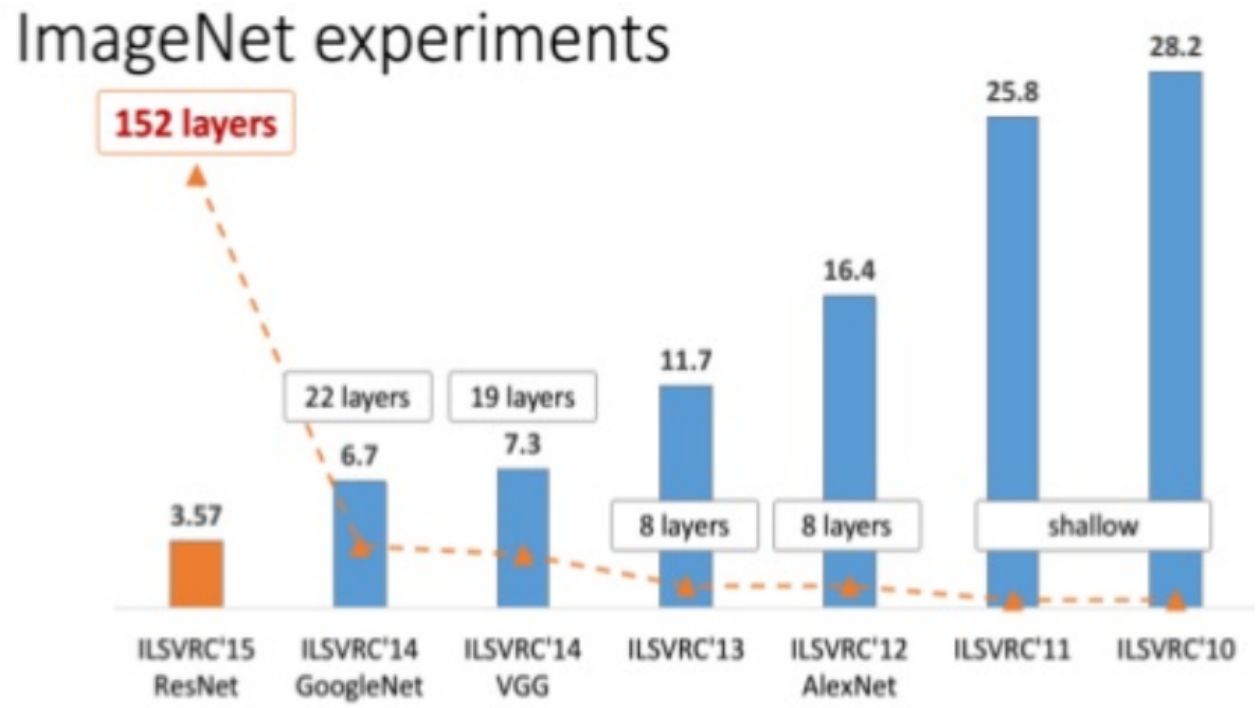
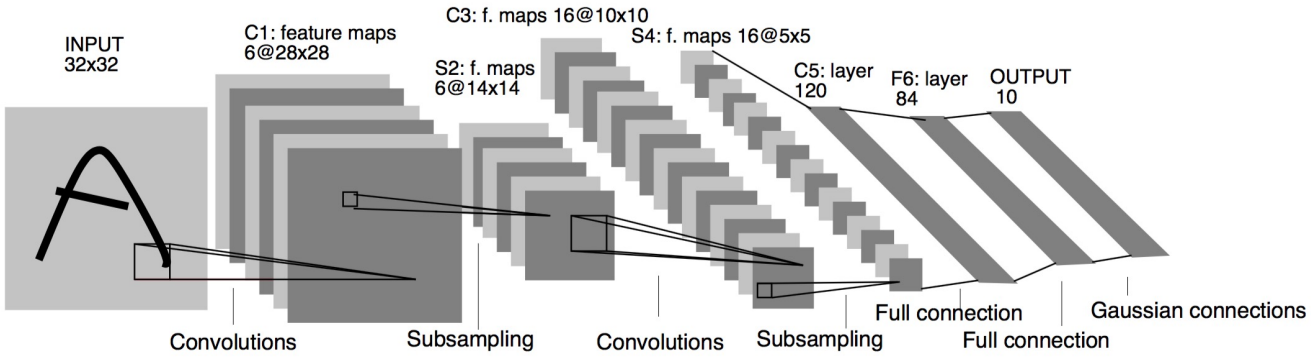
README9.md

