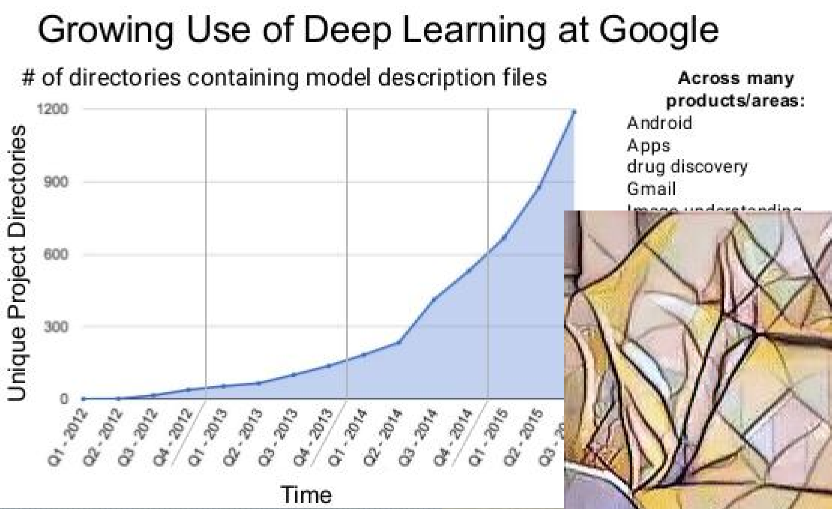
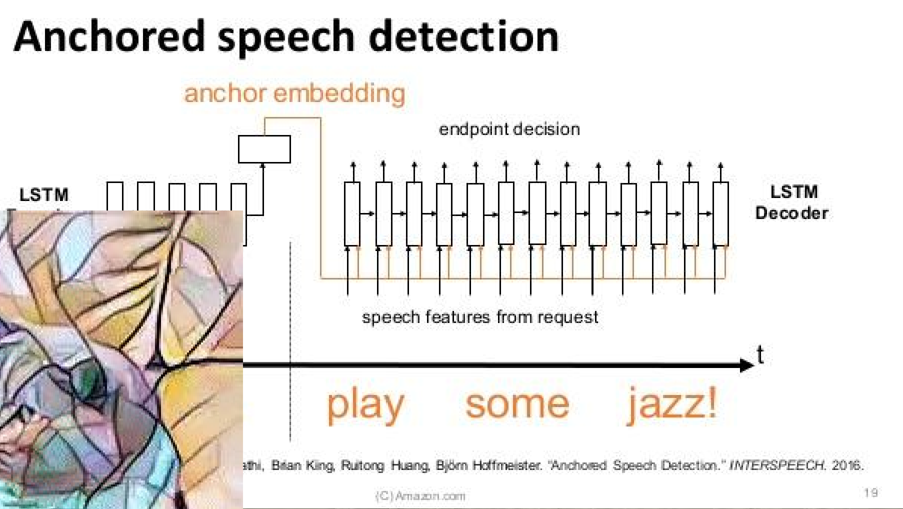
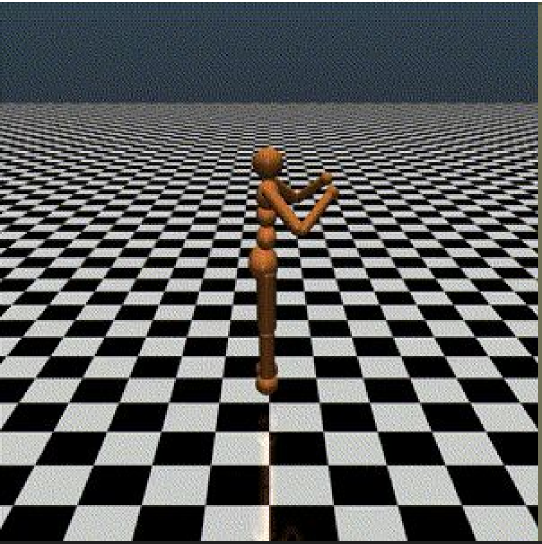
Deep Learning (CNNs) Jumpstart 2018

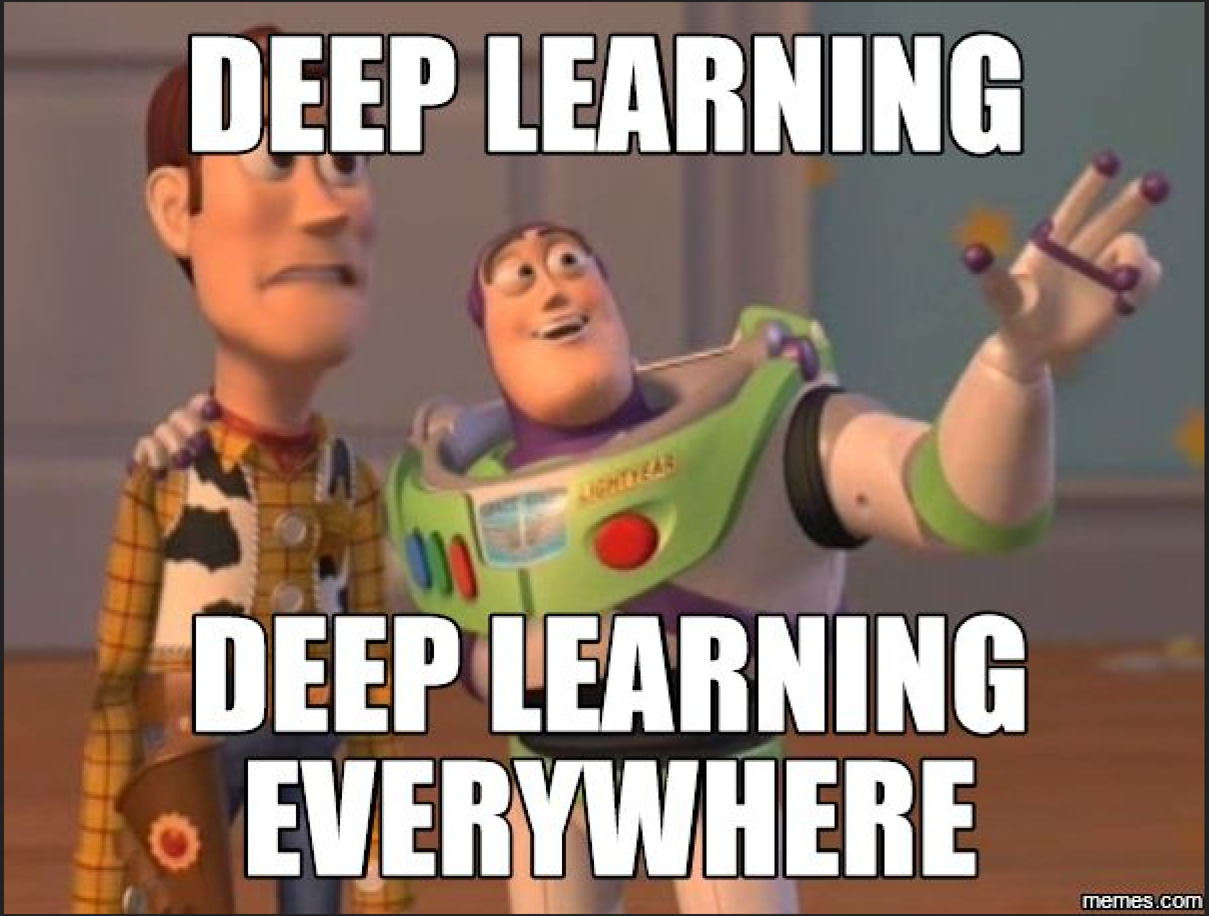
Chaoqi Wang, Amlan Kar

Why study it?


Growing Use of Deep Learning at Google
# of directories containing model description files
Unique Project Directories
1200
900
600
300
0
Q1 - 2012
Q2 - 2012
Q3 - 2012
Q4 - 2012
Q1 - 2013
Q2 - 2013
Q3 - 2013
Q4 - 2013
Q1 - 2014
Q2 - 2014
Q3 - 2014
Q4 - 2014
Q1 - 2015
Q2 - 2015
Time
Across many products/areas:
Android
Apps
drug discovery
Gmail


Anchored speech detection
anchor embedding
endpoint decision
LSTM
LSTM Decoder
speech features from request
t
play
some
jazz!
Brian King, Ruitong Huang, Björn Hoffmeister. "Anchored Speech Detection." INTERSPEECH, 2016.
(C) Amazon.com
19




DEEP LEARNING
DEEP LEARNING
EVERYWHERE
memes.com

To the basics and beyond!

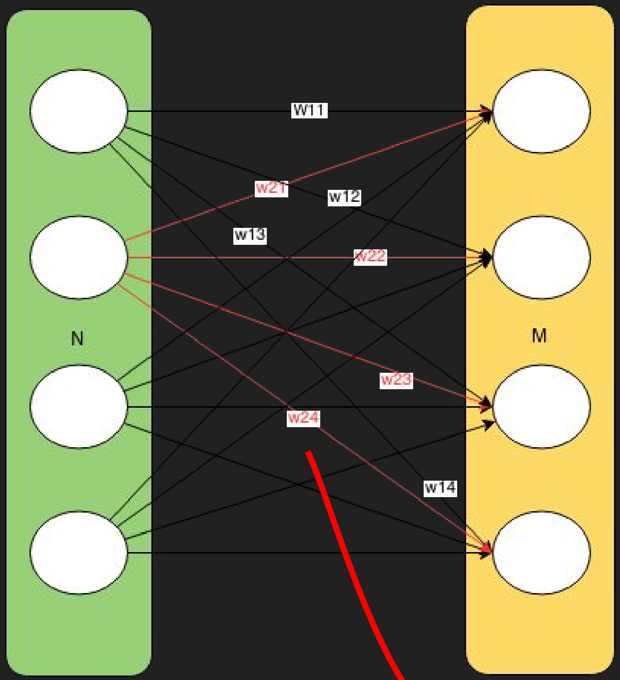
Note: Buzz will point to recommended resources while we fly through at light speed

Building Blocks

We always work with features (represented by real numbers)
Each block transforms features to newer features
Blocks are designed to exploit implicit regularities

Fully Connected Layer

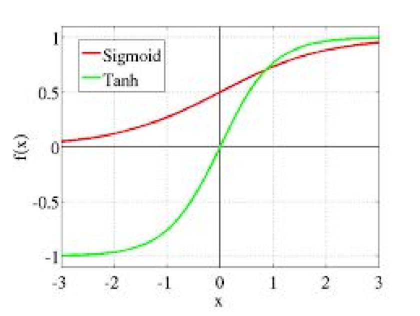
Use all features to compute a new set of features



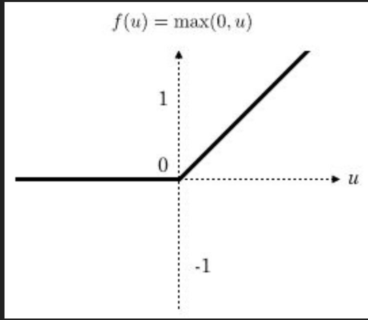
Linear Transformation - $F_2 = W^T F_1 + b$

Non-Linearity

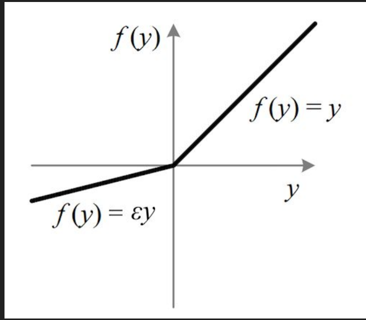
Apply a nonlinear function to features



Sigmoid (Logistic Function)



