



High Level Computer Vision

Hierarchical Transformer Architectures & Advanced Optimization

@ June 7, 2023

Bernt Schiele

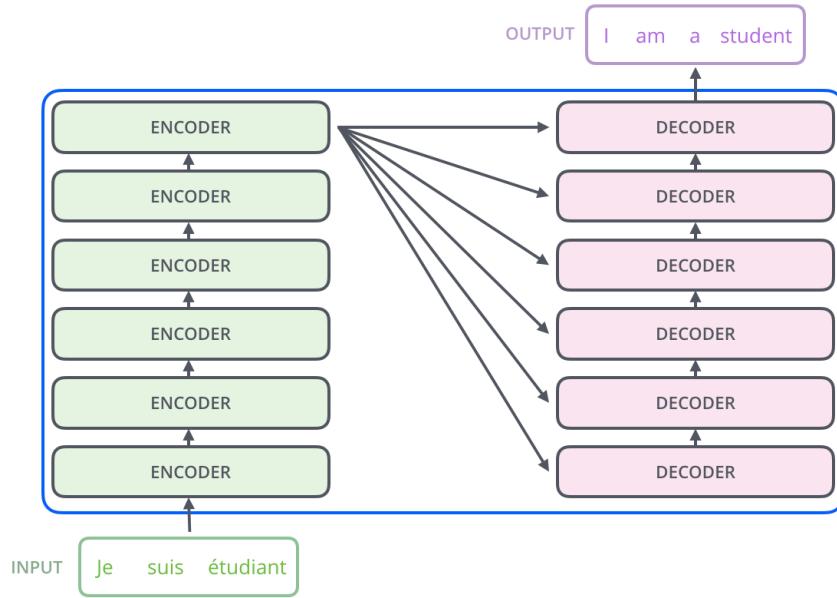
cms.sic.saarland/hlcvss23/

Max Planck Institute for Informatics & Saarland University,
Saarland Informatics Campus Saarbrücken

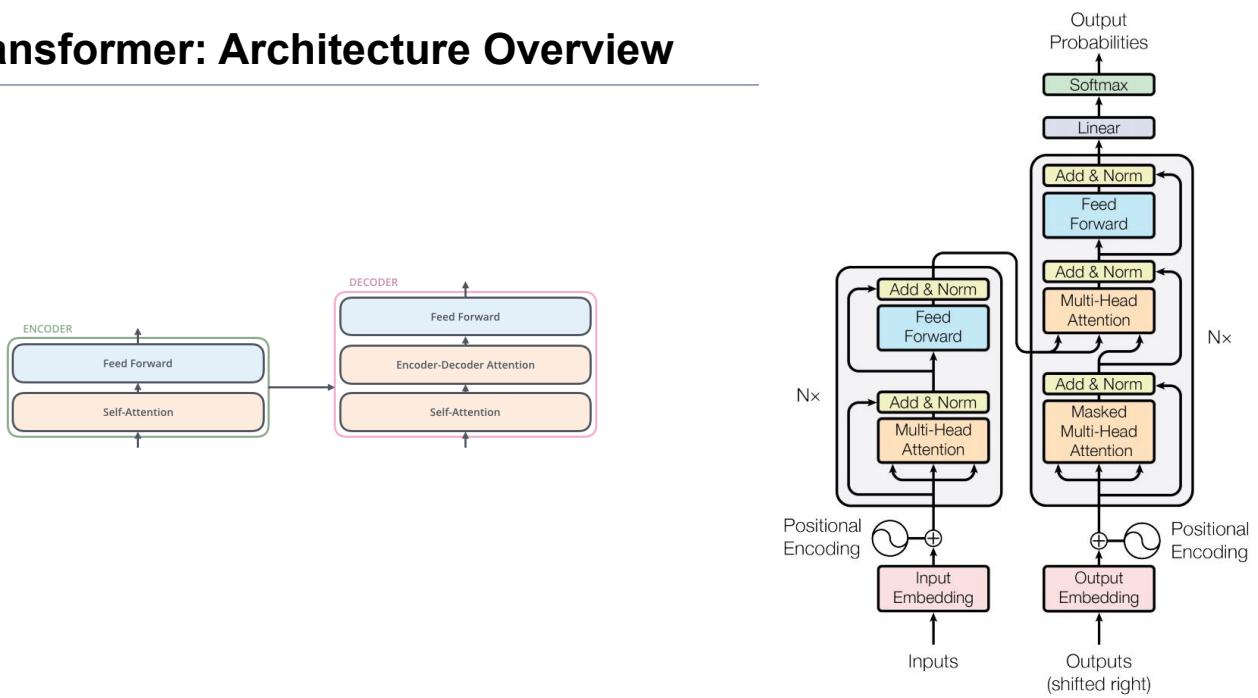
Overview of Today's Lecture

- Recap of Vision Transformer (ViT) for Image Classification
- Hierarchical Transformer Architectures
 - Swin Transformer — Hierarchical Vision Transformer using Shifted Windows
@ ICCV 2021 — <https://arxiv.org/abs/2103.14030>
 - SegFormer — Simple and Efficient Design for Semantic Segmentation with Transformers
@ NeurIPS 2021 — <https://arxiv.org/abs/2105.15203>
- Beyond Vanilla Gradient Descent
 - Adam Optimizer (and its components)
 - Second Order Optimization

