UNIVERSITY
OF OSLO

IN4310 Deep Architecture Evolution

Ali Ramezani-Kebrya

ali@uio.no

Key Learning Goals

Dropout Regularization

Batch and Layer Normalization

VGG, GoogLenNet, and Inception Module

ResNets and Residual Connections

Densenet

Key Learning Goals

Finetune thorugh loading weights from another pretrained model

Load weights bottom-up

Avoid training from scratch in general!

Finetun. can improve performance when training with small training sets

Architecture: VGG Simonyan & Zisserman, ICLR 2015

Very Deep Convlutional Networks for Large-scale Image Recognition

Table 1: ConvNet configurations (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: Number of parameters (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Architecture: VGG [Simonyan & Zisserman, ICLR15]

Very Deep Convlutional Networks for Large-scale Image Recognition

Stacking blocks of Convolution-ReLU-Pooling

Only 3×3-convolutions to achieve larger fields of view by stacking

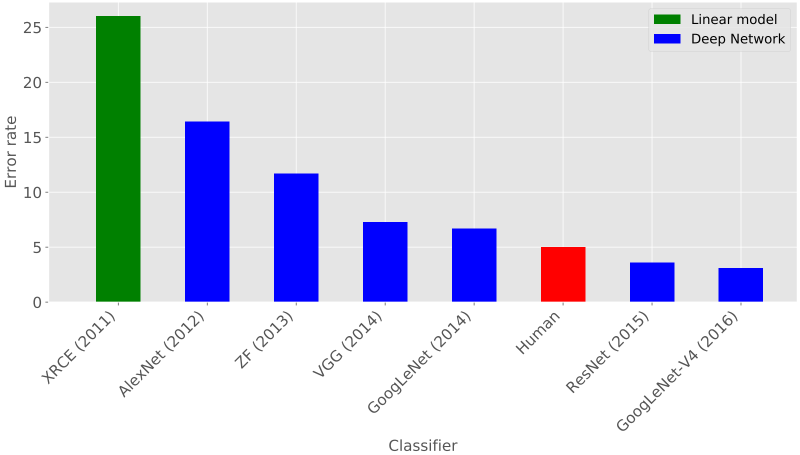
Very large number of parameters: 130 millions!

Fully connected layers contain a large portion of parameters

Dropout regularization for the first two FC layers for better generalization

Second place in ImageNet Challenge 2014 (classification track)

Why Did Neural Networks Explode around 2012?



Error rate on the ImageNet challenge

Outline

Dropout

GoogLeNet/Inception

ResNets and Residual Connections

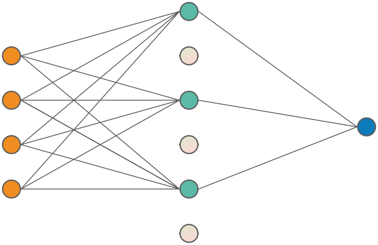
Batch Normalization

DenseNets

Finetuning

ViT

Dropout Regularization [Srivastava et al., JMLR14]



By dropping a neuron out, temporarily remove it from the network

Along with all its incoming and outgoing connections at training time

$$\mathbf{a}^{[l]} = g\left(\mathbf{W}^{[l]}(\mathbf{r}^{[l-1]} \odot \mathbf{a}^{[l-1]}) + \mathbf{b}^{[l]}\right)$$

$r_j^{[l-1]} \sim \text{Bernoulli}(p)$ independent Bernoulli random variables

$\odot$ is Hadamard (element-wise) product

Why Does Dropout Regularization Work?

Two dimensions of the feature map $\phi_i(\mathbf{x})$ and $\phi_j(\mathbf{x})$ may have some correlation which helps to classify sample $\mathbf{x}$ on training data

E.g, 95% of all the time on training data: $2\phi_i(\mathbf{x}) - \phi_j(\mathbf{x}) > 0$ for $y > 0$

But this correlation may not be not present in test data

Setting $\phi_i(\mathbf{x})$ or $\phi_j(\mathbf{x})$ to zero, prevents the algorithm from setting weights to use such correlation in a strong way

Why Does Dropout Regularization Work?