ReLU (Rectified Linear)



Leaky ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

Exponential Linear (eLU)

More:

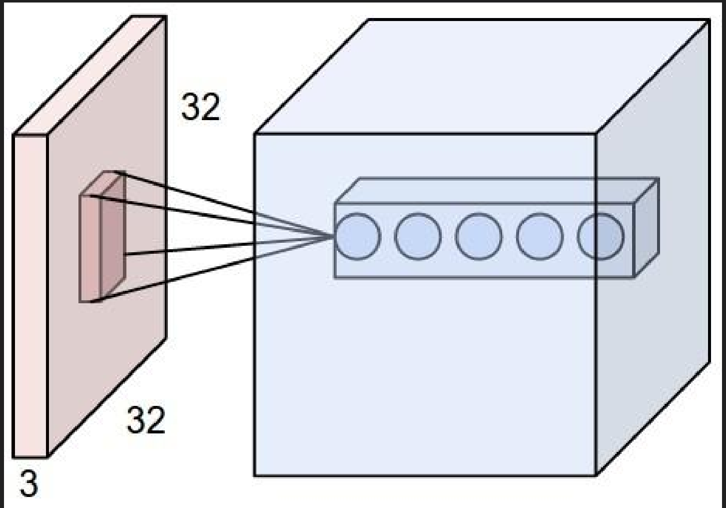
- Maxout
- SeLU
- Swish
- And so many more …

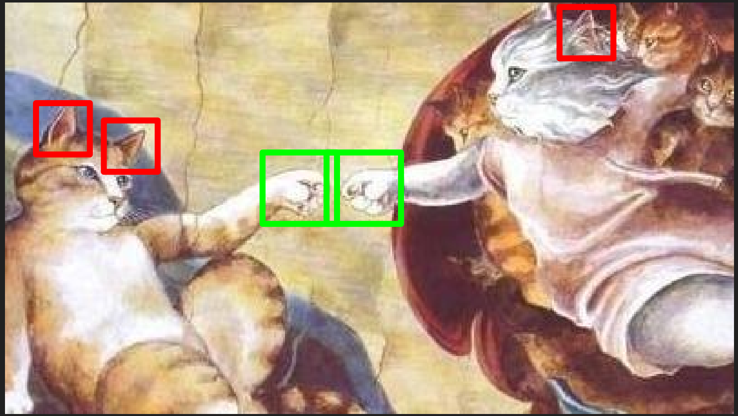
Comprehensive guide to nonlinearities:

https://towardsdatascience.com/secret-sauce-behind-the-beauty-of-deep-learning-beginners-guide-to-activation-functions-a8e23a57d046

Convolutional Layer

Use a small window of features to compute a new set of features



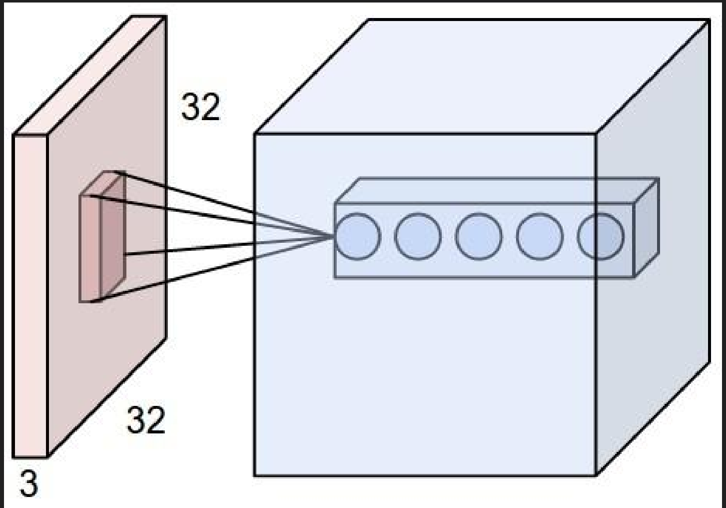


Need different parameters?

Comprehensive guide to convolutional layers:
http://cs231n.github.io/convolutional-networks/

Convolutional Layer

Use a small window of features to compute a new set of features

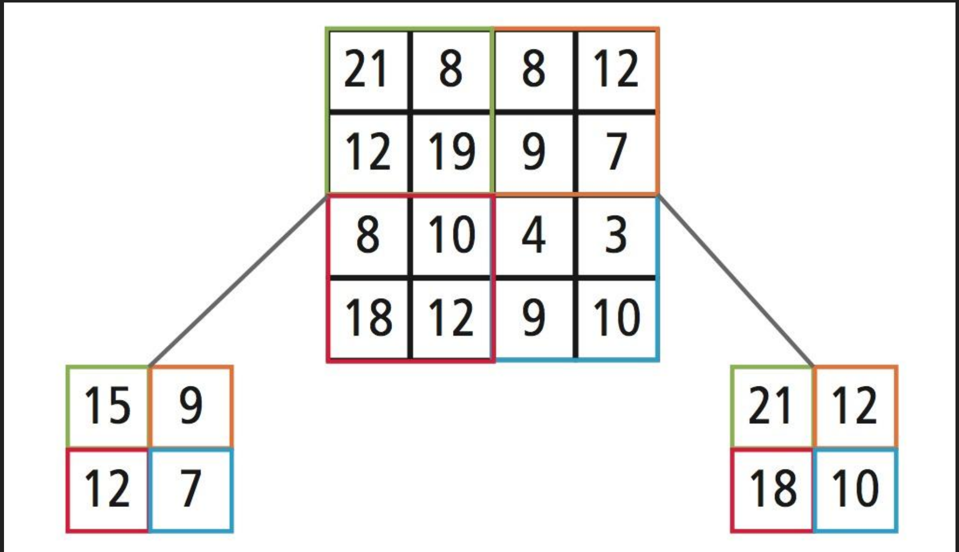


- Lesser parameters than a FC layer
- Exploits the fact that local features repeat across images
- Exploiting implicit order can be seen as a form of model regularization

Normal convolution layers look at information in fixed windows. Deformable ConvNets and Non Local Networks propose methods to alleviate this issue

Pooling

Aggregate features to form lower dimensional features



Average Pooling

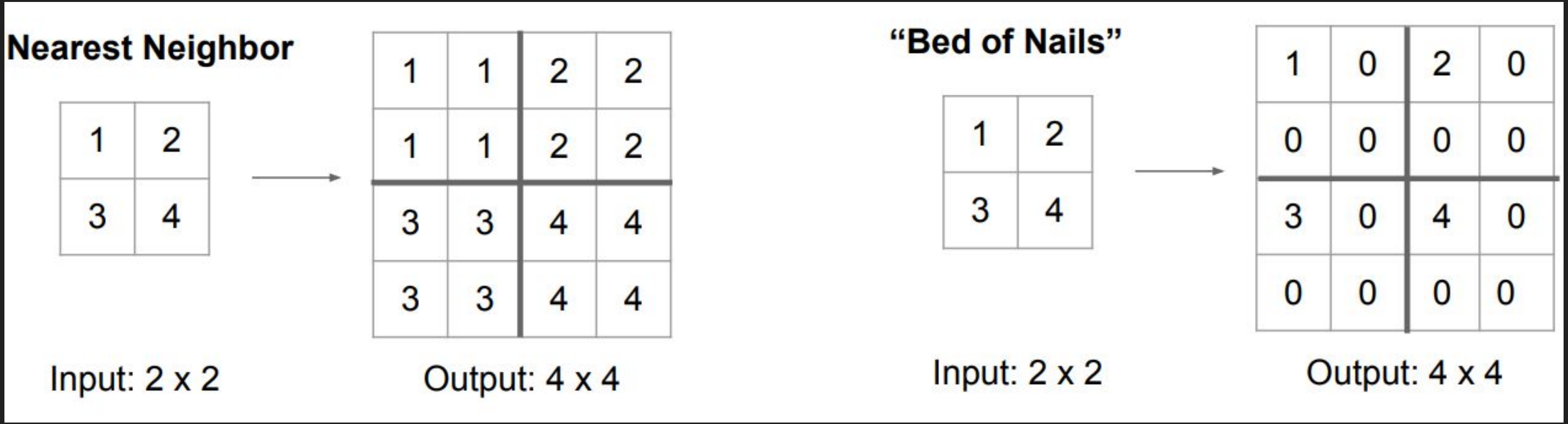
Max Pooling

- Reduce dimensionality of features
- Robustness to tiny shifts

Also see Global Average Pooling (used in the recent best performing architectures)

Upsampling Layers

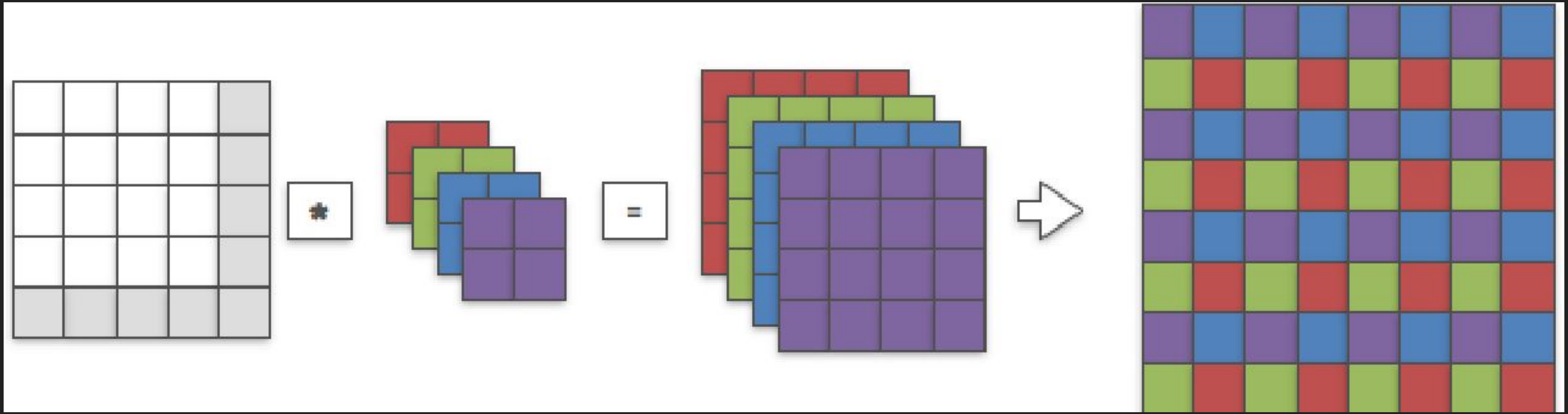
How to generate more features from less?



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Upsampling Layers: Subpixel Convolution

Produce a grid of nxn features as n^2 filters in a convolution layer

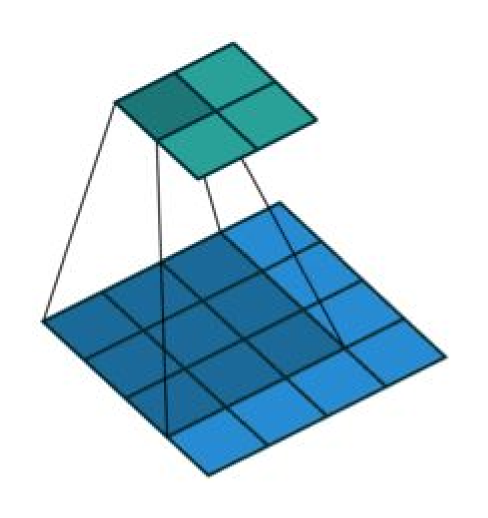


https://arxiv.org/pdf/1609.05158.pdf

Also read about checkerboard artifacts here:
https://distill.pub/2016/deconv-checkerboard/

Upsampling Layers: Transpose Convolution

What features did my current features come from?