Transformer: Architecture Overview

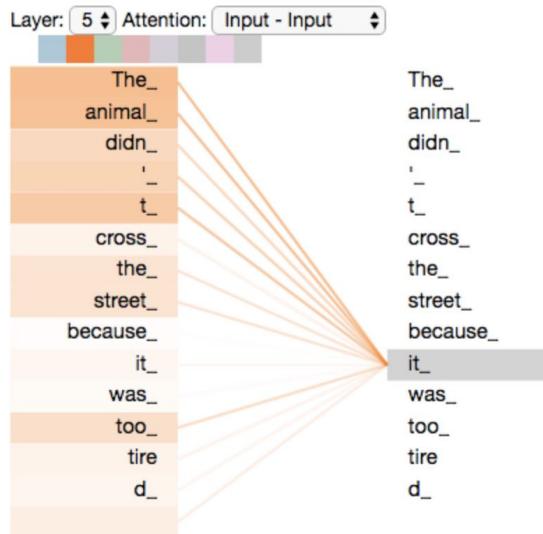


Transformer: Architecture Overview



Attention is all you need. Vaswani et al. NeurIPS 2017

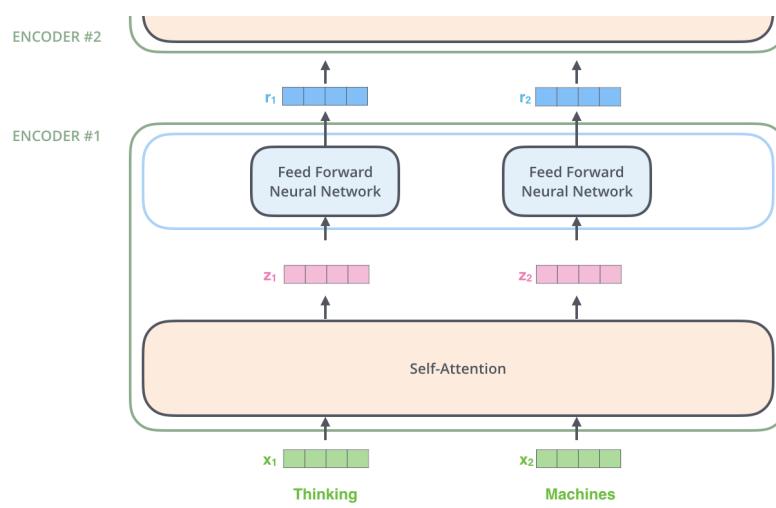
What is Self-Attention?



Attention is all you need. Vaswani et al. NeurIPS 2017



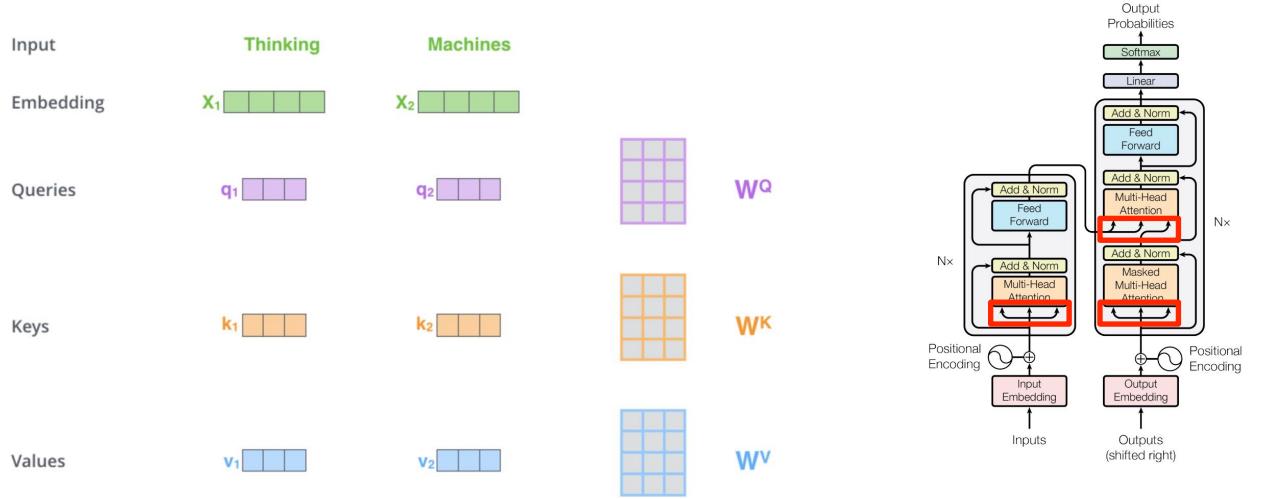
Encoder — Consumes a Set of (Word) Tokens



