New recognition architectures
The VGG pattern

• Every convolution is 3x3, padded by 1
• Every convolution followed by ReLU
• ConvNet is divided into “stages”
  • Layers within a stage: no subsampling
  • Subsampling by 2 at the end of each stage
• Layers within stage have same number of channels
• Every subsampling $\rightarrow$ double the number of channels
Challenges in training: exploding / vanishing gradients

• Vanishing / exploding gradients

\[
\frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_{i+1}}{\partial z_i}
\]

• If each term is (much) greater than 1 \(\rightarrow\) explosion of gradients
• If each term is (much) less than 1 \(\rightarrow\) vanishing gradients
Challenges in training: dependence on init
Solutions

• Careful init

• Batch normalization

• Residual connections
Careful initialization

• Key idea: want variance to remain approx. constant
  • Variance increases in backward pass => exploding gradient
  • Variance decreases in backward pass => vanishing gradient

• “MSRA initialization”
  • weights = Gaussian with 0 mean and variance = $2/(k^2*d)$
How do we make better neural network architectures?

• Better optimization
• Cheaper compute requirements
• New primitives
Residual connections

• In general, gradients tend to vanish
• Key idea: allow gradients to flow unimpeded

\[ z_{i+1} = f_{i+1}(z_i, w_{i+1}) \quad \frac{\partial z_{i+1}}{\partial z_i} = \frac{\partial f_{i+1}(z_i, w_{i+1})}{\partial z_i} \]

\[ \frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_{i+1}}{\partial z_i} \]

Residual connections

• In general, gradients tend to vanish
• Key idea: allow gradients to flow unimpeded

\[ z_{i+1} = g_{i+1}(z_i, w_{i+1}) + z_i \]

\[ \frac{\partial z_{i+1}}{\partial z_i} = \frac{\partial g_{i+1}(z_i, w_{i+1})}{\partial z_i} + I \]

\[ \frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \ldots \frac{\partial z_{i+1}}{\partial z_i} \]
Residual connections

• Assumes all $z_i$ have the same size
• True within a stage
• Across stages?
  • Doubling of feature channels
  • Subsampling
• Increase channels by 1x1 convolution
• Decrease spatial resolution by subsampling

$$z_{i+1} = g_{i+1}(z_i, w_{i+1}) + \text{subsample}(Wz_i)$$
A residual block

• Instead of single layers, have residual connections over block
Better optimization - Batch normalization

• Key idea: normalize so that each layer output has zero mean and unit variance
  • Compute mean and variance for each channel
  • Aggregate over batch
  • Subtract mean, divide by std

• Generally makes optimization landscape smoother

• Need to reconcile train and test
  • No "batches" during test
  • After training, compute means and variances on train set and store

Other forms of normalization


Analyzing computational complexity

• What is the computational complexity of a single convolutional layer?
  • \( h \times w \times c \) input and output
  • \( k \times k \) kernel

• Space:
  • Input/output: \( hwc \)
  • Filters: \( k^2c^2 \)

• Time (Flops): \( hwk^2c^2 \)
Reducing computational complexity

• ...while maintaining accuracy?
• Multiple ways:
  • Make architecture *a priori* cheaper
  • Make *weights* and *operations* cheaper
  • Make inference adaptive
Better compute-cost tradeoffs: bottleneck layers
Bottleneck blocks

• Problem: When channels increases, 3x3 convolutions introduce many parameters
  • $3 \times 3 \times c^2$

• Key idea: use 1x1 to project to lower dimensionality, do convolution, then come back
  • $c \times d + 3 \times 3 \times d^2 + d \times c$
Other architectural changes

- Biggest memory consumption: large feature maps
Other architectural changes

- Biggest memory consumption: large feature maps
- Simple solution: reduce resolution early
The ResNet pattern

• Decrease resolution substantially in first layer
  • Reduces memory consumption due to intermediate outputs

• Divide into stages
  • maintain resolution, channels in each stage
  • halve resolution, double channels between stages

• Divide each stage into residual blocks

• At the end, compute average value of each channel to feed linear classifier
The ResNet Pattern
Putting it all together - Residual networks

Challenge winner's accuracy

2010 2011 2012 2013 2014 2015

0 5 10 15 20 25 30

0 50 100 150 200
Better compute-cost tradeoffs: Other kinds of convolution
Better compute-cost tradeoffs: Other kinds of convolution