Convolution

$$\begin{pmatrix}
w_{0,0} & 0 & 0 & 0 \\
w_{0,1} & w_{0,0} & 0 & 0 \\
w_{0,2} & w_{0,1} & 0 & 0 \\
0 & w_{0,2} & 0 & 0 \\
w_{1,0} & 0 & w_{0,0} & 0 \\
w_{1,1} & w_{1,0} & w_{0,1} & w_{0,0} \\
w_{1,2} & w_{1,1} & w_{0,2} & w_{0,1} \\
0 & w_{1,2} & 0 & w_{0,2} \\
w_{2,0} & 0 & w_{1,0} & 0 \\
w_{2,1} & w_{2,0} & w_{1,1} & w_{1,0} \\
w_{2,2} & w_{2,1} & w_{1,2} & w_{1,1} \\
0 & w_{2,2} & 0 & w_{1,2} \\
0 & 0 & w_{2,0} & 0 \\
0 & 0 & w_{2,1} & w_{2,0} \\
0 & 0 & w_{2,2} & w_{2,1} \\
0 & 0 & 0 & w_{2,2}
\end{pmatrix}^T$$

Matrix Multiplication

- Convolutions are sparse matrix multiplications
- Multiplying the transpose of this matrix to the 4 dimensional input gives a 16 dimensional vector
- This is also how backpropagation (used to train networks) works for conv layers!

Do read: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html#transposed-convolution-arithmetic

Learning

Loss Functions
Backpropagation

Loss Functions

What should our training algorithm optimize? (some common ones)

Classification -> Cross Entropy between predicted distribution over classes and ground truth distribution
Regression -> L2 Loss, L1 Loss, Huber (smooth-L1) Loss
Decision Making (mainly in Reinforcement Learning)-> Expected sum of reward (very often non-differentiable, use many tricks to compute gradients)

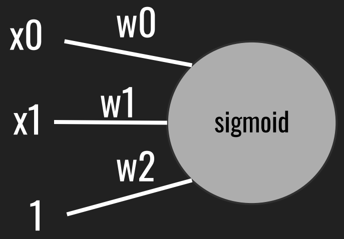
- Most other tasks have very carefully selected domain specific loss functions and it is one of the most important make it or break it for a network

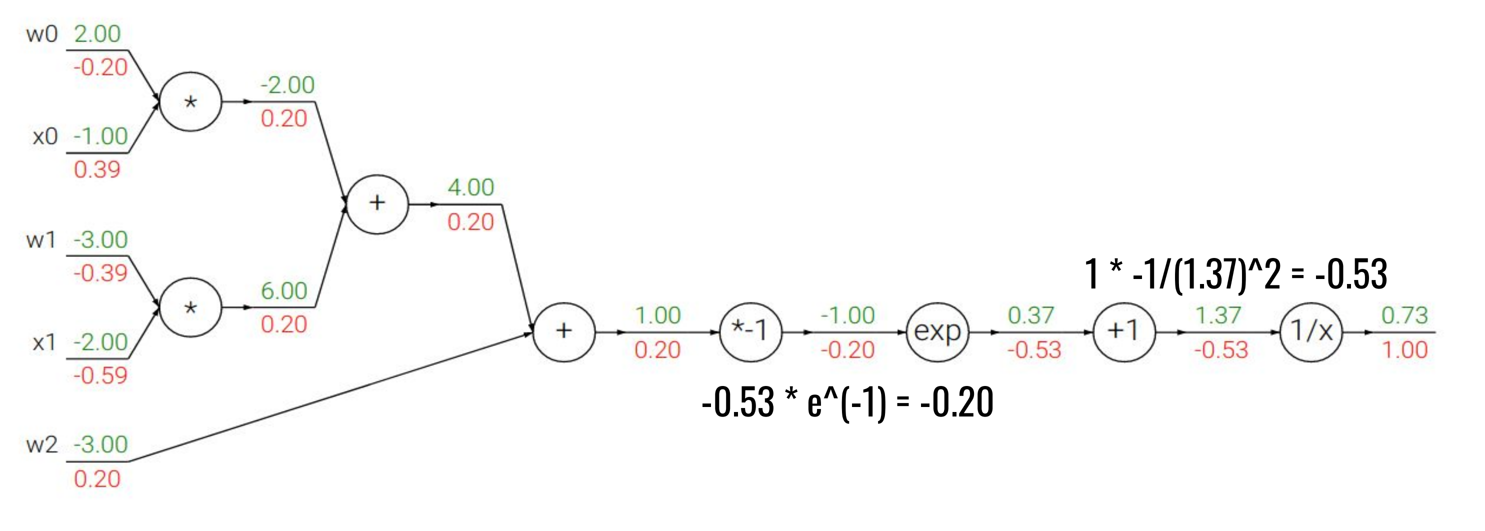
How do we optimize?
We use different variants of stochastic gradient descent: $w^t = w^{t-1} + a \nabla w$

http://www.deeplearningbook.org/contents/optimization.html - See for more on optimization

Backpropagation

Chain Rule!





http://cs231n.github.io/optimization-2/

Task

Do it yourself!

- Derive the gradients w.r.t. the input and weights for a single fully connected layer
- Derive the same for a convolutional layer

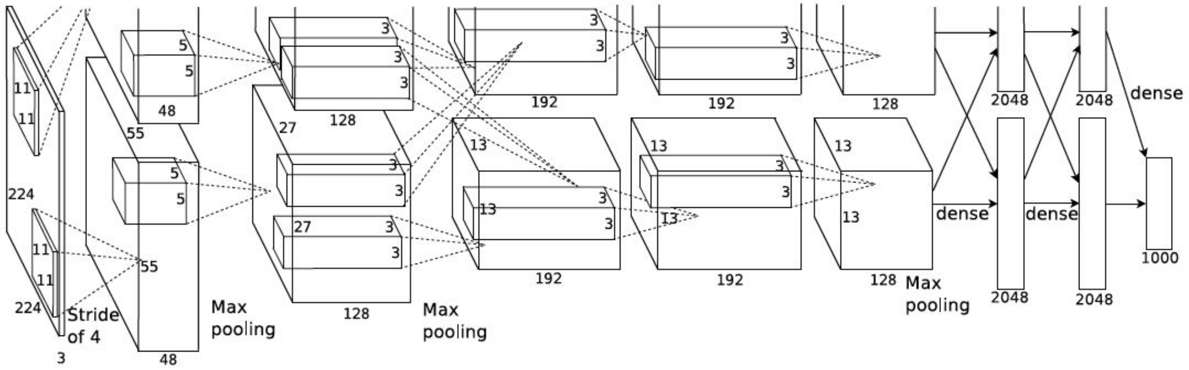
- Assume that the gradient from the layers above is known and calculate the gradients w.r.t. the weights and activations of this layer. You can do it for any non linearity

In case you're lazy or you want to check your answer:
FC - https://medium.com/@erikhallstrm/backpropagation-from-the-beginning-77356edf427d
Conv - https://grzegorzgwardys.wordpress.com/2016/04/22/8/

Next Up: A Tour of Star Command's latest and greatest weapons!

Case Study 1: AlexNet-2012

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



~60M parameters

5 Convolutional layers
3 Max pooling layers
2 LRN(Local Response Normalization) layers, (not common anymore)
3 Fully connected layers

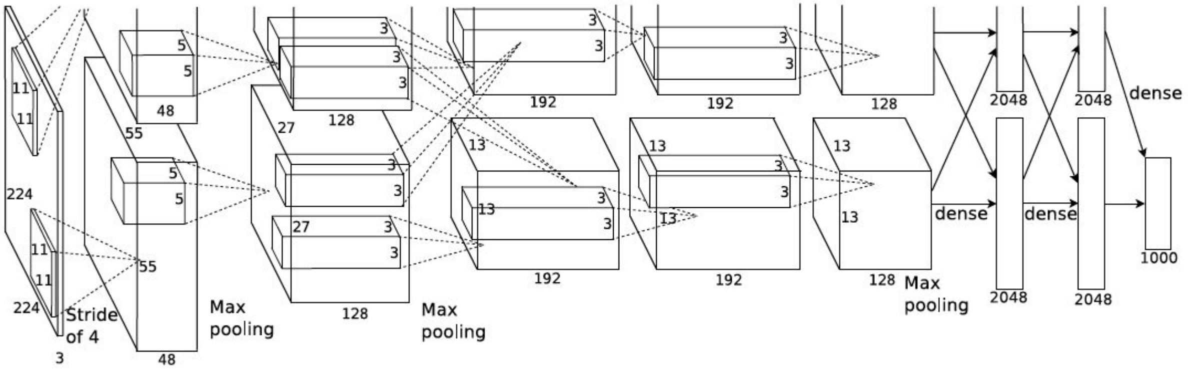
$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study 1: AlexNet-2012

Architecture:

CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



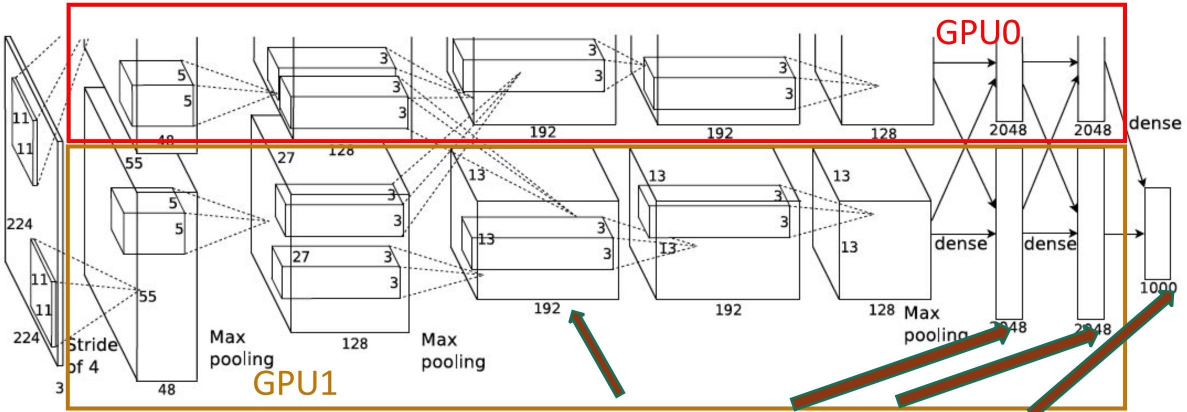
Details:

1. Using ReLU for non-linearity
2. Using dropout(0.5), data augmentation, L2 weight decay(5e-4)

Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study 1: AlexNet-2012

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



CONV3, FC6, FC7, FC8: Connections with all feature maps in preceding layer, communication across GPUs

1. Using ReLU for non-linearity
2. Using dropout(0.5), data augmentation, L2 weight decay(5e-4)
3. Multi-GPU (2 GTX 580 GPUs)
4. SGD Momentum 0.9, batch size 128
5. LR reduced by 10 when val acc plateaus

Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study 1: AlexNet-2012

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

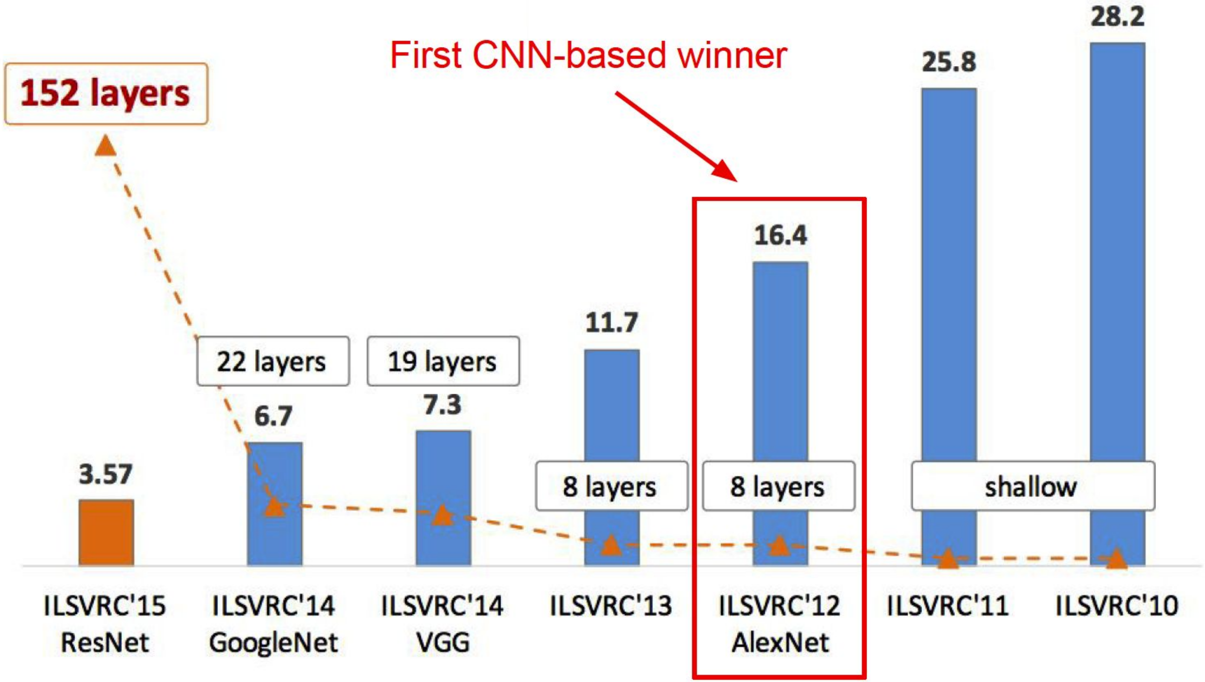


Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Figure copyright Kaiming He, 2016. Reproduced with permission.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

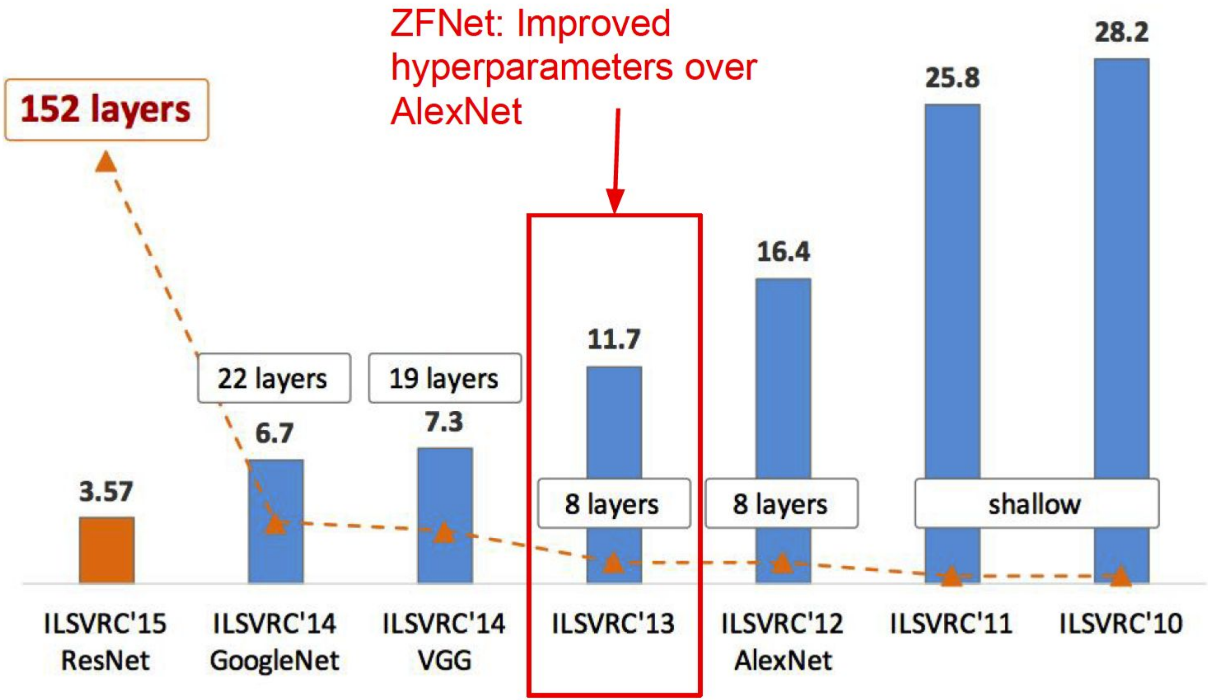
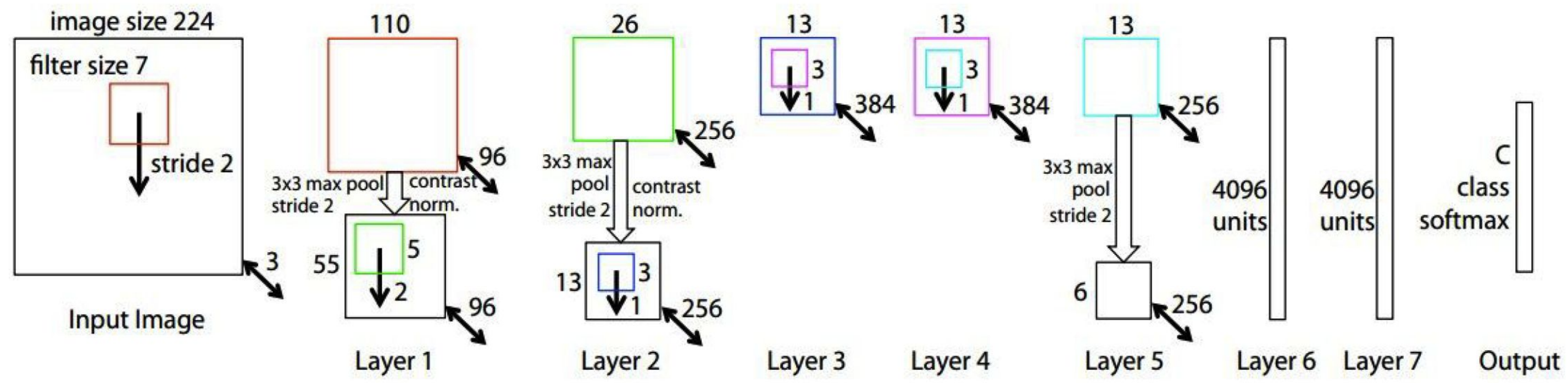


Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Figure copyright Kaiming He, 2016. Reproduced with permission.

ZFNet

[Zeiler and Fergus, 2013]



AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

Image From cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

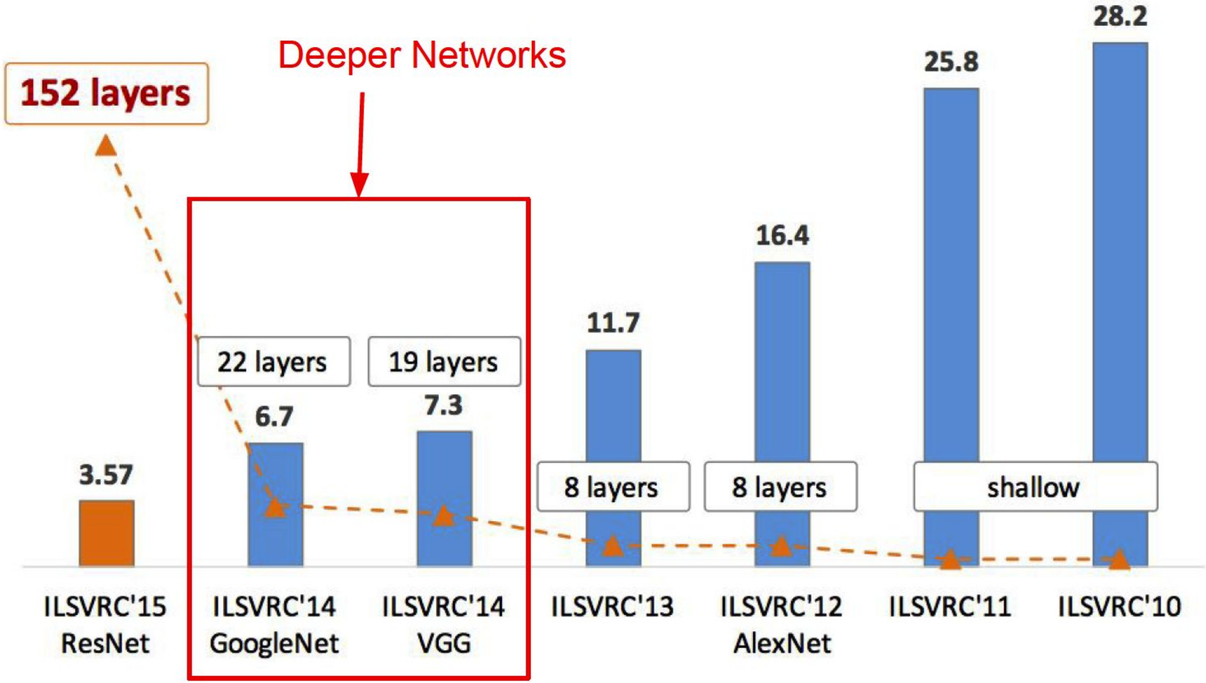


Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Figure copyright Kaiming He, 2016. Reproduced with permission.