09. CNN architectures

- This section talks about the famous CNN architectures. Focuses on CNN architectures that won ImageNet competition since 2012.



- These architectures includes: AlexNet, VGG, GoogLeNet, and ResNet.
- Also we will discuss some interesting architectures as we go.
- The first ConvNet that was made was LeNet-5 architectures are:by Yann Lecun at 1998.
  - Architecture are: `CONV-POOL-CONV-POOL-FC-FC-FC`



  - Each conv filters was `5x5` applied at stride 1
  - Each pool was `2x2` applied at stride `2`
  - It was useful in Digit recognition.
  - In particular the insight that image features are distributed across the entire image, and convolutions with learnable parameters are an effective way to extract similar features at multiple location with few parameters.
  - It contains exactly **5** layers

- In 2010 Dan Claudiu Ciresan and Jurgen Schmidhuber published one of the very fist implementations of GPU Neural nets. This implementation had both forward and backward implemented on a a NVIDIA GTX 280 graphic processor of an up to 9 layers neural network.

- **AlexNet** (2012):

  - ConvNet that started the evolution and wins the ImageNet at 2012.
  - Architecture are: `CONV1-MAXPOOL1-NORM1-CONV2-MAXPOOL2-NORM2-CONV3-CONV4-CONV5-MAXPOOL3-FC6-FC7-FC8`
  - Contains exactly **8** layers the first 5 are Convolutional and the last 3 are fully connected layers.
  - AlexNet accuracy error was `16.4%`
  - For example if the input is 227 x 227 x3 then these are the shapes of the of the outputs at each layer:
    - CONV1 (96 11 x 11 filters at stride 4, pad 0)
      - Output shape `(55,55,96)` , Number of weights are `(11*11*3*96)+96 = 34944`
    - MAXPOOL1 (3 x 3 filters applied at stride 2)
      - Output shape `(27,27,96)` , No Weights
    - NORM1
      - Output shape `(27,27,96)` , We don't do this any more
    - CONV2 (256 5 x 5 filters at stride 1, pad 2)
    - MAXPOOL2 (3 x 3 filters at stride 2)
    - NORM2
    - CONV3 (384 3 x 3 filters ar stride 1, pad 1)
    - CONV4 (384 3 x 3 filters ar stride 1, pad 1)
    - CONV5 (256 3 x 3 filters ar stride 1, pad 1)
    - MAXPOOL3 (3 x 3 filters at stride 2)
      - Output shape `(6,6,256)`
    - FC6 (4096)
    - FC7 (4096)
    - FC8 (1000 neurons for class score)
  - Some other details:
    - First use of RELU.
    - Norm layers but not used any more.
    - heavy data augmentation
    - Dropout `0.5`
    - batch size `128`
    - SGD momentum `0.9`
    - Learning rate `1e-2` reduce by 10 at some iterations
    - 7 CNN ensembles!
  - AlexNet was trained on GTX 580 GPU with only 3 GB which wasn't enough to train in one machine so they have spread the feature maps in half. The first AlexNet was distributed!
  - Its still used in transfer learning in a lot of tasks.
  - Total number of parameters are `60 million`

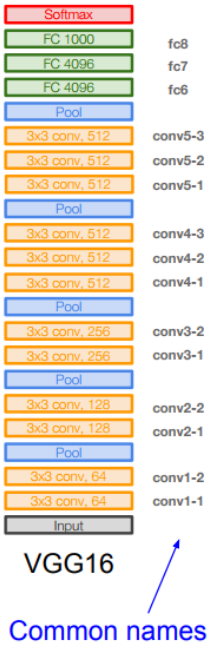
- **ZFNet** (2013)

