

Batch Normalization solves a problem called **Internal Covariate shift**. To understand BN we need to understand what is CS.

Covariate means input features. Covariate shift means that the distribution of the features is different in different parts of the training/test data.

Internal Covariate shift refers to changes within the neural network, between layers. *A kernel always giving out higher activation makes next layer kernels always expect this higher activation and so on*.

Imagine what would happen if One channel ranges between -1 to 1 and another between -10 to 10

Very Deep nets can be trained faster and generalize better when the distribution of activations is kept normalized during BackProp.

We regularly see Ultra-Deep ConvNets like Inception, Highway Networks, and ResNet. And giant RNNs for speech recognition, **machine translation** **(https://research.googleblog.com/2016/09/a-neural-network-for-machine.html)**, etc.

Let's look at some math:

Batch Normalization (BN)

$$\rightarrow \boxed{\text{layer}} \rightarrow x \rightarrow \quad \hat{x} = \frac{x - \mu}{\sigma} \quad \rightarrow \quad y = \gamma \hat{x} + \beta$$

- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learr analogous to weights

This is how we implement "batch" normalization:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

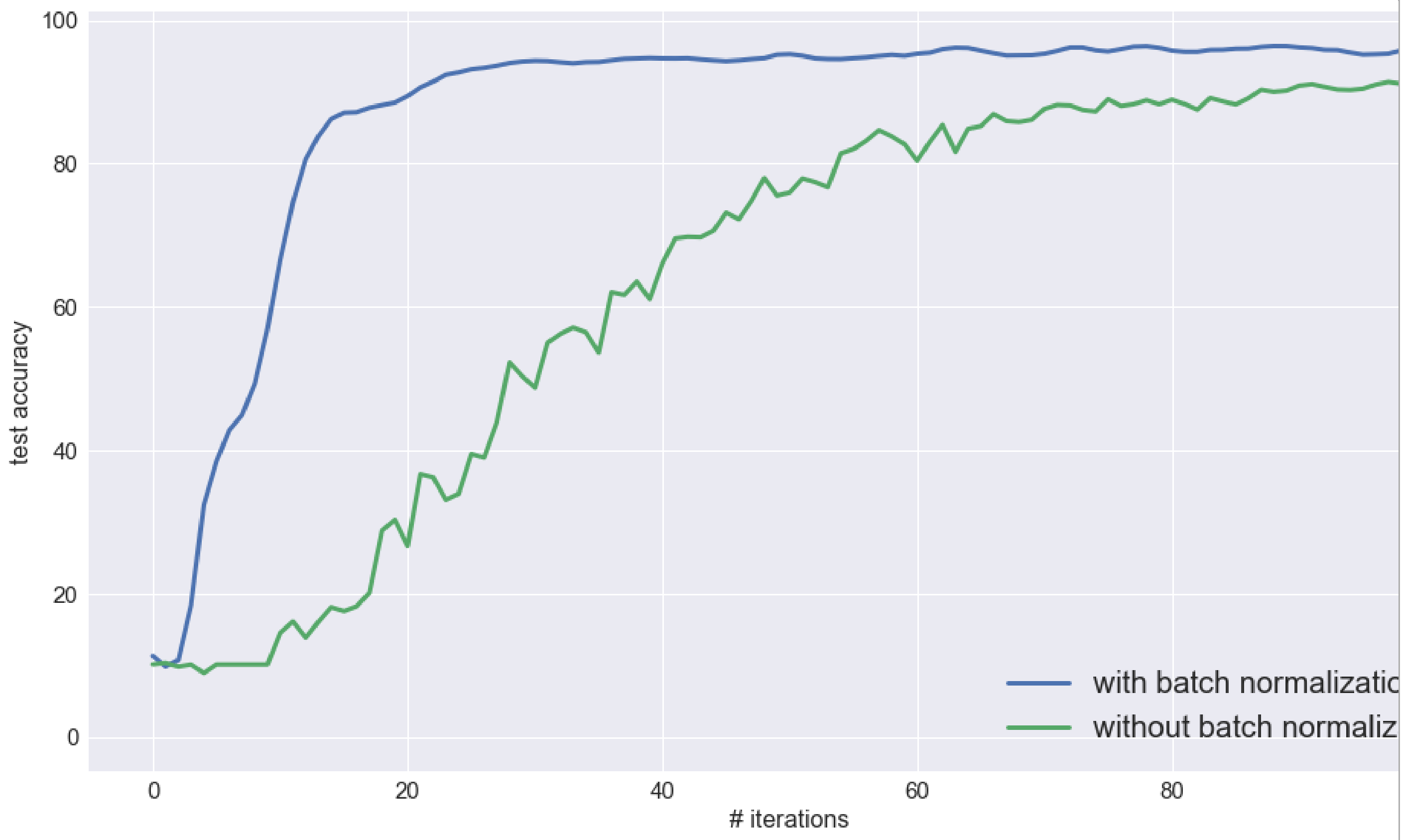
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