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Details:

- ILSVRC’14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks

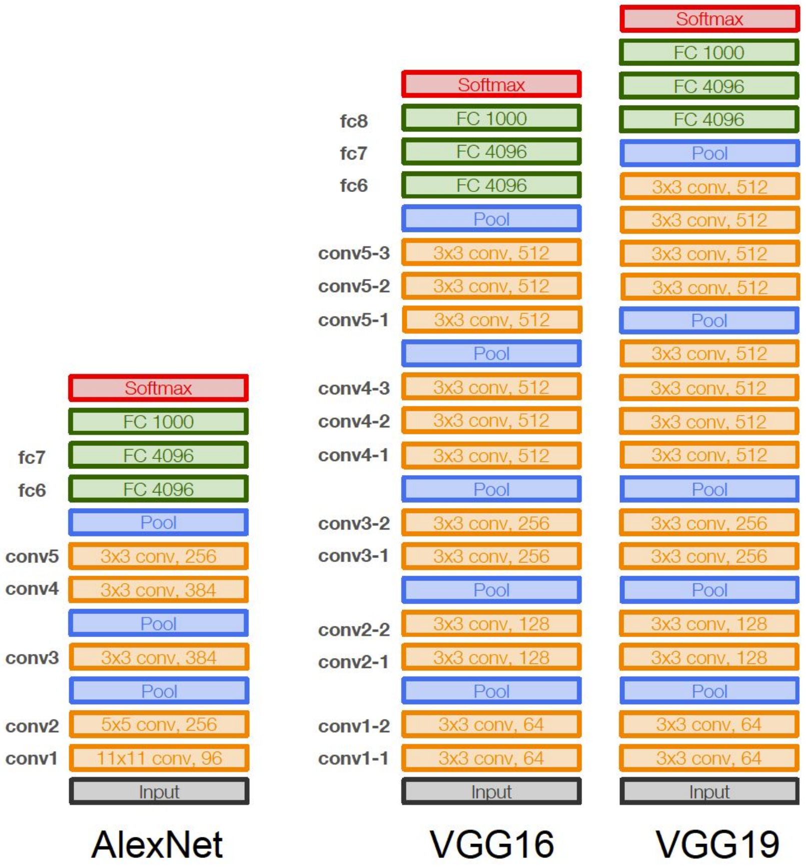


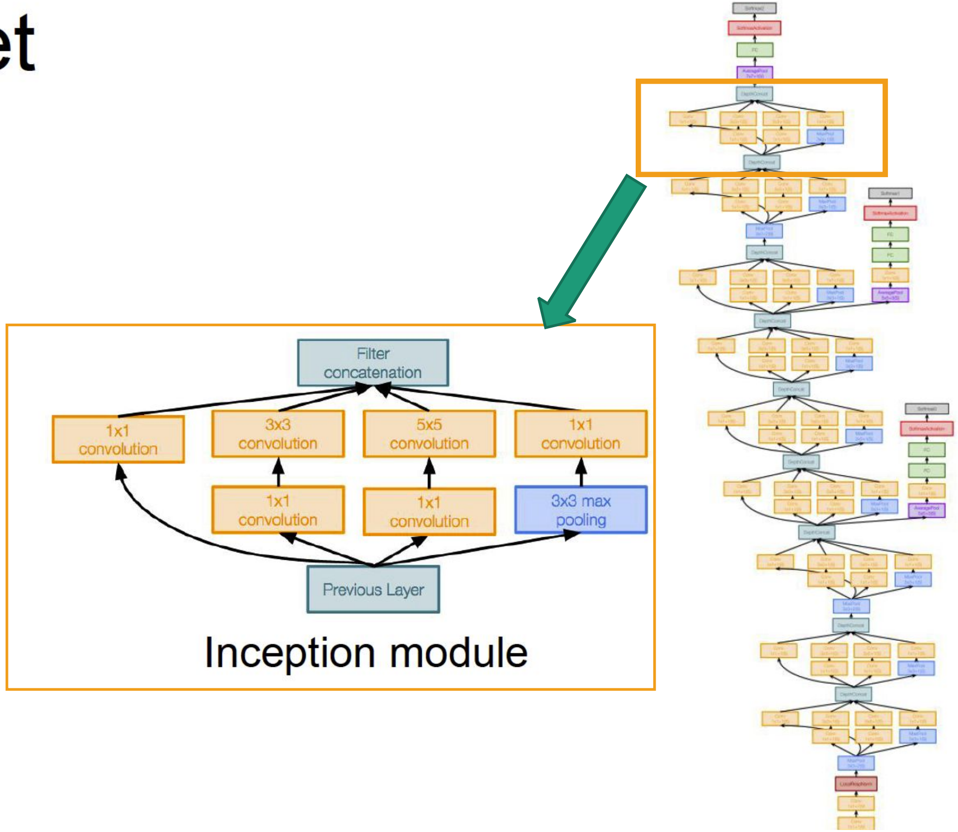
Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC’14 classification winner
  (6.7% top 5 error)



Inception module

Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

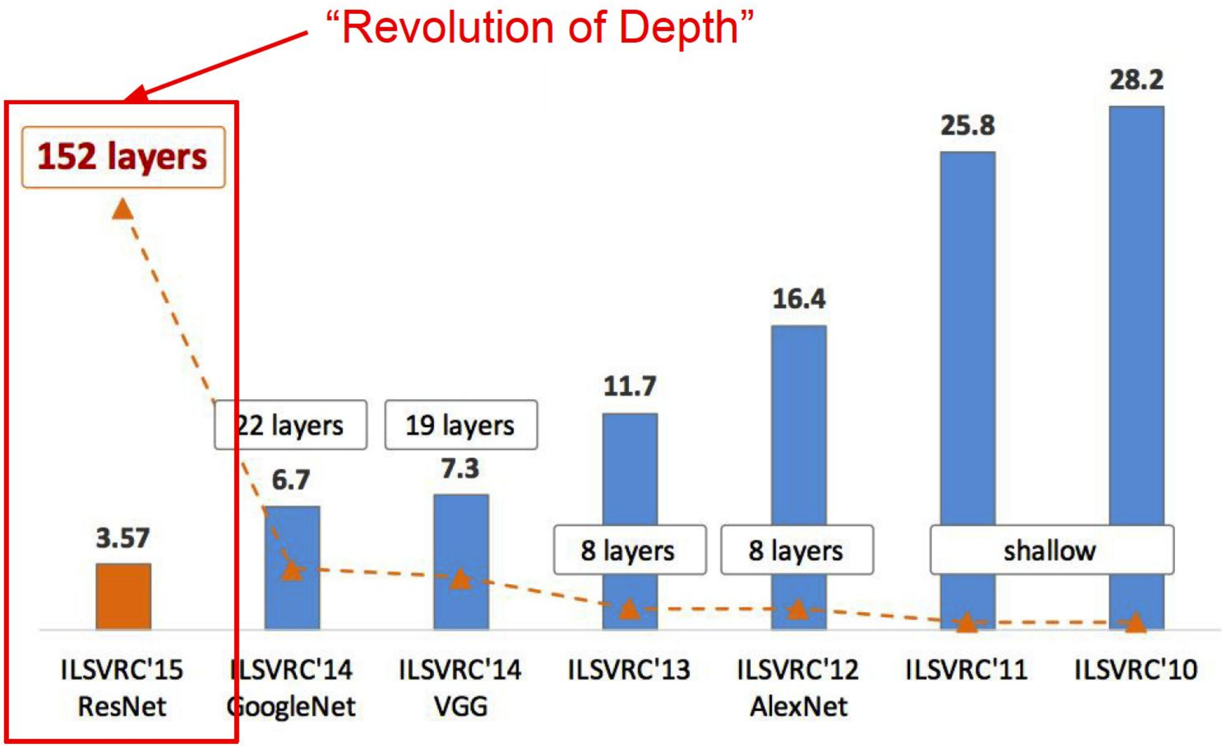
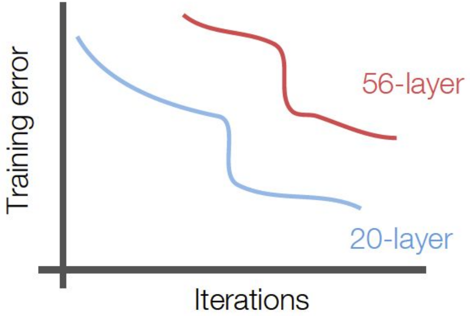
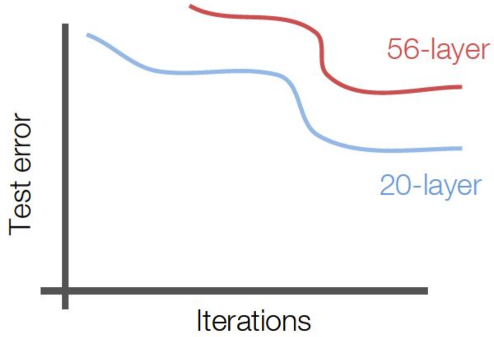


Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study 5: ResNet - 2015

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?





56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study 5: ResNet - 2015

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

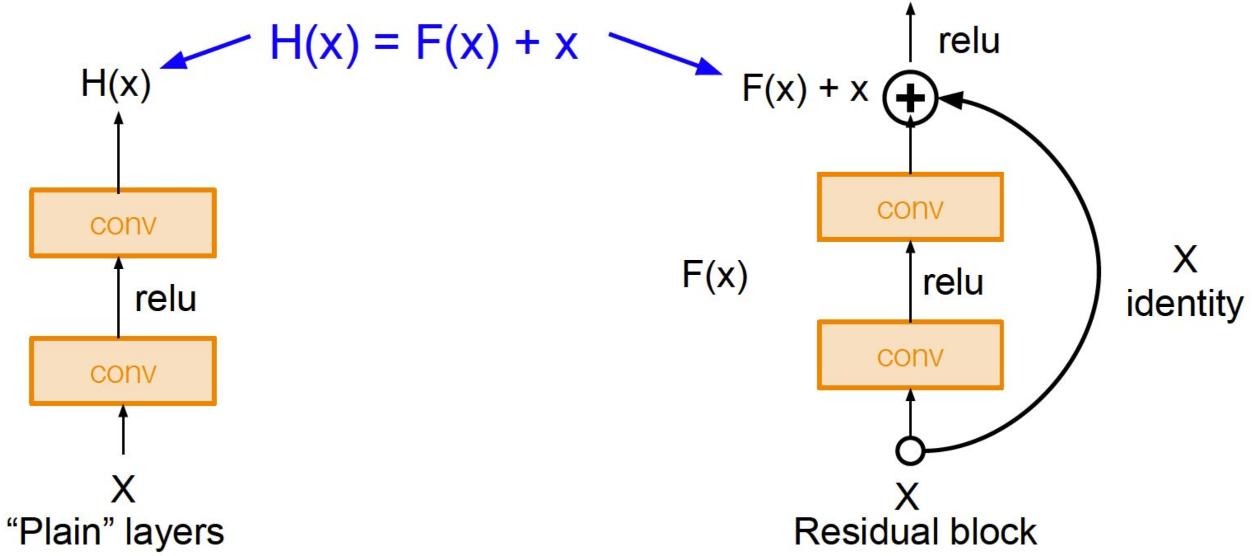
The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study 5: ResNet - 2015

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Use layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly

Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

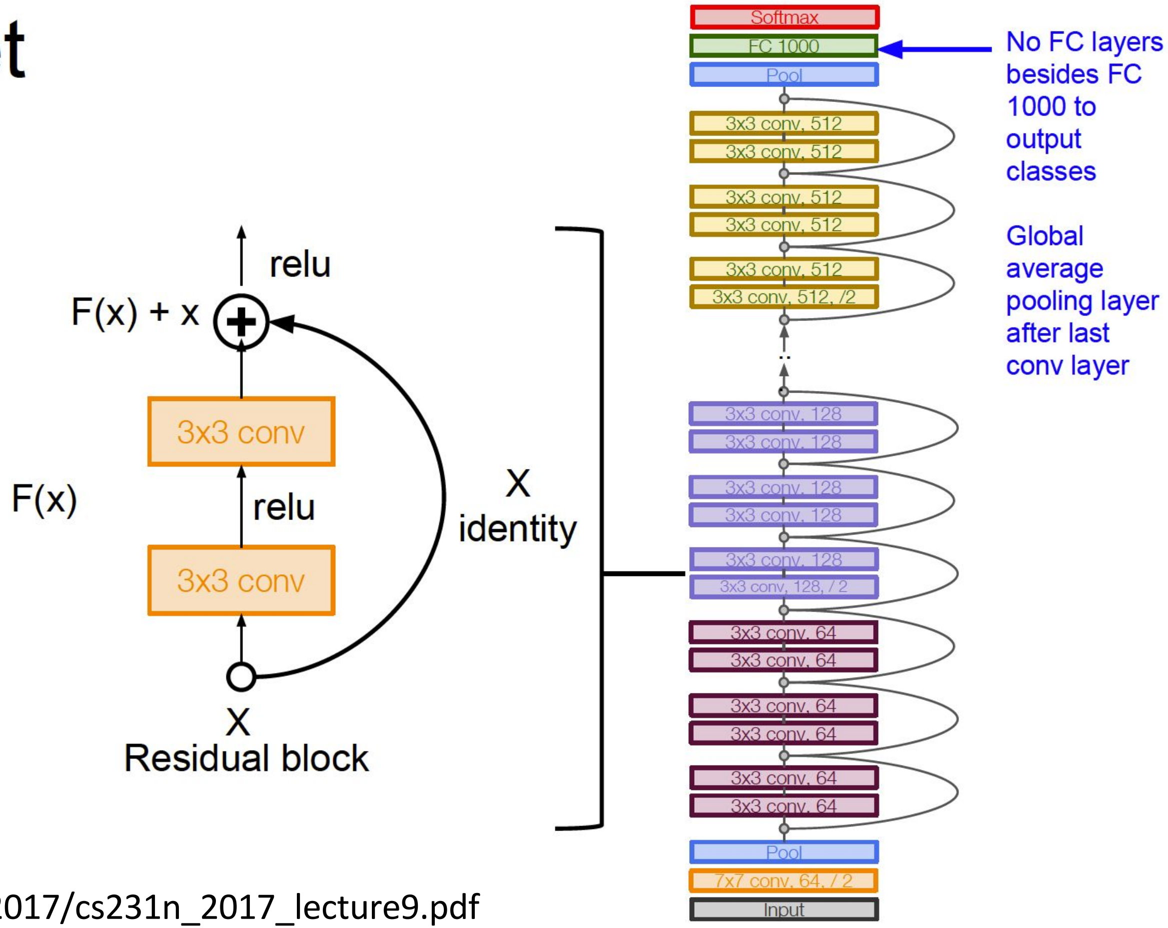


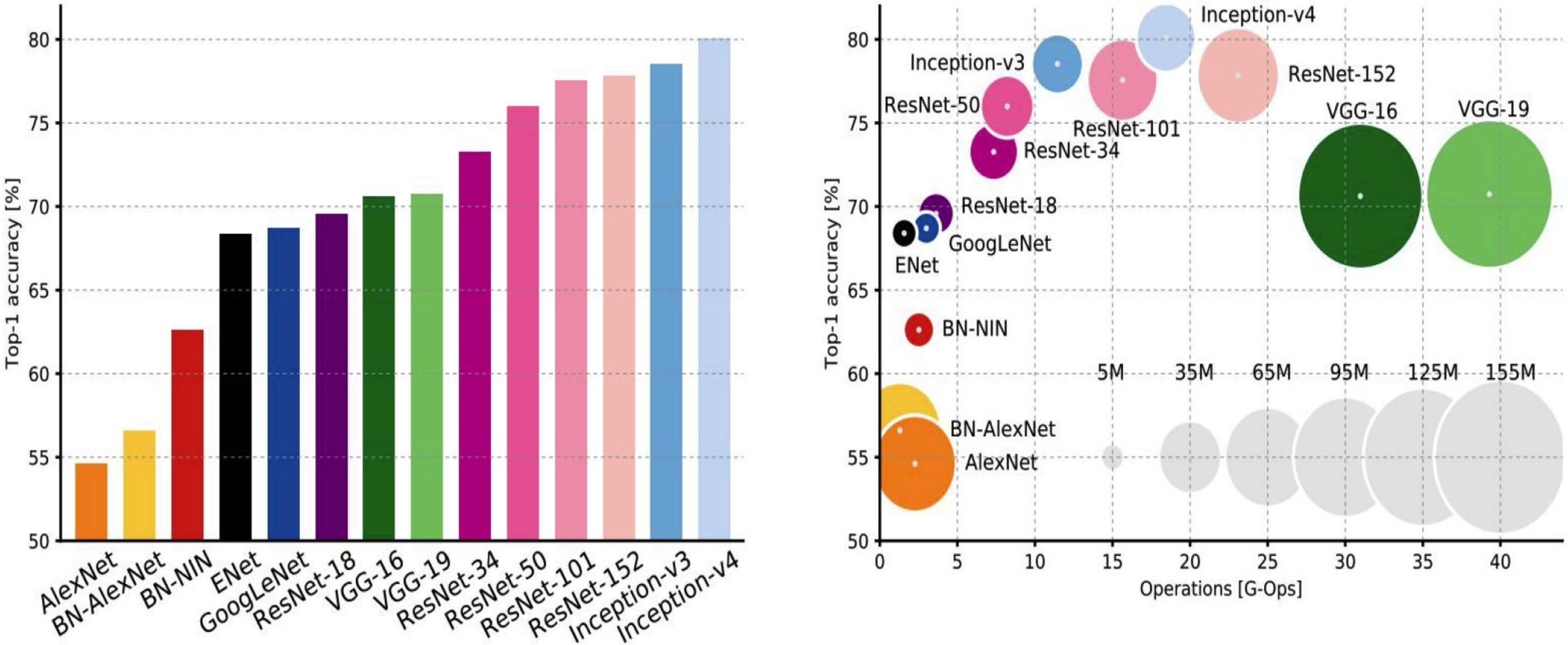
Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Case Study 5: ResNet - 2015

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

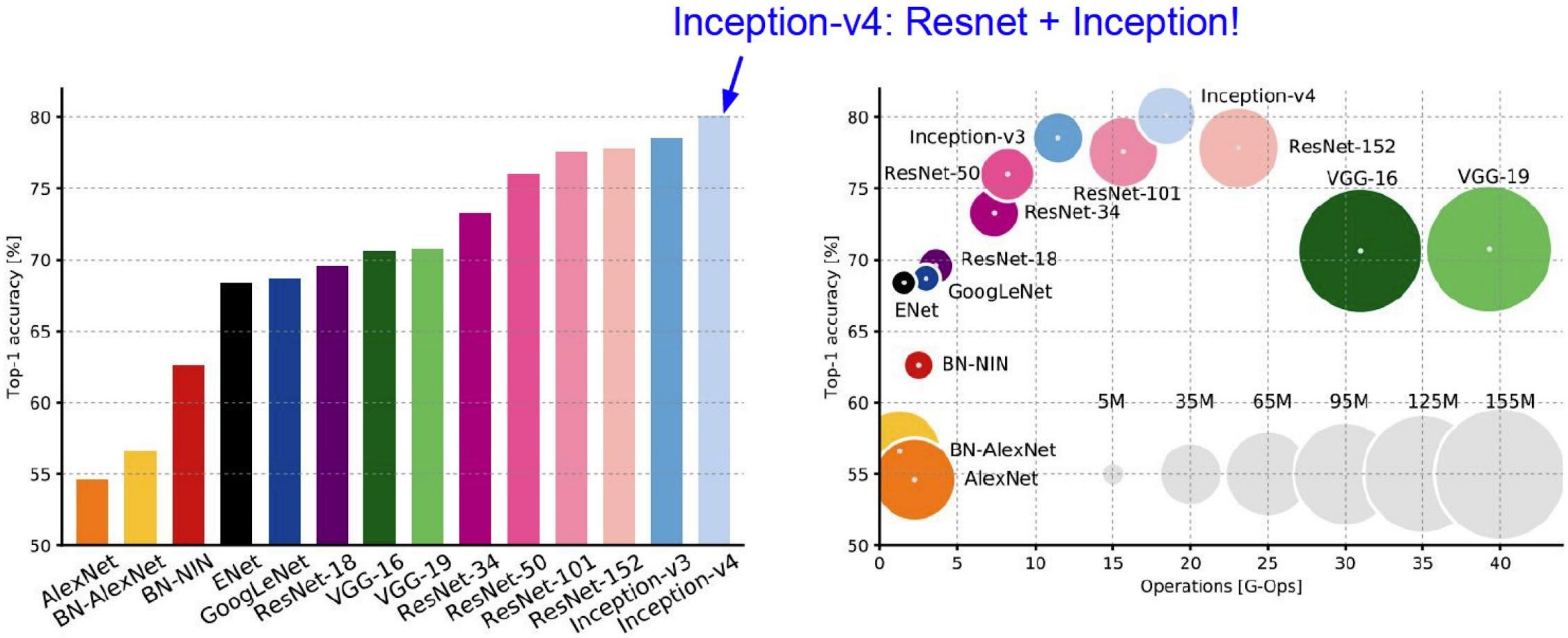
Complexity Comparisons



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Complexity Comparisons



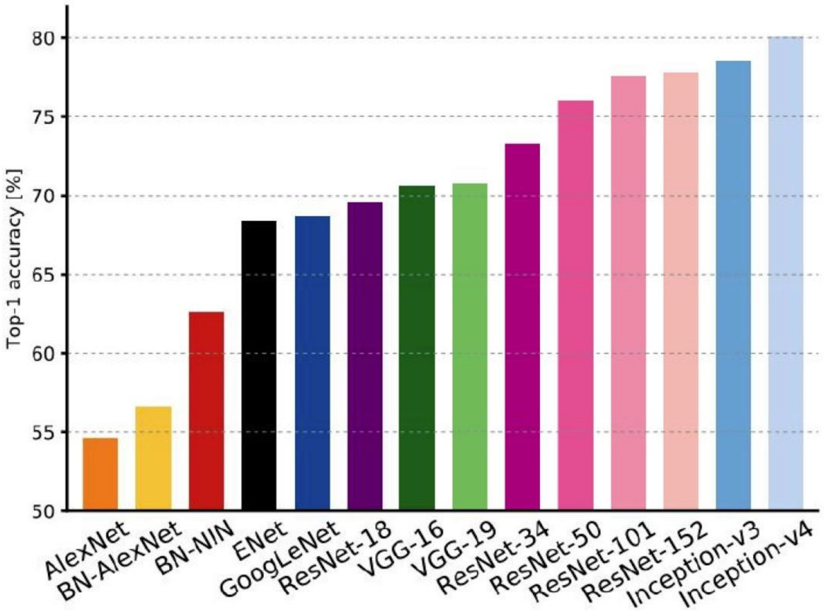
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

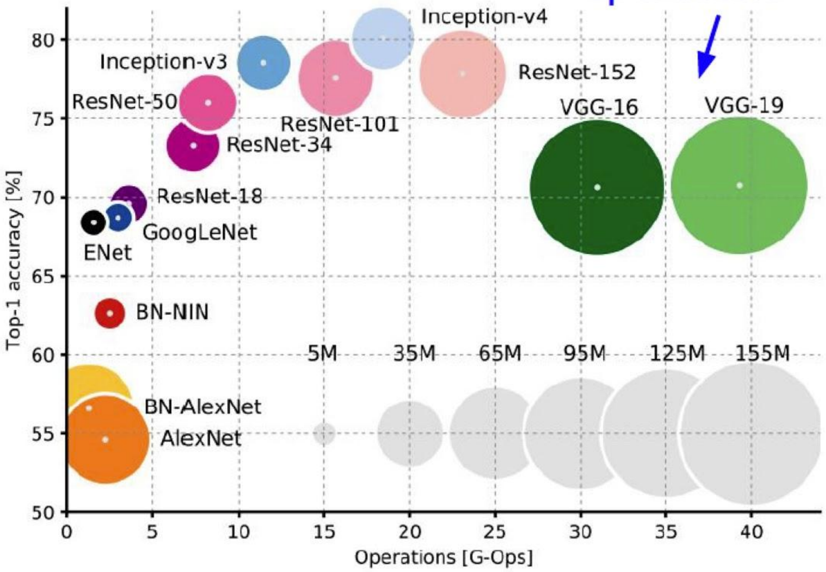
Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Complexity Comparisons

Comparing complexity...