  - Won in 2013 with error 11.7%
  - It has the same general structure but they changed a little in hyperparameters to get the best output.
  - Also contains **8** layers.
  - AlexNet but:
    - `CONV1` : change from (11 x 11 stride 4) to (7 x 7 stride 2)
    - `CONV3,4,5` : instead of 384, 384, 256 filters use 512, 1024, 512

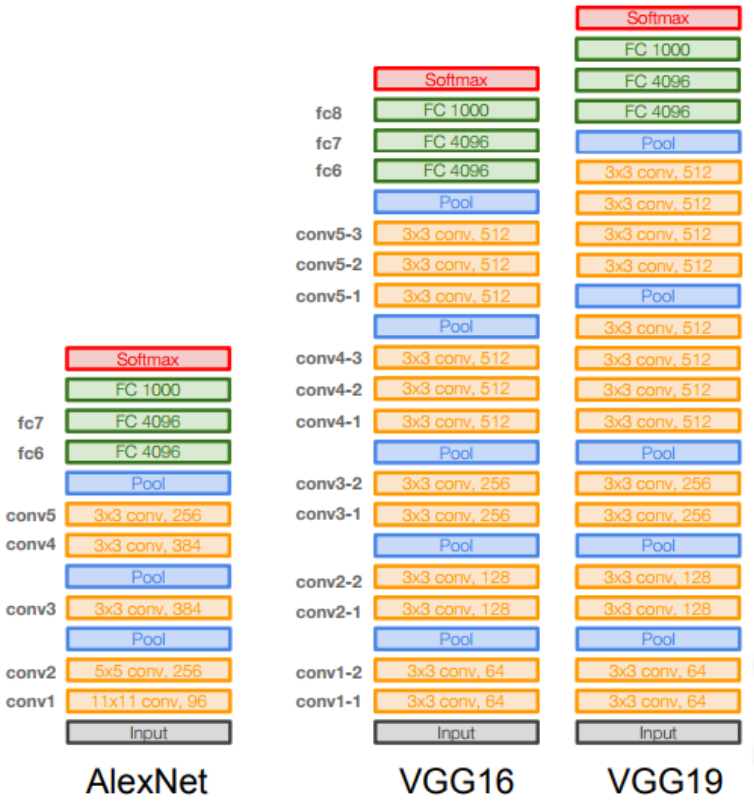
- OverFeat (2013)
  - Won the localization in imageNet in 2013
  - We show how a multiscale and sliding window approach can be efficiently implemented within a ConvNet. We also introduce a novel deep learning approach to localization by learning to predict object boundaries.
- **VGGNet** (2014) (Oxford)
  - Deeper network with more layers.
  - Contains 19 layers.
  - Won on 2014 with GoogleNet with error 7.3%
  - Smaller filters with deeper layers.
  - The great advantage of VGG was the insight that multiple 3 × 3 convolution in sequence can emulate the effect of larger receptive fields, for examples 5 × 5 and 7 × 7.
  - Used the simple 3 x 3 Conv all through the network.
    - 3 (3 x 3) filters has the same effect as 7 x 7

    INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
    CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
    CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
    POOL2: [112x112x64] memory: 112*112*64=800K params: 0
    CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
    CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
    POOL2: [56x56x128] memory: 56*56*128=400K params: 0
    CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
    CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
    CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
    POOL2: [28x28x256] memory: 28*28*256=200K params: 0
    CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
    CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
    CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
    POOL2: [14x14x512] memory: 14*14*512=100K params: 0
    CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
    CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
    CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
    POOL2: [7x7x512] memory: 7*7*512=25K params: 0
    FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
    FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
    FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

    TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
    TOTAL params: 138M parameters