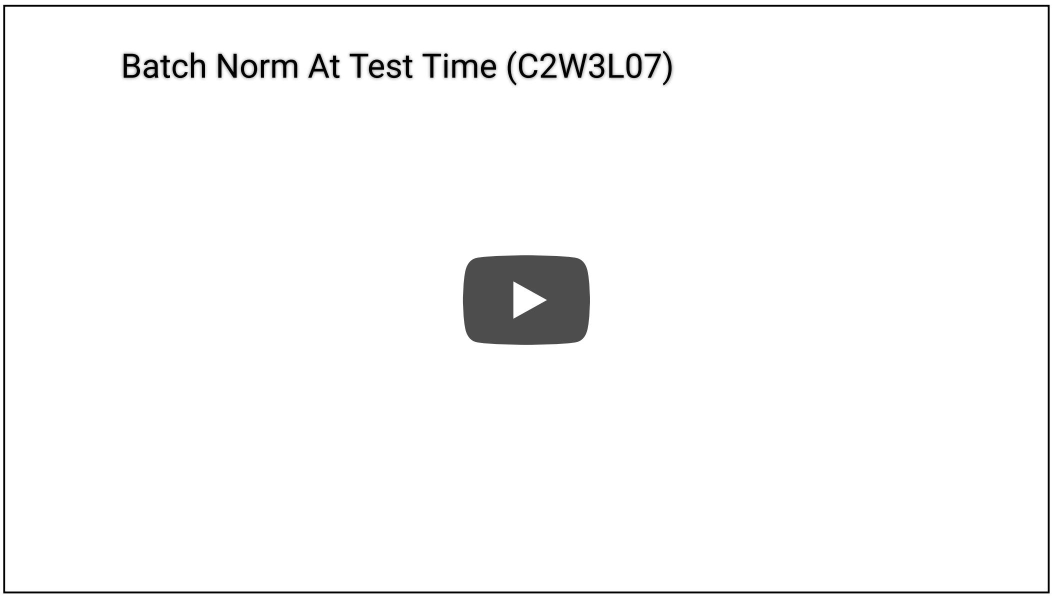
ReLU before BN or BN before ReLU?

Reference Videos:

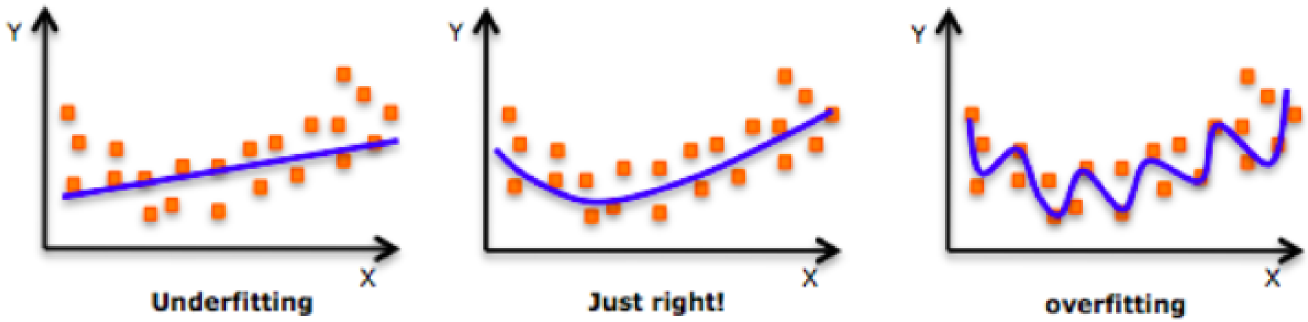
Normalizing Activations in a Network (C2W3L04)