VGG: Highest memory, most operations



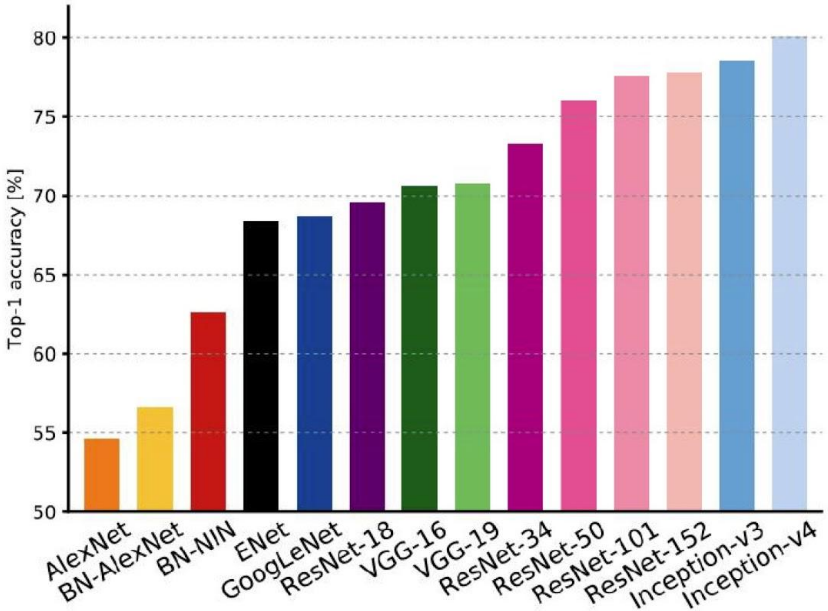


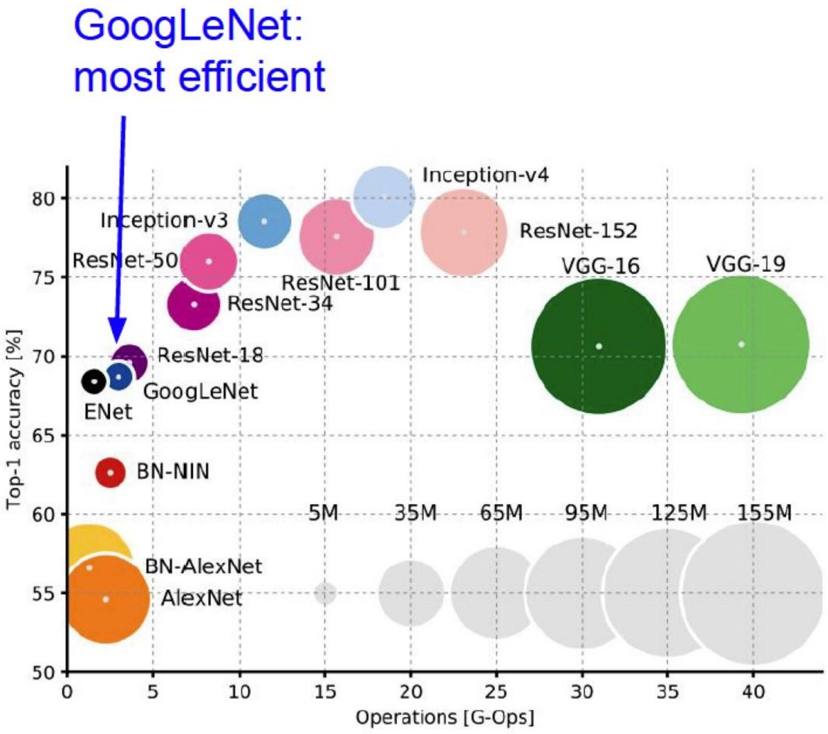
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Complexity Comparisons

Comparing complexity...



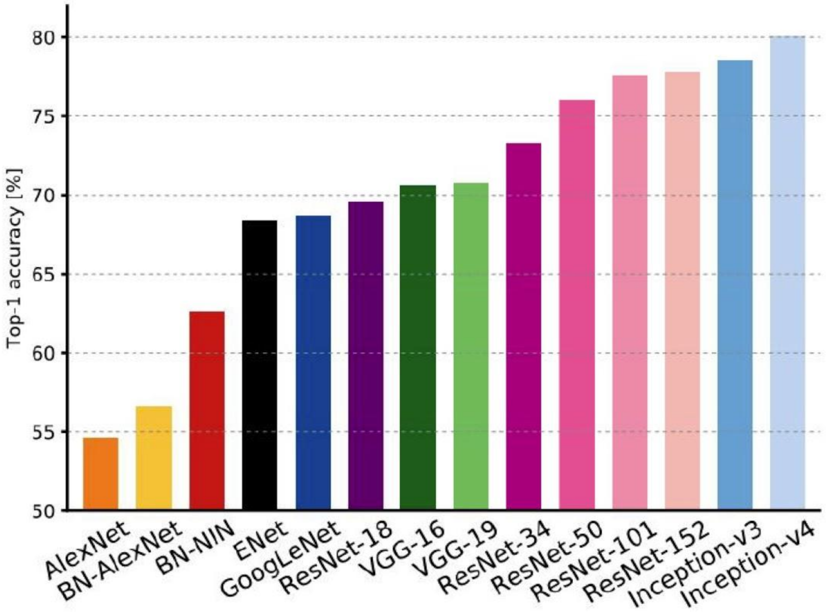


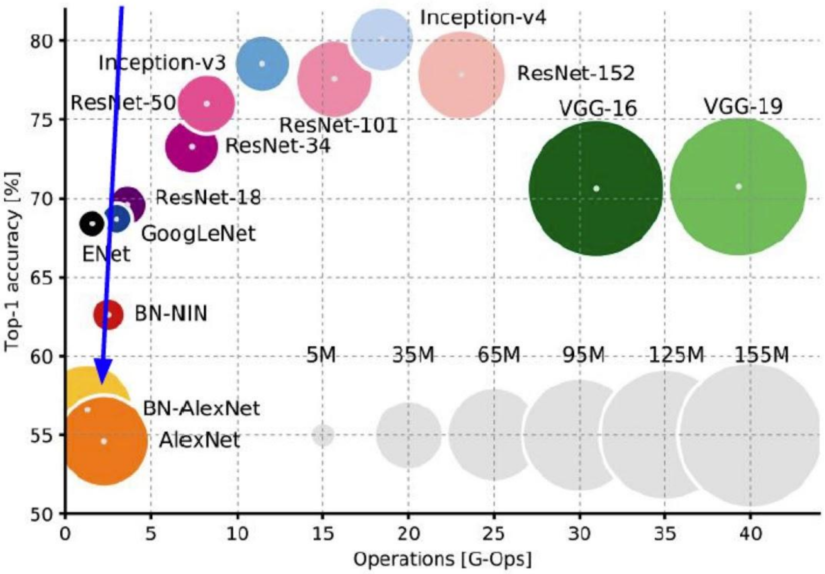
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...

AlexNet:
Smaller compute, still memory heavy, lower accuracy





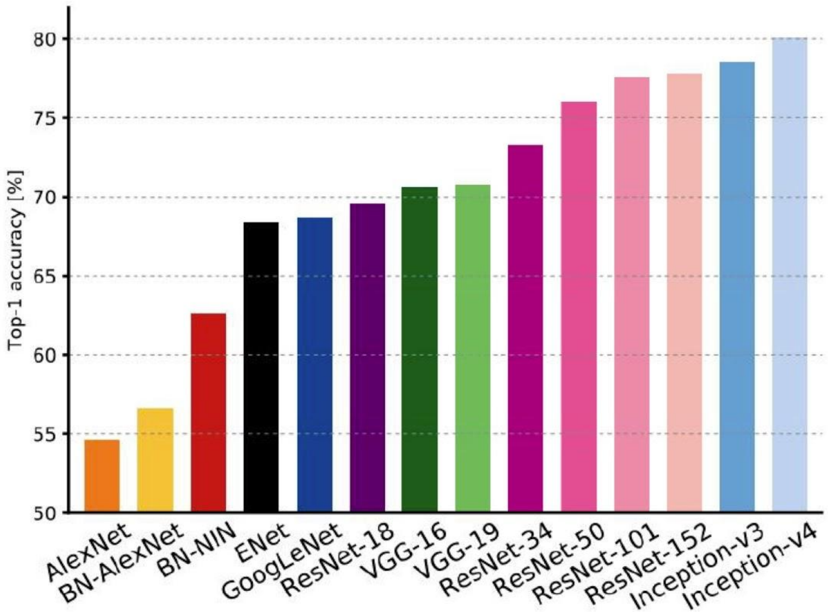
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

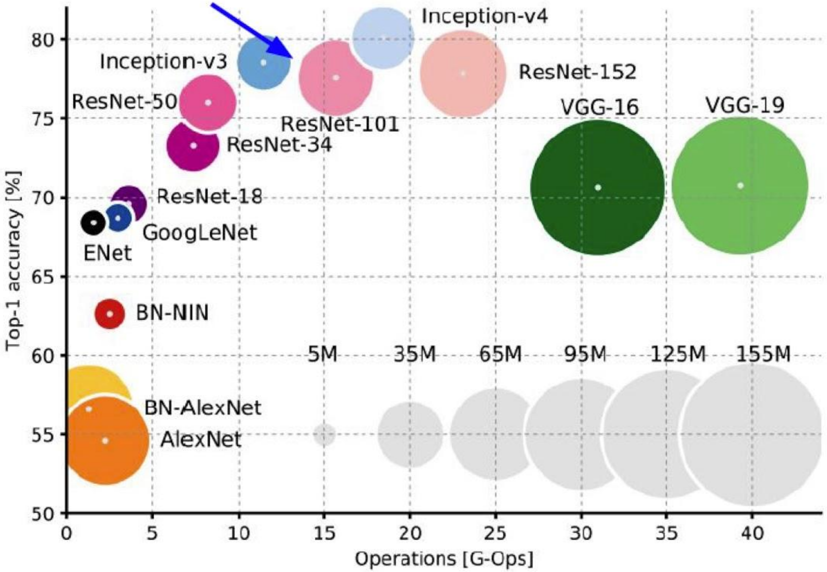
Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Complexity Comparisons

Comparing complexity...

ResNet:
Moderate efficiency depending on model, highest accuracy





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Improving ResNets...

