CNN Architectures

Sunday, December 12, 2021 10:02 PM

1. **CNN Architectures**

Computational Resource Analysis

- Compare architectures:
  - Runtime considerations:
    - how long to make prediction (forward prop)
    - how long to train (bakward prop, # params)
  - Memory considerations
    - how much memory (GPU memory) needed
      - train the model (fwd + bkd prop)
      - make a prediction
- **Number of floating point operations (FLOPs) for conv layer**
  - conv = multiply-accumulate (MAC) operations
  - One MAC = single FLOP
  - each activation = dot prod b/w 2 (f,f,c_in) tensors = f*f*c_in MACs
  - Total FLOPs:
    - Total flops: $h_{out} * w_{out} * K * f * f * c_{in}$
      - Number of outputs * Number of flops to compute each output
- **Number of FLOPs for Pooling layer**
  - for single (f,f) region for pooling:
    - max pool = f*f flops
    - avg pool = f*f flops
  - for pooling layer w/ output shape (h_out, w_out, c_out)
    - h_out*w_out*c_out
    - each pooling region needs f*f flops
  - Total FLOPs:
    - Total flops: $h_{out} * w_{out} * c_{out} * f * f$
      - Number of outputs * Number of flops to compute each output
- **Number of FLOPs for FCL**
  - layer has n_f[l] units and n_h[l-1] inputs
  - output of each unit = weighted sum of n_f[l] num = n_f[l-1] MACs
  - Total FLOPs:
    - Output of all units is $n_h^{[l]} * n_h^{[l-1]}$
- Real metrics should be throughput/latency/energy/power of real application
- **Number of parameters per layer**
  - Convolutional Layer:
    - $K(f * f * c_{in} + 1)$
  - MaxPooling Layer:
    - 0
  - Fully Connected:
    - $n_h^{[l]}\left(n_h^{[l-1]} + 1\right)$

LeNet Analysis

- stack conv and pooling a number of times, followed by some FCL

Layer	HyperParams	Output Volume	# Parameters	flops
Input		(32,32,1)		
Conv	K=6, f=(5,5), s=(1,1)	(28,28,6)	6*(5*5*1+1)=156	117k
Avg. Pool	f=(2,2), s=(2,2)	(14,14,6)		4074
Conv	K=16, f=(5,5), s=(1,1)	(10,10,16)	16*(5*5*6+1)=2416	240k
Avg. Pool	f=(2,2), s=(2,2)	(5,5,16)		1.6k
Flatten		(400,)		0
FC	120 units	(120,)	120*(400+1) = 48,120	48k
FC	84 units	(84,)	84*(120+1)=10,164	10k
FC (Output)	10 units	(10,)	10*(84+1)=850	840

~0.42MFlop for one forward pass, ~62k parameters

AlexNet

- popular CNN for computer vision

Layer	HyperParams	Output	# Params	Mflops
Input		(227,227,3)		
Conv1	K=96,f=(11,11,),s=(4,4)	(55,55,96)	34,857	105
Max Pool	f=(3,3),s=(2,2)	(27,27,96)	0	0.6
Conv2	K=256,f=(5,5),s=(1,1),p=same	(27,27,256)	614,656	448
Max Pool	f=(3,3),s=(2,2)	(13,13,256)	0	0.4
Conv3	K=384,f=(3,3),s=(1,1),p=same	(13,13,384)	885,120	150
Conv4	K=384,f=(3,3),s=(1,1),p=same	(13,13,384)	1,327,488	224
Conv5	K=256,f=(3,3),s=(1,1),p=same	(13,13,256)	884,992	150
Max Pool	f=(3,3),s=(2,2)	(6,6,256)	0	0.08
Flatten		(9216,)	0	
FC	n=4096	(4096,)	37,752,832	38
FC	n=4096	(4096,)	16,781,312	17
FC(softmax)	n=1000	(1000,)	4,097,000	4

- ~1000 MFlop for one fwd pass
- ~62 million parameters
- Most params in FCL
- Most computations in conv layers
- pooling layers are "free"
- Popularized ReLU
- Overlapping Pooling helped them reduce error, helps model generalize (reduce overfit)
- Used local response normalization layers
- Architecture hyperparameters chosen by trial-and-error