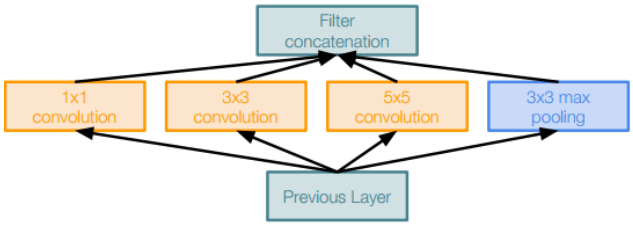
    

  - The Architecture contains several CONV layers then POOL layer over 5 times and then the full connected layers.
  - It has a total memory of 96MB per image for only forward propagation!
    - Most memory are in the earlier layers
  - Total number of parameters are 138 million
    - Most of the parameters are in the fully connected layers
  - Has a similar details in training like AlexNet. Like using momentum and dropout.
  - VGG19 are an upgrade for VGG16 that are slightly better but with more memory

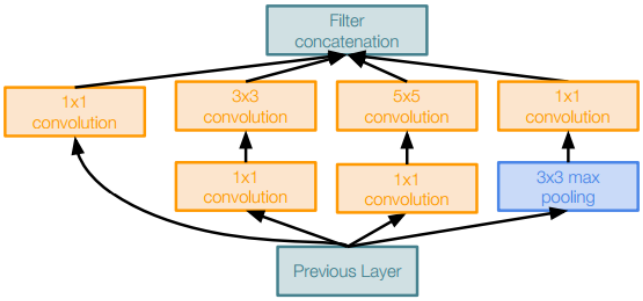


- [GoogleNet](#) (2014)
  - Deeper network with more layers.
  - Contains 22 layers.
  - It has Efficient **Inception** module.
  - Only 5 million parameters! 12x less than AlexNet
  - Won on 2014 with VGGNet with error 6.7%
  - Inception module:
    - Design a good local network topology (network within a network (NiN)) and then stack these modules on top of each other.
    - It consists of:
      - Apply parallel filter operations on the input from previous layer
        - Multiple convs of sizes (1 x 1, 3 x 3, 5 x 5)
          - Adds padding to maintain the sizes.
        - Pooling operation. (Max Pooling)
          - Adds padding to maintain the sizes.
      - Concatenate all filter outputs together depth-wise.
    - For example:
      - Input for inception module is 28 x 28 x 256
      - Then the parallel filters applied:
        - (1 x 1), 128 filter `# output shape (28,28,128)`
        - (3 x 3), 192 filter `# output shape (28,28,192)`
        - (5 x 5), 96 filter `# output shape (28,28,96)`
        - (3 x 3) Max pooling `# output shape (28,28,256)`
      - After concatenation this will be `(28,28,672)`
    - By this design -We call Naive- it has a big computation complexity.
      - The last example will make:
        - [1 x 1 conv, 128] ==> 28 * 28 * 128 * 1 * 1 * 256 = 25 Million approx

        - [3 x 3 conv, 192] ==> 28 * 28 * 192 *3 *3 * 256 = 346 Million approx
        - [5 x 5 conv, 96] ==> 28 * 28 * 96 * 5 * 5 * 256 = 482 Million approx
        - In total around 854 Million operation!
    - Solution: **bottleneck** layers that use 1x1 convolutions to reduce feature depth.
        - Inspired from NiN (Network in network)