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways ("cardinality")
- Parallel pathways similar in spirit to Inception module

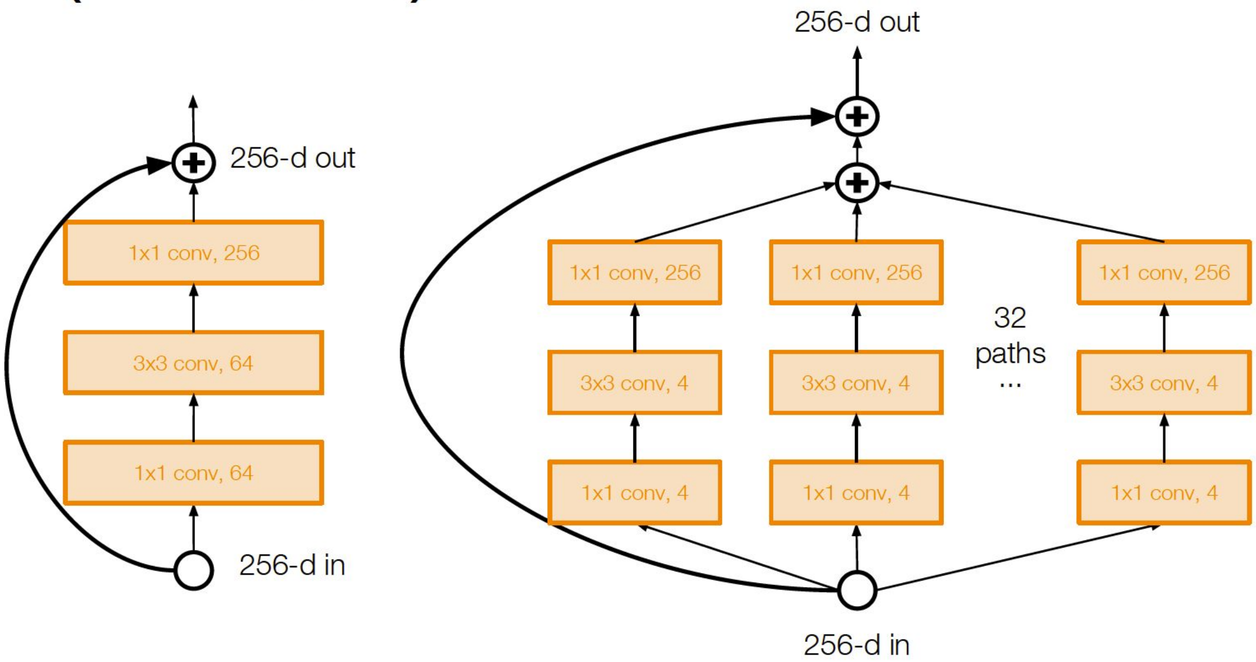


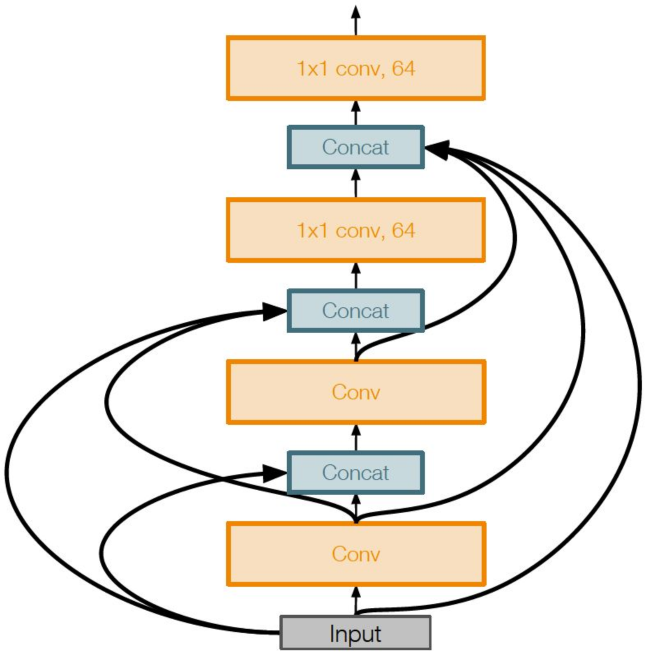
Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Beyond ResNets...

Densely Connected Convolutional Networks

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse



Dense Block

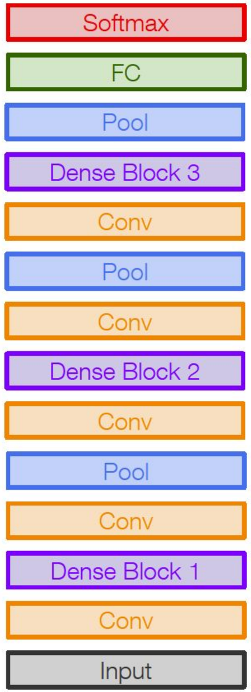
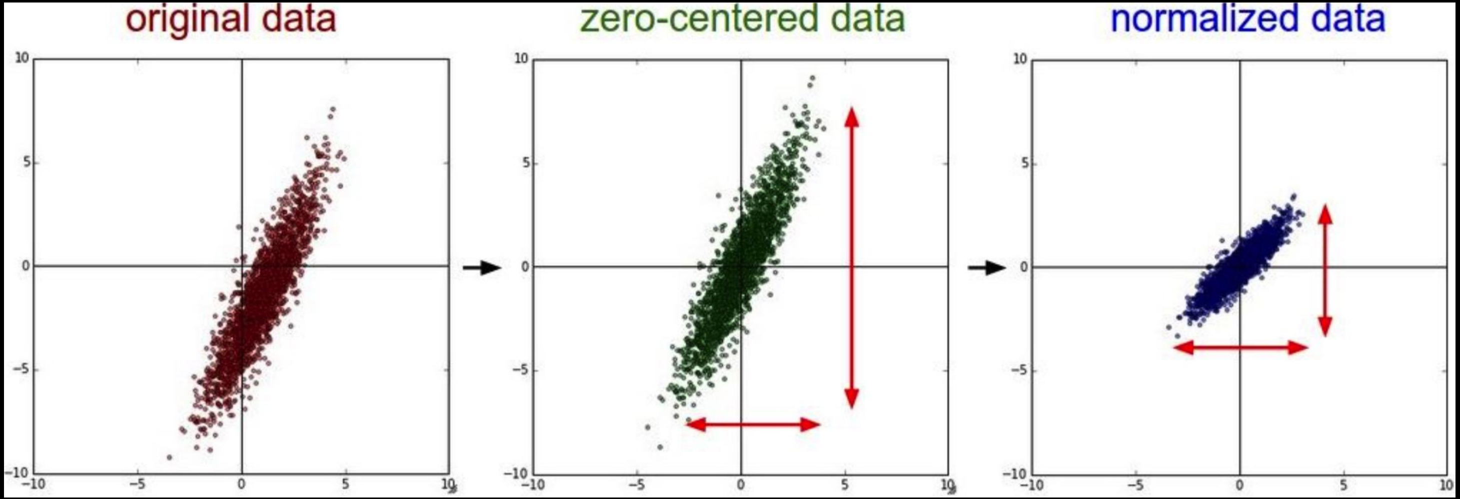


Image From http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Tips for training CNN

Know your data, clean your data, and normalize your data.
(A common trick: subtract the mean and divide its std.)

```
X -= np.mean(X, axis = 0) # zero-center
X /= np.std(X, axis = 0) # normalize
```



From http://www.cs.toronto.edu/~fidler/teaching/2015/slides/CSC2523/CNN-tutorial.pdf

Tips for training CNN

Augment your data:

horizontally flipping, random crops and color jittering.

From http://www.cs.toronto.edu/~fidler/teaching/2015/slides/CSC2523/CNN-tutorial.pdf

Tips for training CNN

Initialization:

a). **Calibrating the variances** with 1/sqrt(n)

w = np.random.randn(n) / sqrt(n) # (mean=0, var=1/n)

This ensures that all neurons have approximately the same output distribution and empirically improves the rate of convergence.

(For neural network with **ReLUs**, **w = np.random.randn(n) * sqrt(2.0/n)** Is recommended)

b). **Initializing the bias**:

Initialize the biases to be zero.

For ReLU non-linearities, some people like to use small constant value such as 0.01 for all biases .

References: https://arxiv.org/pdf/1502.01852.pdf (Delving Deep into Rectifiers...)

Tips for training CNN

Initialization:

c). Batch Normalization.
    Less sensitive to initialization

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

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

References: https://arxiv.org/abs/1502.03167 Batch Normalization: Accelerating Deep Network Training by Reducing ...

Tips for training CNN

Regularization:

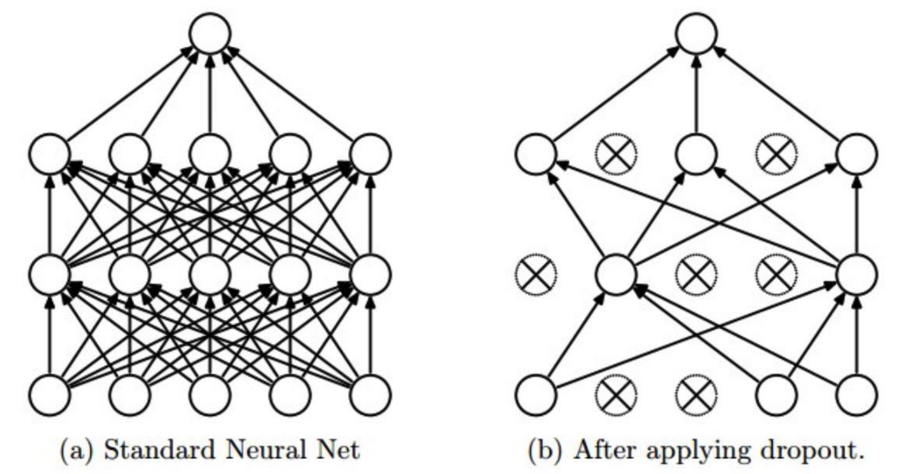
L1 : for sparsity

L2 : penalties peaky weight vectors, and prefers diffuse weight vectors.

Dropout:

Dropou ithin the full Neu e sampled networ

During tation of evaluat sized ensemb



From http://www.cs.toronto.edu/~fidler/teaching/2015/slides/CSC2523/CNN-tutorial.pdf

Tips for training CNN

Setting hyperparameters:

Learning Rate / Momentum ($\Delta wt^* = \Delta wt + m\Delta wt\text{-}1$)

Decrease learning rate while training

Setting momentum to 0.8 - 0.9

Batch Size:

For large dataset: set to whatever fits your memory

For smaller dataset: find a tradeoff between instance randomness and gradient smoothness

From http://www.cs.toronto.edu/~fidler/teaching/2015/slides/CSC2523/CNN-tutorial.pdf

Tips for training CNN

Monitoring your training (e.g. tensorboard):

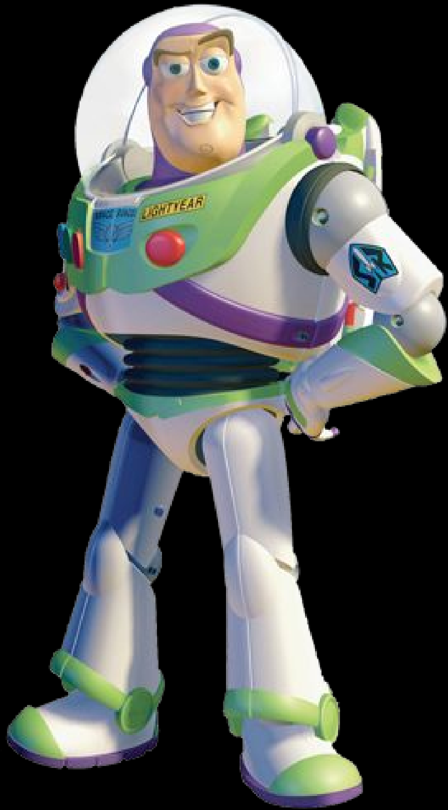
- Optimize your hyperparameter on val and evaluate on test
- Keep track of training and validation loss during training
- Do early stopping if training and validation loss diverge
- Loss doesn't tell you all. Try precision, class-wise precision, and more

From http://www.cs.toronto.edu/~fidler/teaching/2015/slides/CSC2523/CNN-tutorial.pdf

That’s it!

You’re now ready for field experience at the deep end of Star Command!

Remember: You can only learn while doing it yourself!



Acknowledgements/Other Resources

Yukun Zhu's tutorial from CSC2523 (2015):
http://www.cs.toronto.edu/~fidler/teaching/2015/slides/CSC2523/CNN-tutorial.pdf,

CS231n CNN Architectures (Stanford):
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

UIUC Advanced Deep Learning Course (2017):
http://slazebni.cs.illinois.edu/spring17/lec04_advanced_cnn.pdf