ZFNet

- A bigger AlexNet

Layer	HyperParams	Output	# Params	Mflops
Input		(224,224,3)		
Conv1	K=96,f=(7,7),s=(2,2)	(110,110,96)	14,208	170.8
Max Pool	f=(3,3),s=(2,2)	(55,55,96)	0	2.6
Conv2	K=256,f=(5,5),s=(1,1),p=same	(26,26,256)	614,656	415.3
Max Pool	f=(3,3),s=(2,2)	(13,13,256)	0	0.09
Conv3	K=512,f=(3,3),s=(1,1),p=same	(13,13,512)	1,180,160	199.4
Conv4	K=1024,f=(3,3),s=(1,1),p=same	(13,13,1024)	4,719,616	797.4
Conv5	K=512,f=(3,3),s=(1,1),p=same	(13,13,512)	4,719,616	797.4
Max Pool	f=(3,3),s=(2,2)	(6,6,512)	0	0.17
Flatten		(18432,)	0	
FC	n=4096	(4096,)	75,515,904	75.5
FC	n=4096	(4096,)	16,781,312	16.8
FC(softmax)	n=1000	(1000,)	4,097,000	4.1

- ~2.47 Gflop for one fwd pass
- ~108 million params

- Bigger capacity is still better
- Still trial-and-error for architecture design
- no consideration of computation efficiency

VGGNet

- All conv layers are 3x3 stride 1, same padding
  - single 5x5 layer vs 2 3x3 layers
    - Parameters
      - Two (3x3) layers: $2 * K * (3 * 3 * c_{in} + 1)$
      - One (5x5) layer: $K * (5 * 5 * c_{in} + 1)$
      - → One (5x5) has ~ 25/18 ~ 1.4x more learned parameters
    - FLOPs:
      - Two (3x3) layers : $2 * h_{out} * w_{out} * c_{out} * 3 * 3 * c_{in}$
      - One (5x5) layer: $h_{out} * w_{out} * c_{out} * 5 * 5 * c_{in}$
      - → One (5x5) layer needs more flops (25/18 ~ 1.4x more)
    - Memory
      - Two (3x3) layers needs 2x memory because of intermediate activation maps
  - Stacking small filters = better
    - can achieve equivalent receptive field
    - fewer params to train
    - less computation
    - needs more memory, but can fit into GPU memory
    - multiple levels of non-linearities (ReLU)
    - less overfitting
- All max pool layers are 2x2 stride 2
  - pooling is discarding info
  - non-overlapping stride follows intuition of downsampling
- conv layer following pool layer will have enough filters to double volume channel size
- Class of architectures, with 5 stages
- ~138 million params

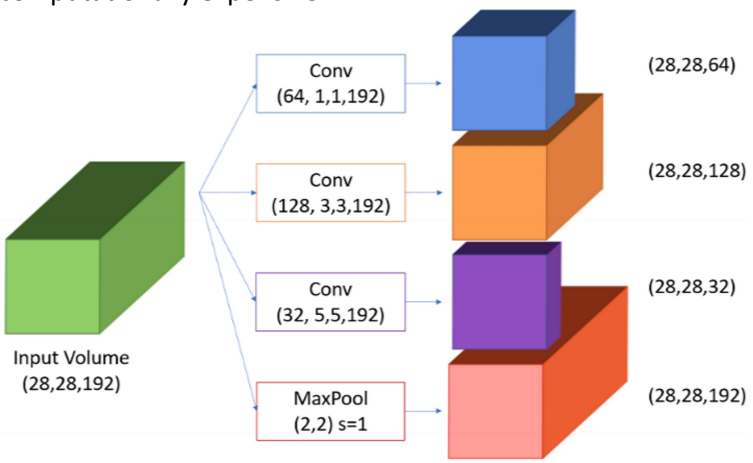
Layer	HyperParams	Output	# Params	Mflops
Input		(224,224,3)		
Conv1	K=64,f=(3,3),s=(1,1),p=same	(224,224,64)	1792	86.7
Conv2	K=64,f=(3,3),s=(1,1),p=same	(224,224,64)	36,928	1,849
Max Pool	f=(2,2),s=(2,2)	(112,112,64)	0	3.2
Conv3	K=128,f=(3,3),s=(1,1),p=same	(112,112,128)	73,856	924
Conv4	K=128,f=(3,3),s=(1,1),p=same	(112,112,128)	147,584	1,849
Max Pool	f=(2,2),s=(2,2)	(56,56,128)		1.6
Conv5	K=256,f=(3,3),s=(1,1),p=same	(56,56,256)	295,168	924
Conv6	K=256,f=(3,3),s=(1,1),p=same	(56,56,256)	590,080	1,849
Conv7	K=256,f=(3,3),s=(1,1),p=same	(56,56,256)	590,080	1,849
Max Pool	f=z(2,2),s=(2,2)	(28,28,256)	0	0.8