Depth-wise convolution:
Better compute-cost tradeoffs: Other kinds of convolution

Grouped convolution:

$\text{Convolution}$
Cheaper convolutional blocks

• Standard convolution:
  • Each filter operates on all channels
  • Single $k \times k$ filter operating on $c$ channels producing one output channel: $k^2c$ parameters, cost
  • $c$ such filters: $k^2c^2$ parameters, cost

• Depthwise separable convolution
  • Each filter operates on a single channel
  • $c$ filters operating on $c$ channels: $k^2c$ parameters, cost
  • But each channel is independently processed
  • Add a 1x1 convolution at the end with cost $c^2 : k^2c + c^2$ parameters
Cheaper convolutional blocks

• Depthwise separable convolutions are specific instance of more general idea: *grouped convolutions*
• Grouped convolutions in original AlexNet network
• Grouped convolution:
  • Divide input channels into $g$ groups
  • Apply convolutional layers on each group independently
  • Concatenate
Grouped and depth-wise convolutions

Table 4. Depthwise Separable vs Full Convolution MobileNet

<table>
<thead>
<tr>
<th>Model</th>
<th>ImageNet Accuracy</th>
<th>Million Multi-Adds</th>
<th>Million Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv MobileNet</td>
<td>71.7%</td>
<td>4866</td>
<td>29.3</td>
</tr>
<tr>
<td>MobileNet</td>
<td>70.6%</td>
<td>569</td>
<td>4.2</td>
</tr>
</tbody>
</table>


Other architectural changes

- Biggest memory consumption: large feature maps
- Simple solution (ResNet):
  - Reduce resolution drastically (/4) early
- More sophisticated changes: Inverted residuals (MobileNet v2)

Other kinds of connections

• DenseNets
  • Replace addition of residuals with concatenation
  • Alternative to solving vanishing gradient problem
  • Should *increase* number of parameters, but *decreases* them
  • Better re-use of features

Transformers
A brief dive into language generation

• Consider the machine translation problem

He went to the market

वह बाजार गया
Attention (Transformers)

• Comes from the NLP community
• Is an approach for processing sets

Attention (Transformers)

• Comes from the NLP community
• Is an approach for processing sets
Attention (Transformers)

- Comes from the NLP community
- Is an approach for processing sets
Attention (Transformers)

Input

Queries

Keys

Values

Output

Attention
Attention (Transformers)

Transformers on images

Vision Transformer (ViT)

Transformer Encoder

Class
Bird
Ball
Car
...

MLP Head

Transformer Encoder

Linear Projection of Flattened Patches

Patch + Position Embedding
* Extra learnable [class] embedding

0* 1 2 3 4 5 6 7 8 9

Transformer Encoder

MLP

Norm

Multi-Head Attention

Norm

Embedded Patches
Transformers + Convolution

Original ViT (baseline, termed ViT_p):
- Sensitive to lr and wd choice
- Converges slowly
- Works with AdamW, but not SGD
- Underperforms sota CNNs on ImageNet

Ours (termed ViT_C, same runtime):
- Robust to lr and wd choice
- Converges quickly
- Works with AdamW, and also SGD
- Outperforms sota CNNs on ImageNet
Transfer Learning
Transfer learning with neural networks

Trained feature extractor \( \phi \)
Transfer learning with convolutional networks

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Non-Convnet Method</th>
<th>Non-Convnet perf</th>
<th>Pretrained convnet + classifier</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech 101</td>
<td>MKL</td>
<td>84.3</td>
<td>87.7</td>
<td>+3.4</td>
</tr>
<tr>
<td>VOC 2007</td>
<td>SIFT+FK</td>
<td>61.7</td>
<td>79.7</td>
<td>+18</td>
</tr>
<tr>
<td>CUB 200</td>
<td>SIFT+FK</td>
<td>18.8</td>
<td>61.0</td>
<td>+42.2</td>
</tr>
<tr>
<td>Aircraft</td>
<td>SIFT+FK</td>
<td>61.0</td>
<td>45.0</td>
<td>-16</td>
</tr>
<tr>
<td>Cars</td>
<td>SIFT+FK</td>
<td>59.2</td>
<td>36.5</td>
<td>-22.7</td>
</tr>
</tbody>
</table>
Why transfer learning?