Noise via dropout reduces statistical correlations among features

The model cannot overemphasize on one single correlation

It has to rely on a mixture of several different correlations among features

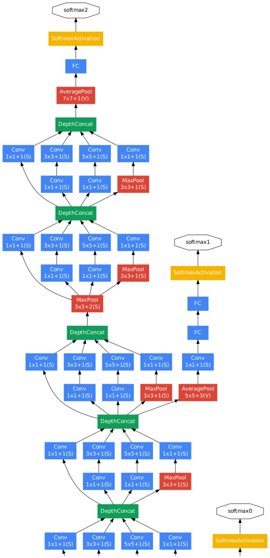
No neuron dominates the output

Fraction of neurons active at each iteration $1 - p$

At inference, use the entire network with scaling parameters down by $1 - p$

Outline

GoogLeNet v1/Inception v1 [Szegedy et al., CVPR15]



Idea 1: Auxiliary Classifiers at training time only

Encourage discrimination in the lower layers

Increase gradient signal getting propagated back

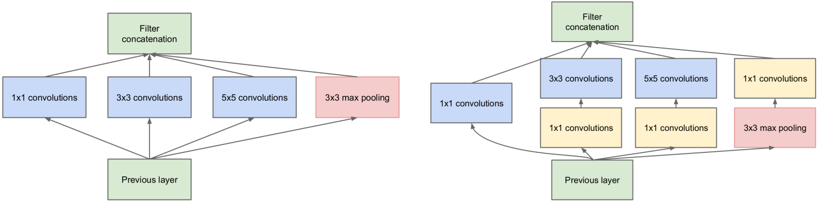
Provide additional regularization

Auxiliary output is a separate classification output

Cross entropy loss to estimate class probabilities

Overall Loss is weighted sum of main and aux losses

GoogLeNet v1/Inception v1 [Szegedy et al., CVPR15]



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

Idea 2: Inception Module

Convolution layers in parallel with different effective filter sizes

Visual information processed at various scales and then aggregated

Next layer can abstract features from different scales simultaneously

1×1 convolutions to reduce parameters with large # filters of last layer

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

Input 224×224 taking RGB color channels with mean subtraction

"$\#k \times k$ reduce" stands for the number of 1×1 filters in the reduction layer

GoogLeNet v1/Inception v1 [Szegedy et al., CVPR15]

Number of models	**Number of Crops**	**Cost**	**Top-5 error**	**compared to base**
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

Table 3: GoogLeNet classification performance break down

At inference: avg. over multiple classifiers and massive data augmentation

Inception v3 [Szegedy et al., CVPR16]

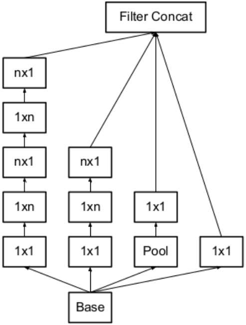


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

Using asymmetric convolutions $n \times 1$

A 3×1 convolution followed by a 1×3 convolution is equivalent to sliding a two layer network with the same receptive field as in a 3×3 convolution

Two-layer solution is 33% cheaper

In practice, this factorization does not work well on early layers

Works very well on medium layers

Each 5×5 conv. is replaced by two 3×3 convs.

Inception v3 [Szegedy et al., CVPR16]

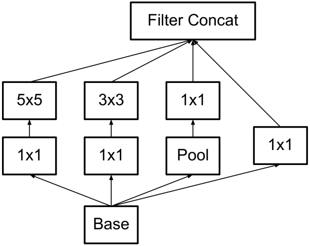


Figure 4. Original Inception module as described in [20].

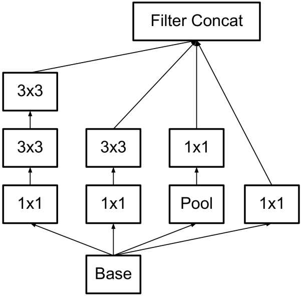


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

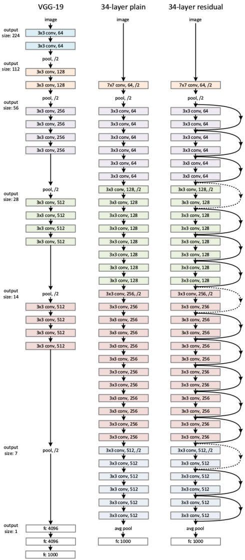
Each 5×5 convolution is replaced by two 3×3 convolutions

Reduction of filters geometric sizes comes at a cost of expressiveness

Enhanced space of variations the network can learn with batch normalization of activations

Outline

ResNets [He et al., CVPR16]



Idea 1: residual connections: shortcuts across layers

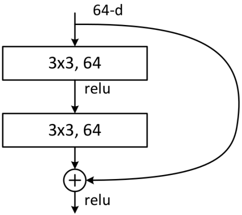
A linear mapping (instead of identity) whenever number of filters changes to match dimensions

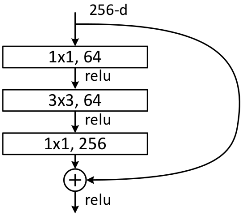
Idea 2: Batch normalization after every convolution

In top layers: half spatial size of feature maps and double number of filters

Rich set of templates at higher layers

Why Residual Connections?