Layer	HyperParams	Output	# Params	Mflops
Conv8	K=512,f=(3,3),s=(1,1),p=same	(28,28,512)	1,180,160	924
Conv9	K=512,f=(3,3),s=(1,1),p=same	(28,28,512)	2,359,808	1,849
Conv10	K=512,f=(3,3),s=(1,1),p=same	(28,28,512)	2,359,808	1,849
Max Pool	f=(2,2),s=(2,2)	(14,14,512)	0	0.4
Conv11	K=512,f=(3,3),s=(1,1),p=same	(14,14,512)	2,359,808	462
Conv12	K=512,f=(3,3),s=(1,1),p=same	(14,14,512)	2,359,808	462
Conv13	K=512,f=(3,3),s=(1,1),p=same	(14,14,512)	2,359,808	462
Max Pool	f=(2,2),s=(2,2)	(7,7,512)	0	0.1
Flatten		(25088,)	0	
FC	n=4096	(4096,)	102,764,544	29
FC	n=4096	(4096,)	16,781,312	16.8
FC(softmax)	n=1000	(1000,)	4,097,000	4.1

- **SUMMARY**
  - uniform, straight forward
  - large num of params (~138 million)
  - VGG19 slightly better than VGG16
  - won localization challenge

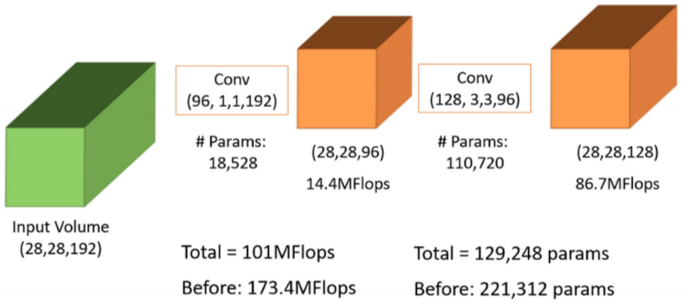
GoogLeNet (Inception)

- more efficient use of compute resources
  1. More parameters → more prone to overfitting → ok get more data → expensive
  2. Requires more computation → computation budget is finite → need to be more efficient with how you go bigger
- eliminates filter size as hyperparameter by "trying them all"
- has filters of different sizes in single layer
- computationally efficiency
  - computationally expensive:



  - Then stacking into a single volume: (28,28,416)
- Using 1x1 convolutions:
  - can reduce/grow channel dimension using conv layer with 1x1 filters
  - filters have implied 3rd dimension = to input volume's number of channels
  - for one 32 filters:
    - weighted sum across all feature maps at each spatial location

- Use 1x1 Convolution to shrink input volume first



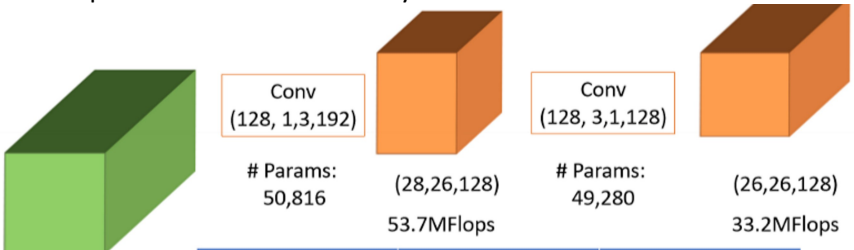
- **SUMMARY:**
  - has filters diff sizes in same layer
  - use 1x1 conv to improve conv efficiency
    - combining feature maps
  - doesn't hurt as long as not too aggressive

Global Average Pooling

- pooling operation "free"
- no parameters to optimize, less prone to overfitting
- more robust to spatial translation of final activations since looking over entire feature map

Inception V3 (ReCeption)

- 3 types of modules
  - same as GoogLeNet, but with 3x3 filters
- spatially separable conv
  - decompose a conv into two conv
  - can shrink channels with 1x1, then do 3x3
- can do spatial factorization into asymmetric conv

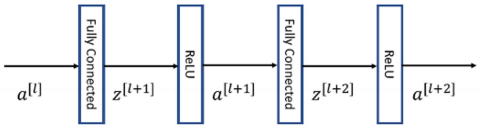


	# Params	MFLOPs
3x3	221k	173.4
1x1 → 3x3	129k	101
1x3 → 3x1	100k	86.1

ResNet

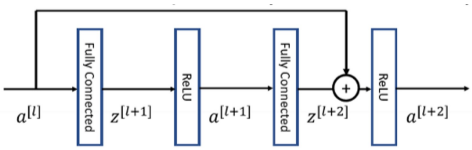
- can we get better results w more layers?
  - test error inc w more layers - not overfitting since training error also increased
  - DNN should be at least as good as shallow network
  - This is an optimization problem
    - hard to find identity func for layer
  - solution:

    - augment architecture to start with identify func, learn from there
- Residual block for FCL



$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}$$
$$a^{[l+1]} = ReLU(z^{[l+1]})$$
$$z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]}$$
$$a^{[l+2]} = ReLU(z^{[l+2]})$$

    - add shortcut connection:



$$a^{[l+2]} = ReLU(z^{[l+2]} + a^{[l]})$$
$$= ReLU(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

If $W^{[l+2]}$ and $b^{[l+2]}$ approach 0, then

$$= ReLU(a^{[l]}) = ReLU\left(ReLU(z^{[l]})\right) = ReLU(z^{[l]}) = a^{[l]}$$

    - residual identity func gives good baseline to improve
    - doesn't add learned params
    - doesn't increase computational complexity much
    - shortcut paths provide another path for backprop gradient flow
- 34 param layers
- no pooling layers
- stride = 2 in conv layer to shrink volumes
- use global avg pooling instead of FC layers at end

Plain Network Compared to VGG19

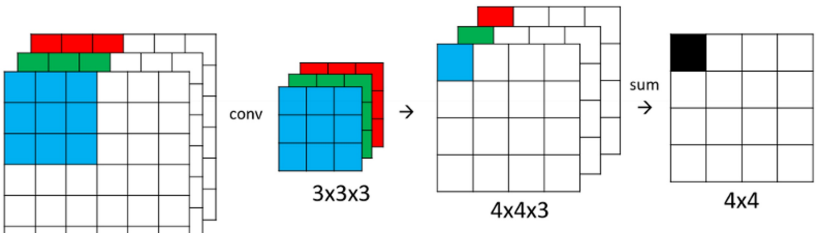
- 3.6 GFLOPs vs 19.6 GFLOPs
- 30-35M params vs 144M params

Memory Usage

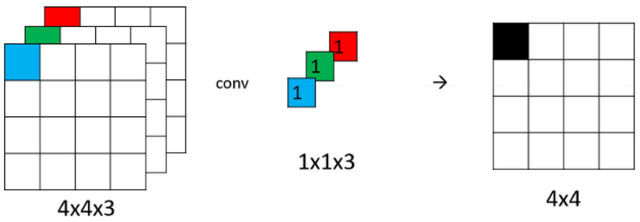
- Sources:
    - Activations (intermediate volumes and gradients)
    - Parameters (Param values and gradients)
    - Training Data (batch currently being processed)
- need to fit everything into GPU memory for training

MobileNet

- use 1x1 conv to create factorized conv to improve efficiency
- hyperparam to trade off accuracy/FLOPs/params
- Traditional Conv:

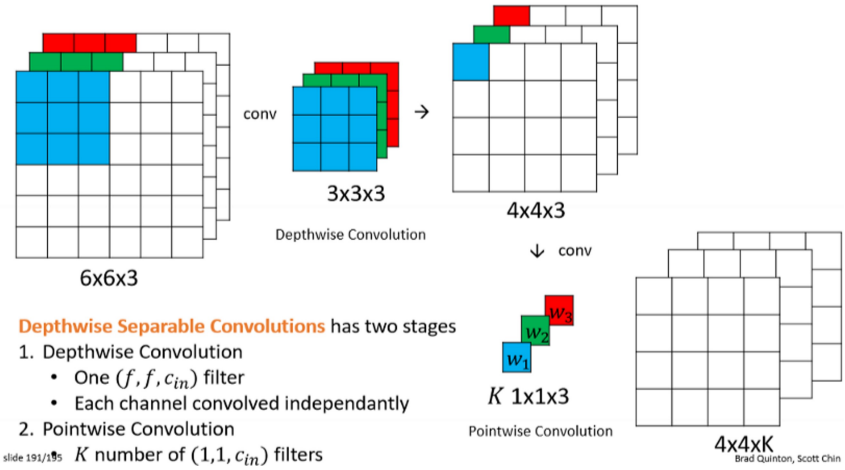


- Traditionally, filter produces a single map
- Can also think of it as two steps
    1. Channel independent convolution
    2. Summing across channels



- 1x1 conv with fixed filter values (1s)

- Use learned weights instead for 1x1 conv
  - **Depthwise Separable Convolutions**



EfficientNet

Keras

- offers all of the mainstream architectures
  - ResNet Imagenet weights on Keras