Fitting Batch Norm Into Neural Networks (C2W3L05)

Why Does Batch Norm Work? (C2W3L06)

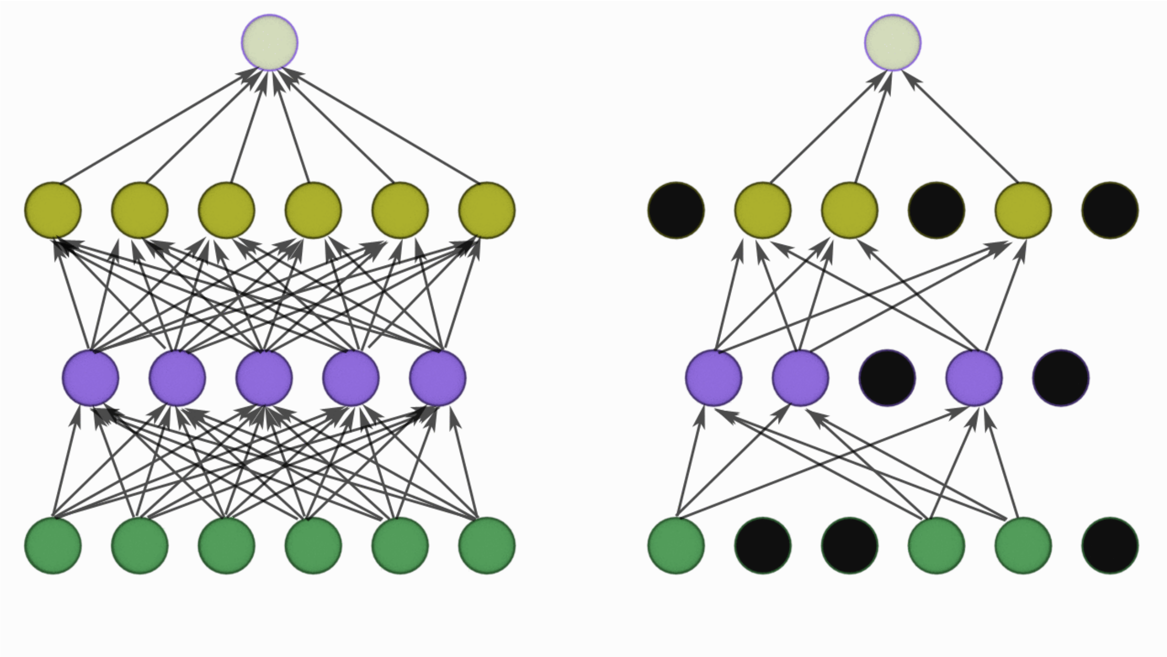


Regularization



Regularization is a key component in preventing overfitting. Also, some techniques of regularization can be used to reduce model parameters while maintaining accuracy, for example, to drive some of the parameters to zero. This might be desirable for reducing the model size or driving down the cost of evaluation in a mobile environment where processor power is constrained.

Drop Out



Most common techniques of regularization used nowadays in the industry are:

- dataset augmentation
- dropout
- weight decay

Recommended:

Receptive Fields

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

1 = (3 + 2*0 - 3)/1 + 1
3 = (3 + 2*1 - 3)/1 + 1
3 = (5 + 2*1- 3) /2 + 1

ONLINE CALCULATOR (https://fomoro.com/research/article/receptive-field-calculator)