Residual connection with 2 conv. blocks $f(\mathbf{x}) = \mathbf{x} + C_1(C_2(\mathbf{x}))$

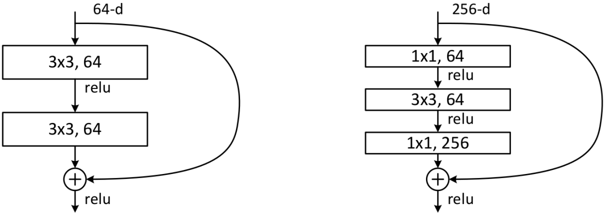
Why do residual connections work?

Backward pass: gradient flows through shortcuts, no vanishing gradient

Forward pass: if a convolution block is not useful, it will be bypassed

Learned function is at least as good as the one without conv. blocks

Why Residual Connections?



Identity+ optional non-linearity: convolutions across the parallel path can learn additionally non-linear functions

Never worse than identity: weights of convolution layers can be set to zero

Gradually learn a more complex representation layer by layer

Initial network can be one conv. layer and one FC layer with shortcuts

Filters can add non-linearities layer by layer

ResNets [He et al., CVPR16]: SoTA on ImageNet Challenge

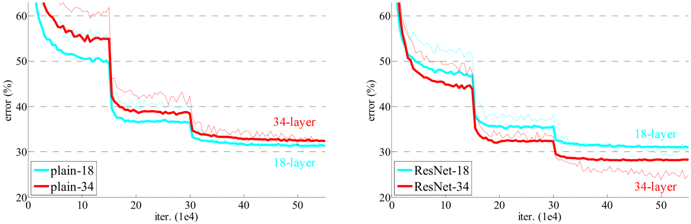


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Outline

Dropout

GoogLeNet/Inception

ResNets and Residual Connections

Batch Normalization

DenseNets

Finetuning

ViT

Batch Normalization [Ioffe and Szegedy, ICML15]

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.

Batchnorm at training time has 2 steps

Step 1: Normalize activations of a layer to have zero mean and stardard deviation one over elements of a minibatch

Step 2: Apply a simple affine transformation on the normalized output $\mathbf{y} = \gamma\hat{\mathbf{x}} + \beta$

Activations will have standard deviation γ and mean β with γ and β trainable

Batch Normalization [Ioffe and Szegedy, ICML15]

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.

For convolution layers:

Treat each neuron in the same channel in the same way

Compute mean and standard deviation for all elements in the feature map of one channel

Not only for one neuron and all samples in the minibatch

Reduce number of parameters

Batch Normalization [Ioffe and Szegedy, ICML15]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
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.

At training time: update running mean μ_{run} and running variance σ^2_{run}

Batchnorm at inference time has 2 steps:

Step 1: Normalize activations of a layer by the running mean and running variance learnt at training time:

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \mu_{\text{run}}}{\sqrt{\sigma^2_{\text{run}} + \epsilon}}$$

Step 2: apply $\mathbf{y} = a\hat{\mathbf{x}} + b$ with a,b the learnt rescaling parameters

To work well, it requires usually a batchsize of 8 at least, better 16 or 32 or more

Batch Normalization at Inference Time

Step 1: Normalize activations of a layer by the running mean and running variance learnt at training time:

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \mu_{\text{run}}}{\sqrt{\sigma^2_{\text{run}} + \epsilon}}$$

Step 2: apply $\mathbf{y} = a\hat{\mathbf{x}} + b$ with a,b the learnt rescaling parameters

This implies that the test sample activations would come from a very large batch with mean and variance statistics equal to the training data

Under this assumption, any synthetic minibatch of test samples would have mean b and std deviation a

Error source in coding: use `model.eval()` or `model.train(False)` at testing time for your neural network!

Why Does Batch Normalization Work?