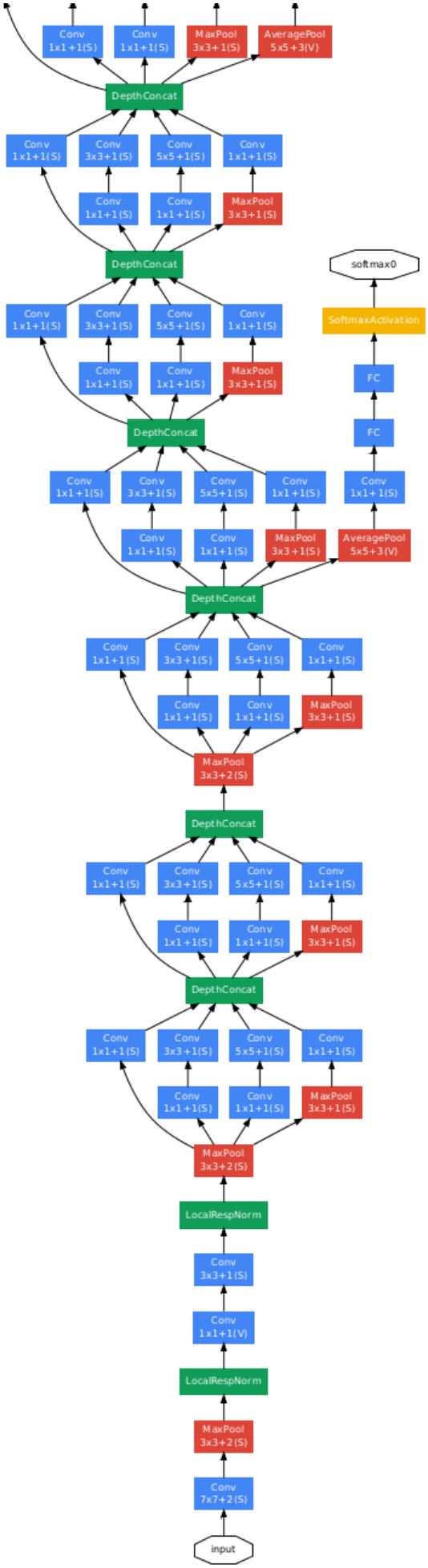


Naive Inception module

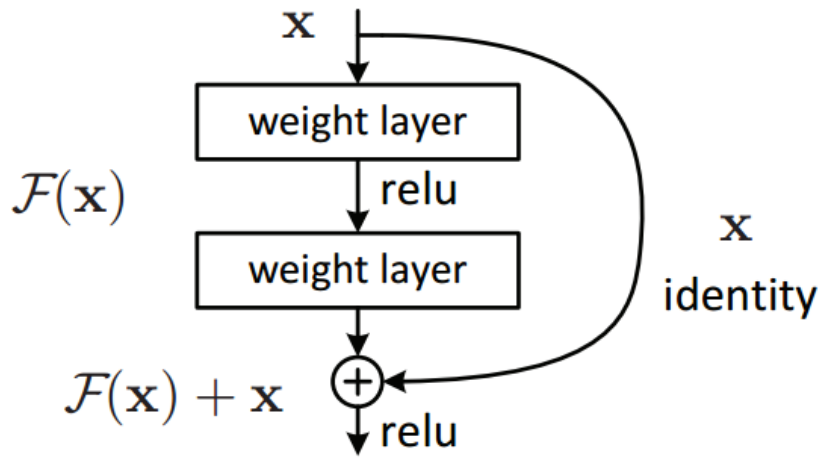


Inception module with dimension reduction

        -
        - The bottleneck solution will make a total operations of 358M on this example which is good compared with the naive implementation.
    - So GoogleNet stacks this Inception module multiple times to get a full architecture of a network that can solve a problem without the Fully connected layers.
    - Just to mention, it uses an average pooling layer at the end before the classification step.
    - Full architecture:

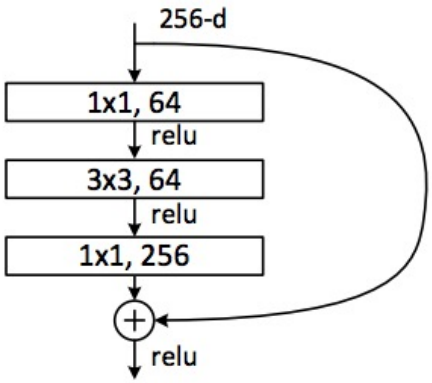

Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
softmax0
SoftmaxActivation
FC
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 3x3+1(S)
Conv 1x1+1(V)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 7x7+2(S)
input