Attention is all you need. Vaswani et al. NeurIPS 2017



Self-Attention: Query, Key and Value



Attention is all you need. Vaswani et al. NeurIPS 2017

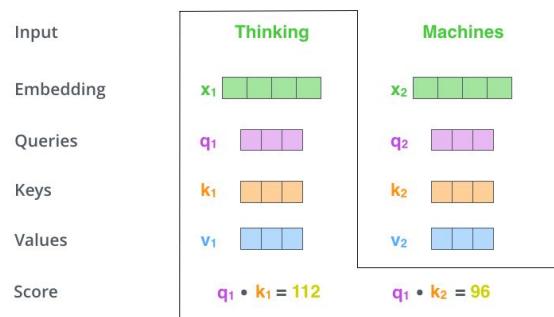
slide credit: Yongqin Xian



High Level Computer Vision | Bernt Schiele

7

Self-Attention: Query, Key and Value



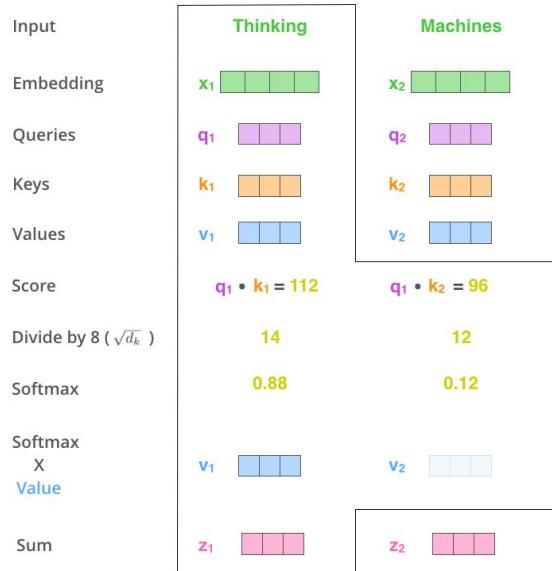
slide credit: Yongqin Xian



High Level Computer Vision | Bernt Schiele

8

Self-Attention: Query, Key and Value



slide credit: Yongqin Xian



Self-Attention: Output

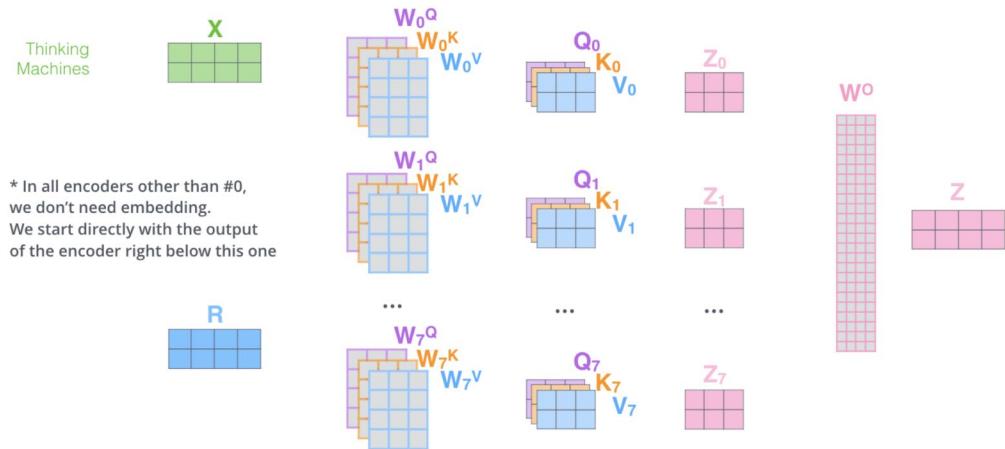
$$\text{softmax} \left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V} = \mathbf{Z}$$

The diagram illustrates the computation of the attention output Z . It shows the input matrix Q , the transpose of the key matrix K^T , and the value matrix V . The attention scores are computed as the dot product of Q and K^T divided by $\sqrt{d_k}$, and then passed through a softmax function. The resulting attention weights are multiplied with V to produce the final output Z .



Mult-Head Self-Attention

- 1) This is our input sentence*
- 2) We embed each word*
- 3) Split into 8 heads. We multiply X or R with weight matrices
- 4) Calculate attention using the resulting $Q/K/V$ matrices
- 5) Concatenate the resulting Z matrices, then multiply with weight matrix W^o to produce the output of the layer



Attention is all you need. Vaswani et al. NeurIPS 2017

slide credit: Yongqin Xian

High Level Computer Vision | Bernt Schiele

11

Vision Transformer (ViT)

- What are sensible tokens for images?