Receptive Field

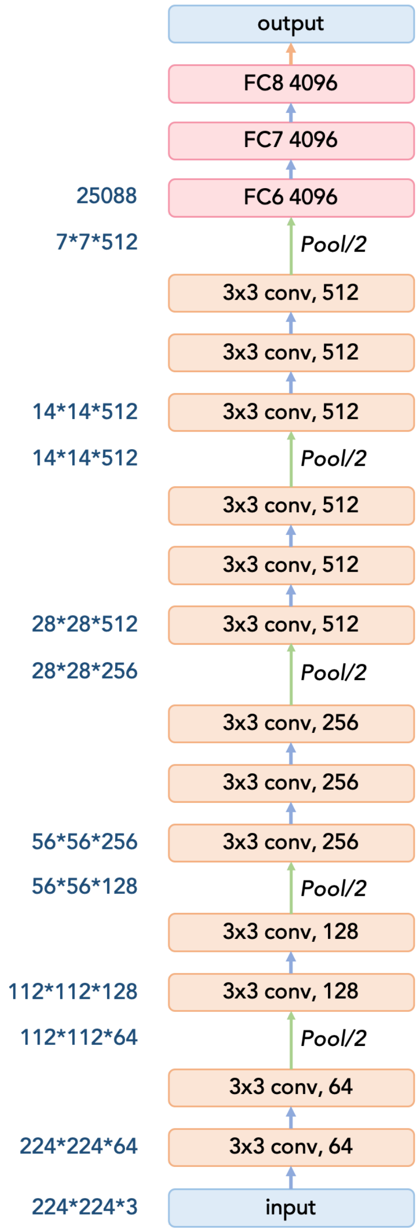
$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$
$$j_{out} = j_{in} * s$$
$$r_{out} = r_{in} + (k - 1) * j_{in}$$
$$start_{out} = start_{in} + \left(\frac{\mathrm{k} - 1}{2} - \mathrm{p}\right) * j_{in}$$

j is the jump, **r** is the receptive field (ignore start for now)

RF1 = 1 + (3-1)*1 = 3
RF2 = 1 + (3-1)*(1*1) = 3 (padding has no effect)
RF3 = 1 + (3-1)*(1*2) = 4 (stride has effect!)

The network is 10x10 | 8x8 | 6x6. Our RF currently is 5x5. IF we apply MaxPooling then

First Formula: n_out = (6 + 2*0 -2)/2 = 3
Second Formula: RF = **5** + (2-1)***1** = **6**
Add 3x3 again
First Formula: (3 + 2*0 - 3) + 1 = 1
Second Formula: RF= **6** + (3-1)***2** = **10**



(k, s)	(k, s)'	receptive field
7 x 1	404 x 32	404 x 404
2 x 2	212 x 32	212 x 212
3 x 1	196 x 16	196 x 196
3 x 1	164 x 16	164 x 164
3 x 1	132 x 16	132 x 132
2 x 2	100 x 16	100 x 100
3 x 1	82 x 8	92 x 92
3 x 1	76 x 8	76 x 76
3 x 1	60 x 8	60 x 60
2 x 2	44 x 8	44 x 44
3 x 1	40 x 4	40 x 40
3 x 1	32 x 4	32 x 32
3 x 1	24 x 4	24 x 24
2 x 2	16 x 4	16 x 16
3 x 1	14 x 2	14 x 14
3 x 1	10 x 2	10 x 10
2 x 2	6 x 2	6 x 6
3 x 1	5 x 1	5 x 5
3 x 1	3 x 1	3 x 3

Source (http://zike.io/posts/calculate-receptive-field-for-vgg-16/)

Assignment 3:

1. Run **this** (https://colab.research.google.com/drive/1STOg33u7haqSptyjUL40FZIxNW4XdBQK) network (base network) for 50 epochs, report Validation Accuracy **after** 50 epochs.
2. Add new cells at the bottom of the code, and write your own network such that:
   1. it uses depthwise separable convolution **ONLY** (no Conv2D)
   2. it uses BatchNormalization
   3. has less than 100,000 parameters
   4. it uses proper dropout values
   5. you've mentioned the output size for each layer
   6. you've mentioned the receptive field for each layer
   7. runs for 50 epochs
   8. beats the validation score within 50 epochs (at any epoch run, doesn't need to be final one)

3. Submit the github link
4. Your github links must have:
    1. Assignment 3.ipynb file with your code as well as logs for both the models
    2. ReadMe.md which should have:
        1. Final Validation accuracy for Base Network
        2. Your model definition (model.add...) with output channel size and receptive field
        3. Your 50 epoch logs

Points 2,000

Submitting a website url

Due	For	Available from	Until
Nov 29 at 5:30am	F6	Nov 22 at 6:30am	Nov 29 at 5:30am
Nov 30 at 11:30am	S12	Nov 23 at 12:30pm	Nov 30 at 11:30am
Dec 2 at 5:30am	M6	Nov 25 at 6:30am	Dec 2 at 5:30am
Dec 4 at 5:30am	W6	Nov 27 at 6:30am	Dec 4 at 5:30am

+ Rubric