    -
  - In February 2015 Batch-normalized Inception was introduced as Inception V2. Batch-normalization computes the mean and standard-deviation of all feature maps at the output of a layer, and normalizes their responses with these values.
  - In December 2015 they introduced a paper "Rethinking the Inception Architecture for Computer Vision" which explains the older inception models well also introducing a new version V3.
- The first GoogleNet and VGG was before batch normalization invented so they had some hacks to train the NN and converge well.
- **ResNet** (2015) (Microsoft Research)
  - 152-layer model for ImageNet. Winner by 3.57% which is more than human level error.
  - This is also the very first time that a network of > hundred, even 1000 layers was trained.
  - Swept all classification and detection competitions in ILSVRC'15 and COCO'15!
  - What happens when we continue stacking deeper layers on a "plain" Convolutional neural network?
    - The deeper model performs worse, but it's not caused by overfitting!
    - The learning stops performs well somehow because deeper NN 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.
  - Residual block:
    - Microsoft came with the Residual block which has this architecture:

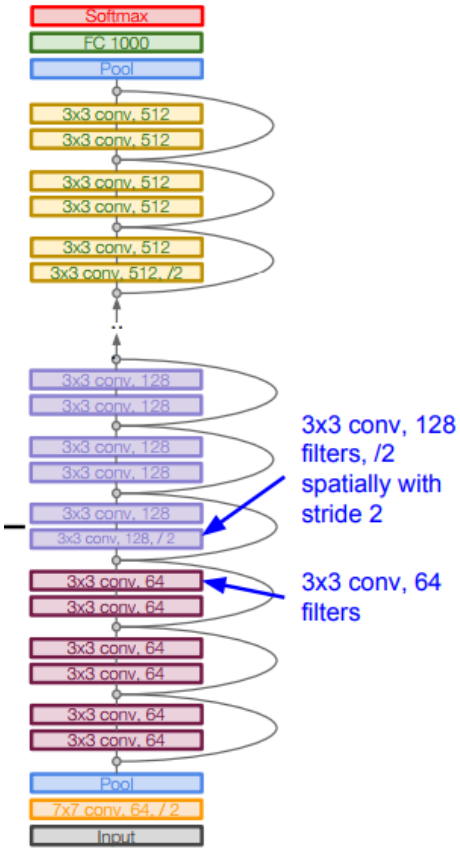


    -
    - ```
      # Instead of us trying To learn a new representation, We learn only Residual
      Y = (W2* RELU(W1x+b1) + b2) + X
      ```
    - Say you have a network till a depth of N layers. You only want to add a new layer if you get something extra out of adding that layer.
    - One way to ensure this new (N+1)th layer learns something new about your network is to also provide the input(x) without any transformation to the output of the (N+1)th layer. This essentially drives the new layer to learn something different from what the input has already encoded.
    - The other advantage is such connections help in handling the Vanishing gradient problem in very deep networks.
  - With the Residual block we can now have a deep NN of any depth without the fearing that we can't optimize the network.
      ```

- ResNet with a large number of layers started to use a bottleneck layer similar to the Inception bottleneck to reduce the dimensions.