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Dosovitskiy et al. ICLR 2021

High Level Computer Vision | Bernt Schiele

12

Vision Transformer (ViT)

- What are sensible tokens for images?
 - ▶ Image Patches...
 - ▶ ... here: each image patch of size 16x16

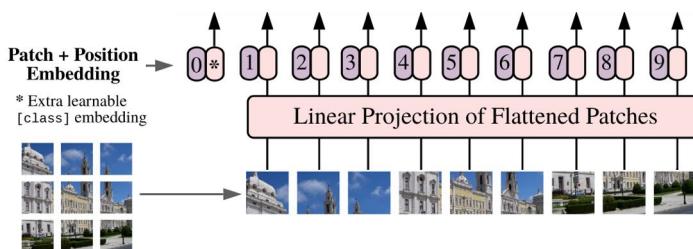


An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Dosovitskiy et al. ICLR 2021



Vision Transformer (ViT)

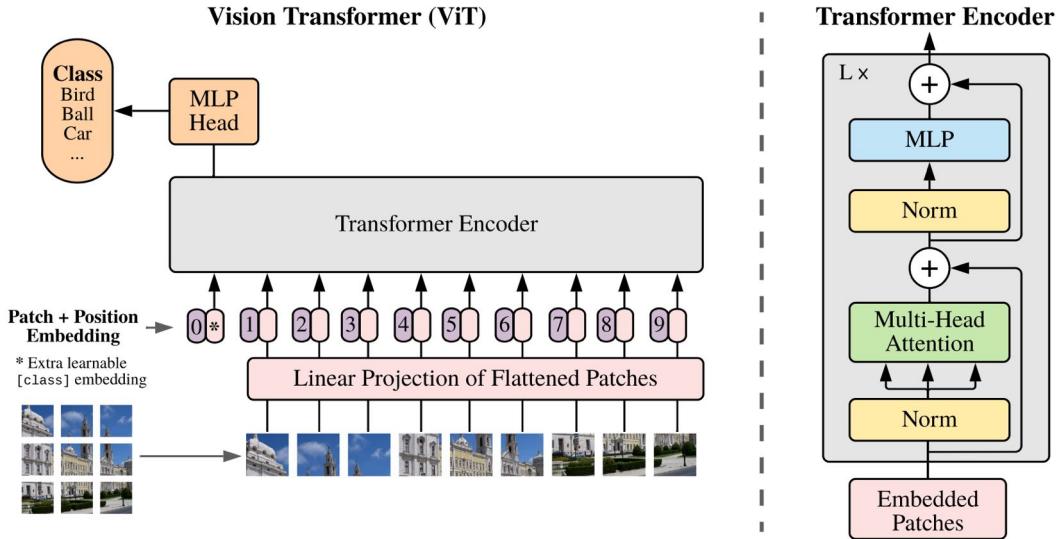
- Positional encoding
 - ▶ learnable embedding for each patch position (sometimes also fixed positional encoding)
- Additional token for “class embedding”
 - ▶ learnable, its output embedding used for global image representation (and image classification — see next slide)



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Dosovitskiy et al. ICLR 2021



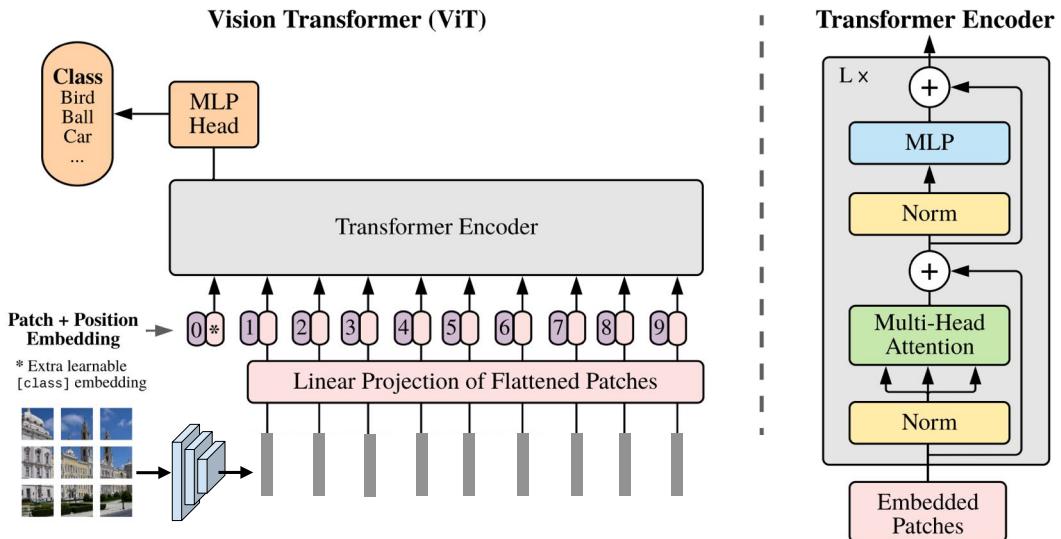
Vision Transformer (ViT)



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Dosovitskiy et al. ICLR 2021



Hybrid ViT Architecture (with CNN-embedding of image patches)



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Dosovitskiy et al. ICLR 2021



Overview of Today's Lecture

- Recap of Vision Transformer (ViT) for Image Classification
- Hierarchical Transformer Architectures
 - Swin Transformer — Hierarchical Vision Transformer using Shifted Windows
@ ICCV 2021 — <https://arxiv.org/abs/2103.14030>
 - SegFormer — Simple and Efficient Design for Semantic Segmentation with Transformers
@ NeurIPS 2021 — <https://arxiv.org/abs/2105.15203>
- Beyond Vanilla Gradient Descent
 - Adam Optimizer (and its components)
 - Second Order Optimization



Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Ze Liu^{1,2†*} Yutong Lin^{1,3†*} Yue Cao^{1*} Han Hu^{1**‡} Yixuan Wei^{1,4†}
Zheng Zhang¹ Stephen Lin¹ Baining Guo¹

¹Microsoft Research Asia ²University of Science and Technology of China

³Xian Jiaotong University ⁴Tsinghua University

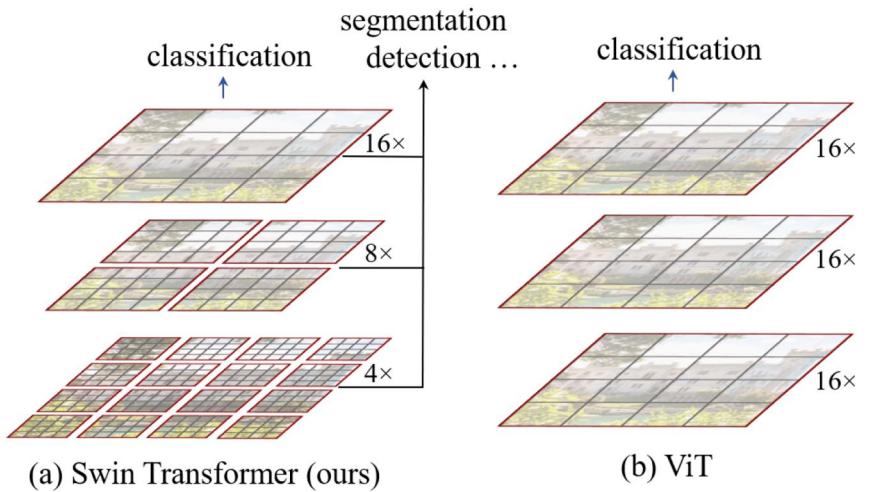
{v-zeliu1, v-yutlin, yuecao, hanhu, v-yixwe, zhez, stevelin, bainguo}@microsoft.com

ICCV 2021 — Best Paper Award (Marr Prize)



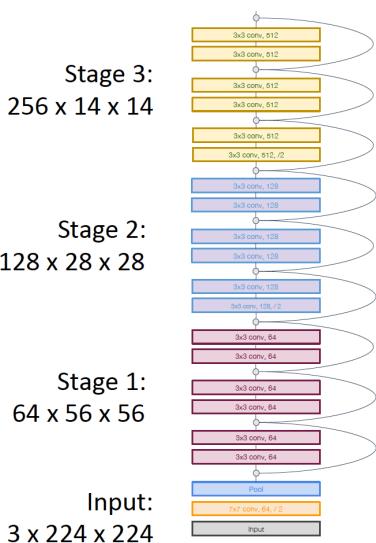
Swin Transformer vs. Standard ViT (Vision Transformer)

- Swin Transformer
 - hierarchical: token size increases from 4x4 to 16x16 pixels
 - architecture can be used for
 - image classification, but also
 - semantic segmentation and
 - object detection
 - instance segmentation
 - computationally more efficient than ViT



slide credit: Justin Johnson

ViT vs CNN

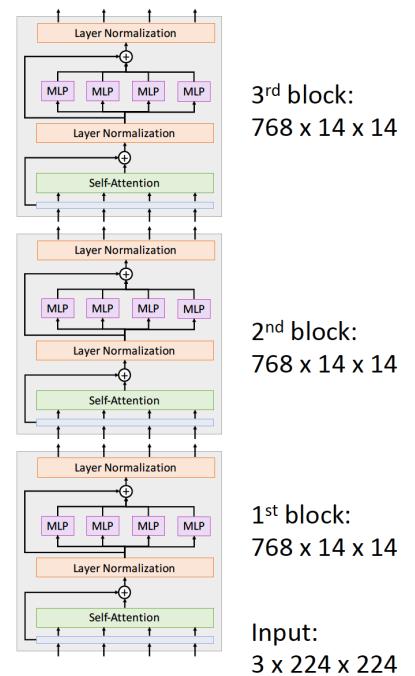


In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

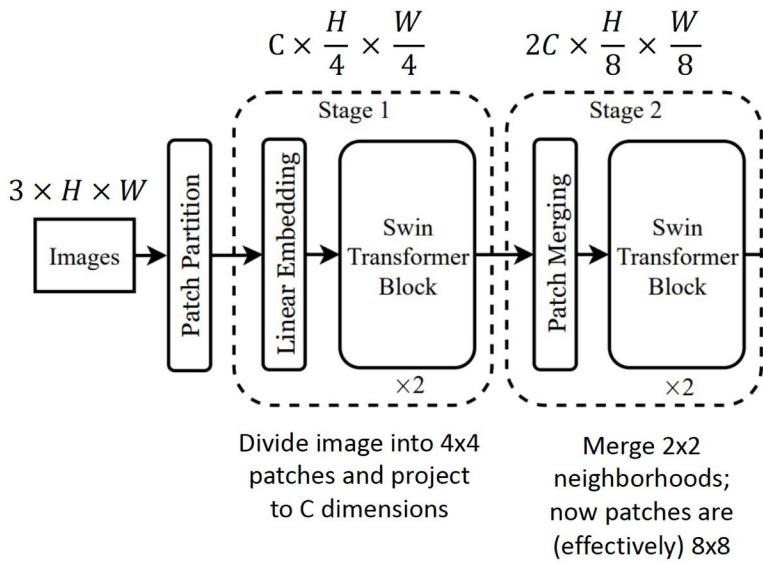
Useful since objects in images can occur at various scales