Prevent small changes to the parameters from amplifying into larger and suboptimal changes in activations in gradients

Prevent the training from getting stuck in the saturated regimes

Make training more resilient to the parameter scale

Make propagation through a layer unaffected by the scale of its weights

Stabilize parameter growth as larger weights lead to smaller gradients

Improve generalization through regularization since a training example is seen in conjunction with other examples in the mini-batch

Group Normalization [Wu and He, ECCV18]

An alternative when cannot use large batchsizes

Convolution outputs an activation (b, c, h, w) for channel c in spatial dimensions h, w and batch b

Batchnorm: compute mean and std dev. for every channel over all spatial positions (h, w) and minibatch samples $\mu_c = \frac{1}{BHW} \sum_{b,h,w} f(b, c, h, w)$

Cannot compute over large minibatch?

Compute statistics over and over subset of filter channels in your feature map $\mu_{b,c} = \frac{1}{HW|G|} \sum_{h,w,c \in G} f(b, c, h, w)$

Depend on sample index b and channel index c

Layer Normalization [Ba et al., 16]

Layernorm: compute mean and std dev. over all hidden units in the same layer $\mu_b = \frac{1}{CHW} \sum_{c,h,w} f(b, c, h, w)$

No constraint on the size of a mini-batch

Normalizing across all features but for each of the inputs to a specific layer removes the dependence on batches

Layer normalization works well for sequential models such as transformers and recurrent neural networks (RNNs)

Outline

Densenets [Huang et al., CVPR17]

Resnets to the extreme: within a block of same feature map size ("dense block"), each layer contains the feature maps of each previous layer (of the same block) via concatenation of feature maps

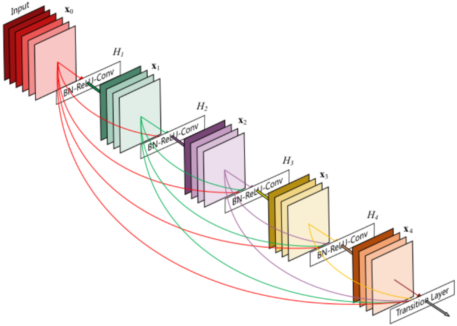


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Densenets [Huang et al., CVPR17]

The whole net looks like:

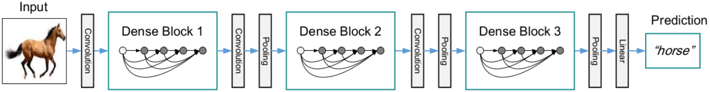


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

Densenets [Huang et al., CVPR17]

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

Growth rate: # newly added output channels in a convolution layer

Problem: within a block that starts with k_0 channels, at depth index l one has as inputs $k_0 + (l-1) \cdot \mathrm{growthrate}$ many channels

Densenet-B: 1×1 convolutions with BN and ReLU before each layer to control # of channels

Densenets [Huang et al., CVPR17]

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

Densenet-C: in transition layer: 1×1 conv halfs the number of channels

Commonly used: Densenet-BC

Densenets [Huang et al., CVPR17]

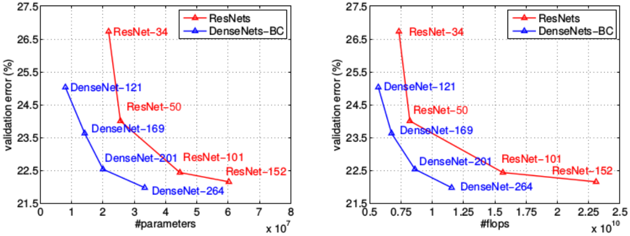


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

Commonly used: Densenet-BC

Good performance with small number of parameters

Outline

Finetuning

Do not train deep neural networks from scratch!

Always initialize the NN with weights from similar tasks trained on a very large dataset unless your data is in the order of hundred thousands and more.

Where to get pre-trained models and how? `torchvision.models`

`https://pytorch.org/docs/stable/torchvision/models.html`

How to Fine Tune?

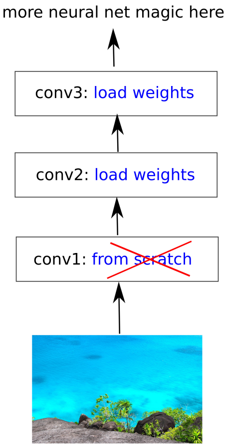
Take a deep network (densenet) and initialize it with weights from a 1000 class ImageNet task

Retrain it for 102 flowers classes

Why one can re-use weights from 1000 object classes that are mostly things and animals for flowers?