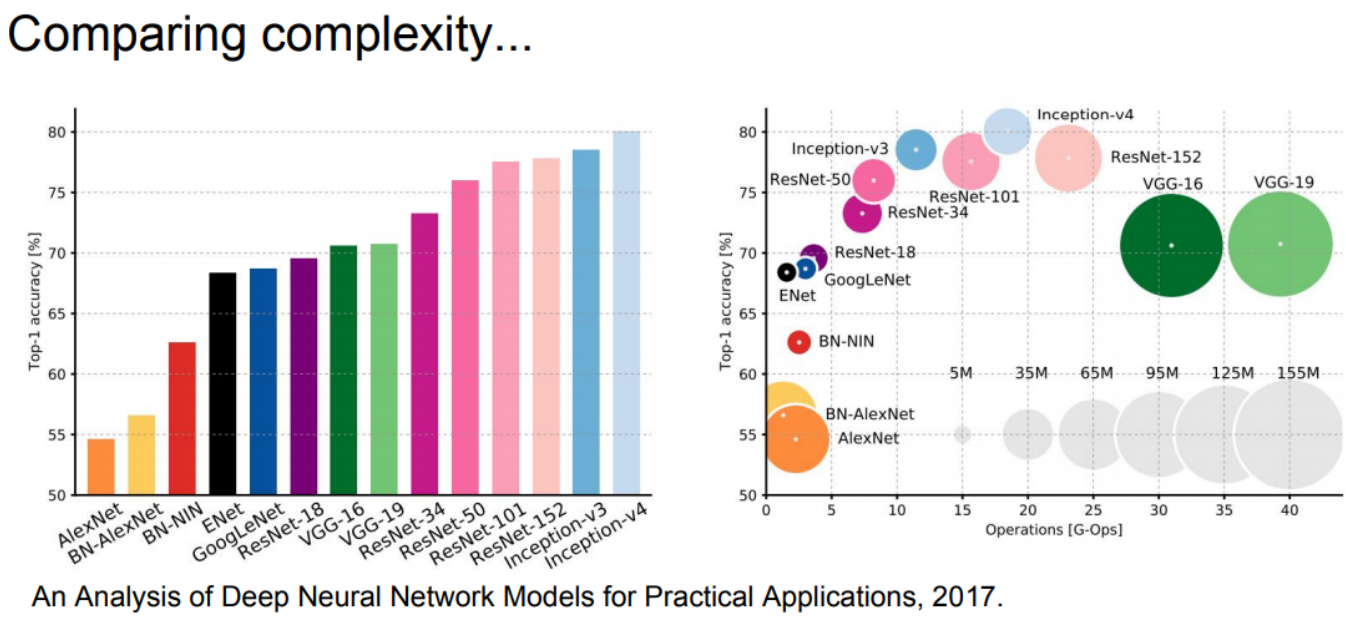
- **Full ResNet architecture**:
  - Stack residual blocks.



  - Every residual block has two 3 x 3 conv layers.
  - Additional conv layer at the beginning.
  - No FC layers at the end (only FC 1000 to output classes)
  - Periodically, double number of filters and downsample spatially using stride 2 (/2 in each dimension)
  - 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.

- Inception-v4: Resnet + Inception and was founded in 2016.
- The complexity comparing over all the architectures:



        - Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.
    - VGG: Highest memory, most operations.
    - GoogLeNet: most efficient.

- **ResNets Improvements**:
    - (2016) Identity Mappings in Deep Residual Networks
        - From the creators of ResNet.
        - Gives better performance.
    - (2016) Wide Residual Networks
        - Argues that residuals are the important factor, not depth
        - 50-layer wide ResNet outperforms 152-layer original ResNet
        - Increasing width instead of depth more computationally efficient (parallelizable)
    - (2016) Deep Networks with Stochastic Depth
        - Motivation: reduce vanishing gradients and training time through short networks during training.
        - Randomly drop a subset of layers during each training pass
        - Use full deep network at test time.

- **Beyond ResNets**:
    - (2017) FractalNet: Ultra-Deep Neural Networks without Residuals
        - Argues that key is transitioning effectively from shallow to deep and residual representations are not necessary.
        - Trained with dropping out sub-paths
        - Full network at test time.
    - (2017) Densely Connected Convolutional Networks
    - (2017) SqueezeNet: AlexNet-level Accuracy With 50x Fewer Parameters and <0.5Mb Model Size
        - Good for production.

        - It is a re-hash of many concepts from ResNet and Inception, and show that after all, a better design of architecture will deliver small network sizes and parameters without needing complex compression algorithms.

- Conclusion:
    - ResNet current best default.
    - Trend towards extremely deep networks
    - In the last couple of years, some models all using the shortcuts like "ResNet" to easily flow the gradients.