In a ViT, all blocks have same resolution and number of channels (Isotropic architecture)

Can we build a **hierarchical** ViT model?



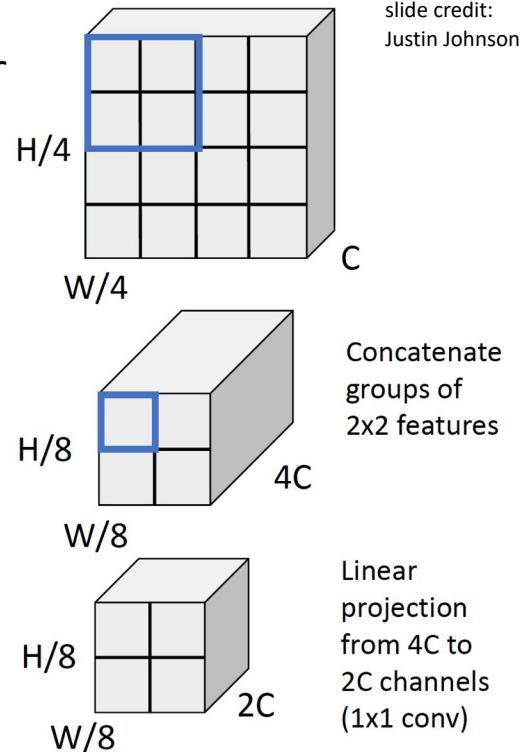
Hierarchical ViT: Swin Transformer



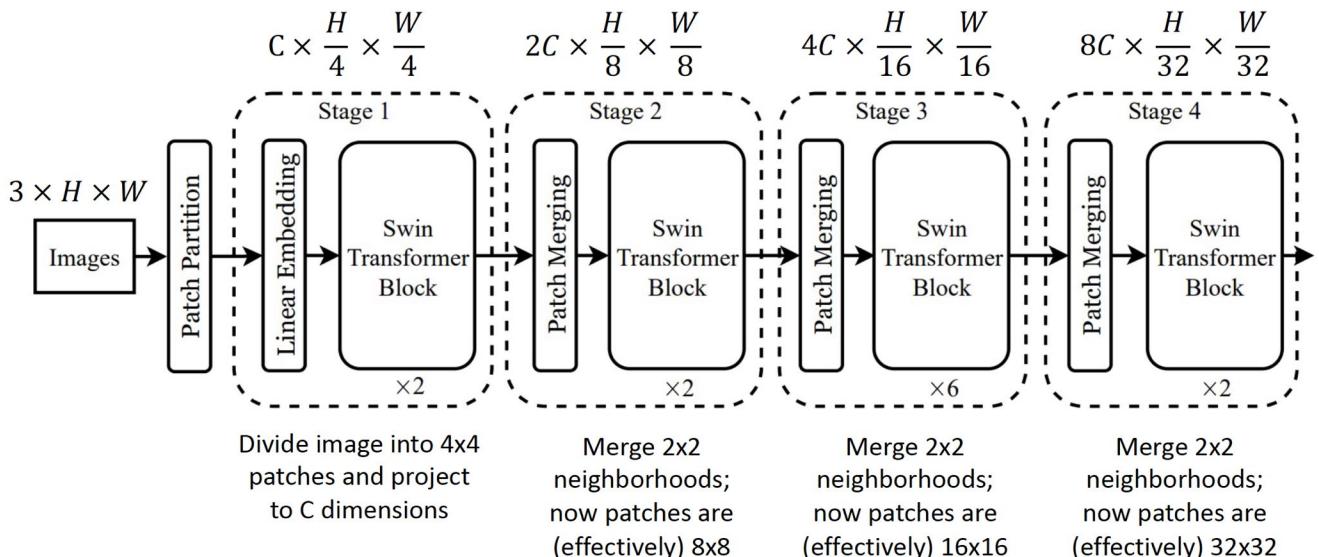
Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