Low level filters likely will be very similar

It makes no sense to load weights for a layer, when one skips loading weights for any layer below



Why Does Fine Tuning Help?

Alias	Network	# Parameters	Top-1 Accuracy	Top-5 Accuracy	Origin
alexnet	AlexNet	61,100,840	0.5492	0.7803	Converted from pytorch vision
densenet121	DenseNet-121	8,062,504	0.7497	0.9225	Converted from pytorch vision
densenet161	DenseNet-161	28,900,936	0.7770	0.9380	Converted from pytorch vision
densenet169	DenseNet-169	14,307,880	0.7617	0.9317	Converted from pytorch vision
densenet201	DenseNet-201	20,242,984	0.7732	0.9362	Converted from pytorch vision
inceptionv3	Inception V3 299x299	23,869,000	0.7755	0.9364	Converted from pytorch vision
mobilenet0.25	MobileNet 0.25	475,544	0.5185	0.7608	Trained with script
mobilenet0.5	MobileNet 0.5	1,342,536	0.6307	0.8475	Trained with script
mobilenet0.75	MobileNet 0.75	2,601,976	0.6738	0.8782	Trained with script
mobilenet1.0	MobileNet 1.0	4,253,864	0.7105	0.9006	Trained with script
mobilenetv2_1.0	MobileNetV2 1.0	3,539,136	0.7192	0.9056	Trained with script
mobilenetv2_0.75	MobileNetV2 0.75	2,653,864	0.6961	0.8895	Trained with script
mobilenetv2_0.5	MobileNetV2 0.5	1,983,104	0.6449	0.8547	Trained with script
mobilenetv2_0.25	MobileNetV2 0.25	1,526,856	0.5074	0.7456	Trained with script

Consider training a neural network

ERM is non-convex problem

Find some local minima optimum

Deep NNs: high dimensionality of their parameters

Training from scratch leads to poor performance (bad local minima) with heuristic-based initialization

Why Does Fine Tuning Help?

Alias	Network	# Parameters	Top-1 Accuracy	Top-5 Accuracy	Origin
alexnet	AlexNet	61,100,840	0.5492	0.7803	Converted from pytorch vision
densenet121	DenseNet-121	8,062,504	0.7497	0.9225	Converted from pytorch vision
densenet161	DenseNet-161	28,900,936	0.7770	0.9380	Converted from pytorch vision
densenet169	DenseNet-169	14,307,880	0.7617	0.9317	Converted from pytorch vision
densenet201	DenseNet-201	20,242,984	0.7732	0.9362	Converted from pytorch vision
inceptionv3	Inception V3 299x299	23,869,000	0.7755	0.9364	Converted from pytorch vision
mobilenet0.25	MobileNet 0.25	475,544	0.5185	0.7608	Trained with script
mobilenet0.5	MobileNet 0.5	1,342,536	0.6307	0.8475	Trained with script
mobilenet0.75	MobileNet 0.75	2,601,976	0.6738	0.8782	Trained with script
mobilenet1.0	MobileNet 1.0	4,253,864	0.7105	0.9006	Trained with script
mobilenetv2_1.0	MobileNetV2 1.0	3,539,136	0.7192	0.9056	Trained with script
mobilenetv2_0.75	MobileNetV2 0.75	2,653,864	0.6961	0.8895	Trained with script
mobilenetv2_0.5	MobileNetV2 0.5	1,983,104	0.6449	0.8547	Trained with script
mobilenetv2_0.25	MobileNetV2 0.25	1,526,856	0.5074	0.7456	Trained with script

You can learn filters well only when you have enough training samples

Often hundreds of thousands

Non-convex ERM: local minima depends on initialization

When having only a few thousand samples it is best to start from a good initialization

Loading weights does that

Why Does Fine Tuning Help?

Finetuning preinitializes your network to some features that are good on another tasks

Empirical evidence: low-level features in deep networks learnt over wide and general tasks (e.g. Imagenet) can be reused for many other tasks, even with strange color distributions or geometrical tasks

Fine Tuning: Train Only Top Layer

Train only top layers:

Can be better for very small datasets

For larger datasets, training all layers can be better. Check validation data

Without data augmentation, bottom features can be precomputed for a speed up (usually data augmentation improves test error! ... trade-off speed vs performance)

Outline

State of the Art (SoTA)?

In the next lectures:

Much available compute? Vision Transformers [Dosovitskiy et al., ICLR21]

References

http://d2l.ai/chapter_convolutional-modern/index.html