• Availability of training data

• Computational cost

• Ability to pre-compute feature vectors and use for multiple tasks

• *Con: NO end-to-end learning*
Finetuning
Finetuning

Initialize with pre-trained, then train with low learning rate
## Finetuning

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Non-Convnet Method</th>
<th>Non-Convnet perf</th>
<th>Pretrained convnet + classifier</th>
<th>Finetuned convnet</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech 101</td>
<td>MKL</td>
<td>84.3</td>
<td>87.7</td>
<td>88.4</td>
<td>+4.1</td>
</tr>
<tr>
<td>VOC 2007</td>
<td>SIFT+FK</td>
<td>61.7</td>
<td>79.7</td>
<td>82.4</td>
<td>+20.7</td>
</tr>
<tr>
<td>CUB 200</td>
<td>SIFT+FK</td>
<td>18.8</td>
<td>61.0</td>
<td>70.4</td>
<td>+51.6</td>
</tr>
<tr>
<td>Aircraft</td>
<td>SIFT+FK</td>
<td>61.0</td>
<td>45.0</td>
<td>74.1</td>
<td>+13.1</td>
</tr>
<tr>
<td>Cars</td>
<td>SIFT+FK</td>
<td>59.2</td>
<td>36.5</td>
<td>79.8</td>
<td>+20.6</td>
</tr>
</tbody>
</table>
What impacts transfer accuracies?

• Relationship between pre-training and target task?
• Unclear: sometimes transfer works across very different domains
  • E.g., ImageNet -> Satellite images
• Very limited work on understanding this
What impacts transfer accuracy?

- Size of the pre-training dataset
- Size of the model
- Bigger is better


Concerns about big transfer

• Opaque datasets?
• Uncurated datasets?
• Bias in the datasets?
  • Do biased datasets affect transfer? Turns out yes.
Neural network visualization
Why?

• “Interpretability”:  
  • Want to know what the neural network is basing its decision on.
• In general, any way of visualization approximate
Receptive field

(2x-3, 2y-3) ...
(2x+3, 2y+3)

(2x-2, 2y-2) ...
(2x+2, 2y+2)

(x-1, y-1) ...
(x+1, y+1)
Visualizing the receptive field

- Pick a layer
- Pick a channel
- Pick a particular location in feature map
- Draw out the corresponding receptive field in the image
Visualizing what activates channels

• Pick a layer
• Pick a channel
• Identify images and locations that give the highest value
Visualizing what patches are important

- Block part of image with a grey square and record class score
- Move gray square over image to get 2D array of scores
- Result is heatmap with low score for important patches
Class activation maps

Grad-CAM

• Can also look at gradient of class scores w.r.t. image pixels
• Indicates which pixels will cause highest change in scores
• By itself not useful because it tends to highlight edges or corners
• But can be combined with CAM

Other ways of reducing computation
Adaptive inference

• Some examples are harder than others
• Should be able to use different amounts of computation for different examples
• Version 1: skip some residuals


Adaptive inference

• Some examples are harder than others
• Should be able to use different amounts of computation for different examples
• Version 1: skip some residuals


Adaptive inference

- Some examples are harder than others
- Should be able to use different amounts of computation for different examples
- Version 2: reduce resolution at different rates

Huang, Gao, et al. "Multi-scale dense networks for resource efficient image classification." ICLR 2018
Compressing model weights

• All of model storage: filters
• Flops also scale with non-zero entries in filters (in principle)
• Compress filters
  • Sparsify them
  • Represent them with fewer bits
Pruning network connections

• Simple approach: prune weights that are below a threshold
• Retrain rest of the weights
• Repeat

Filter quantization

- Two questions:
  - How do we quantize?
  - Quantization $\rightarrow$ discrete values. How do we optimize?

- Example 1: *cluster*
  - Weights $\rightarrow$ indices into dictionary
  - Update dictionary elements as parameters.

Filter quantization

• Two questions:
  • How do we quantize?
  • Quantization $\rightarrow$ discrete values. How do we optimize?

• Example 2: *binarize/ternarize*
  • Weights $\rightarrow$ binary/ternary, + real-valued scale
  • Parameter updates happen in real space