High Level Computer Vision | Bernt Schiele

21



Hierarchical ViT: Swin Transformer



Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

slide credit: Justin Johnson

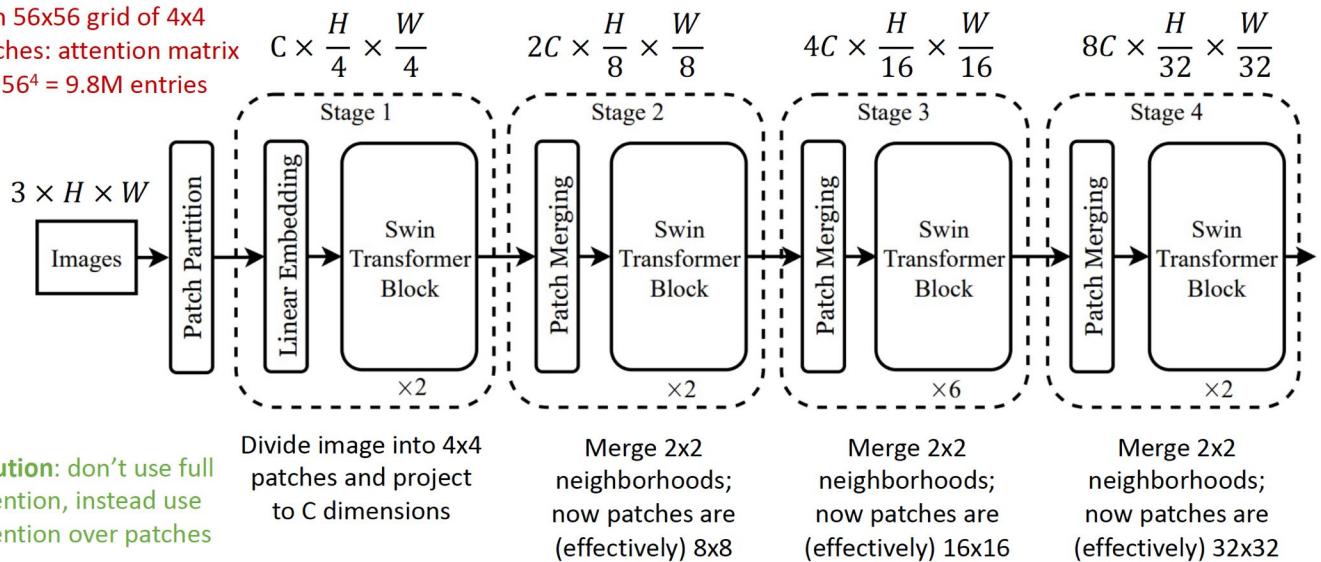
High Level Computer Vision | Bernt Schiele

22

Hierarchical ViT: Swin Transformer

Problem: 224x224 image

with 56x56 grid of 4x4 patches: attention matrix has $56^4 = 9.8M$ entries



Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

slide credit: Justin Johnson



High Level Computer Vision | Bernt Schiele

23

Swin Transformer: Window Attention

With $H \times W$ grid of **tokens**, each attention matrix is H^2W^2 – **quadratic** in image size

Rather than allowing each **token** to attend to all other tokens, instead divide into **windows** of $M \times M$ tokens (here $M=4$); only compute attention within each window

Total size of all attention matrices is now: $M^4(H/M)(W/M) = M^2HW$

Linear in image size for fixed M !
Swin uses $M=7$ throughout the network

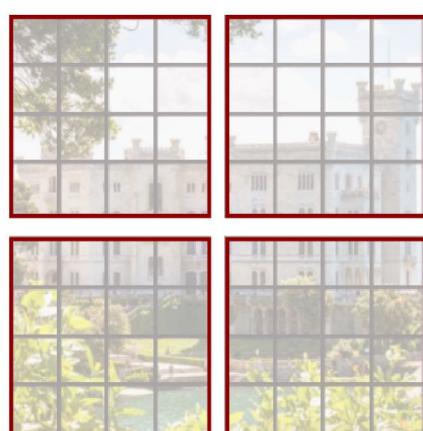
Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021

slide credit: Justin Johnson



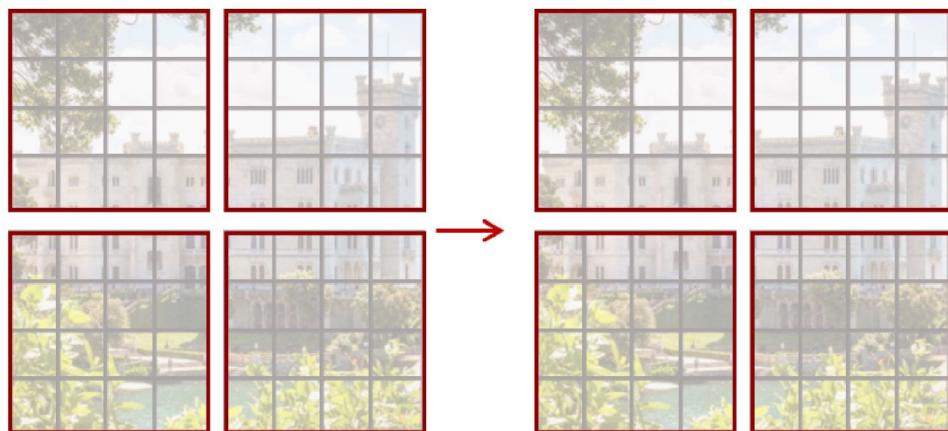
High Level Computer Vision | Bernt Schiele

24



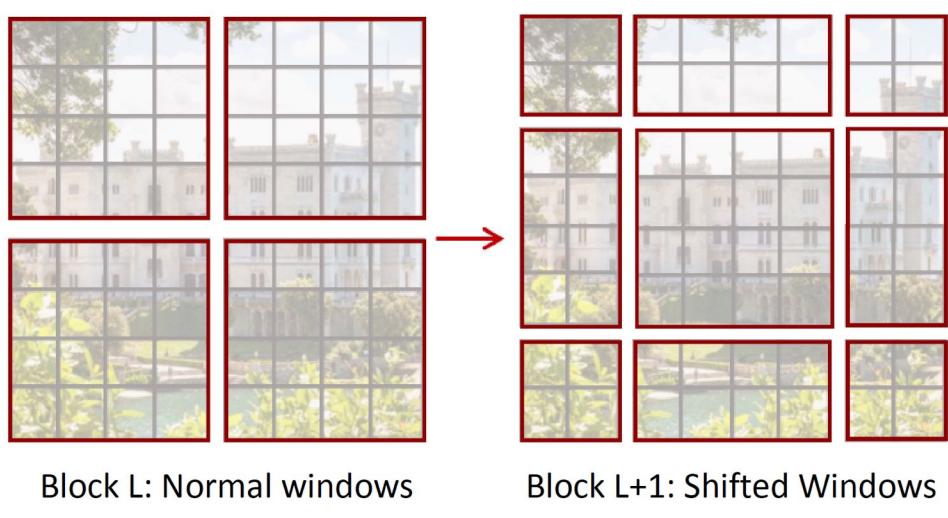
Swin Transformer: Window Attention

Problem: tokens only interact with other tokens within the same window; no communication across windows



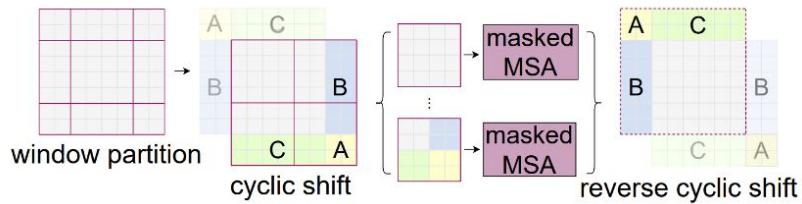
Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks



Efficient batch computation for shifted configuration

- First idea
 - ▶ add padding... works, but increased computation
- Their idea: cyclic-shift:



Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks

Detail: Relative Positional Bias

ViT adds positional embedding to input tokens, encodes *absolute position* of each token in the image

Swin does not use positional embeddings, instead encodes *relative position* between patches when computing attention:

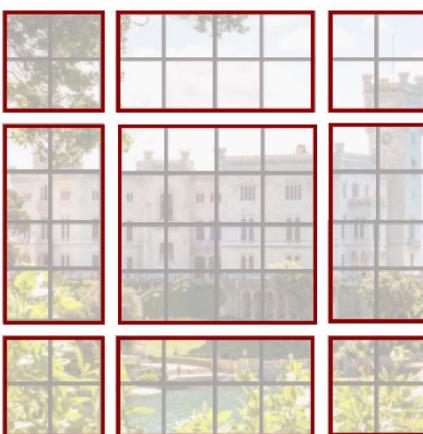
Standard Attention:

$$A = \text{Softmax} \left(\frac{QK^T}{\sqrt{D}} \right) V$$

$Q, K, V: M^2 \times D$ (Query, Key, Value)



Block L: Normal windows



Block L+1: Shifted Windows



Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and shifted windows in successive Transformer blocks

Detail: Relative Positional Bias

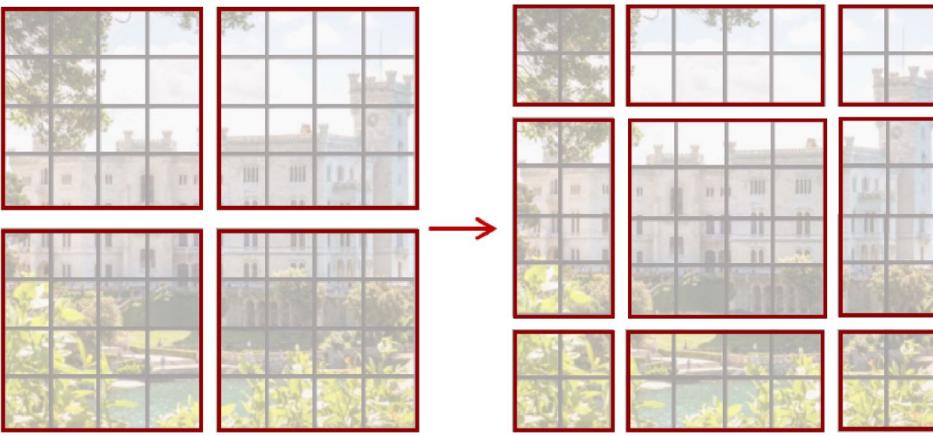
ViT adds positional embedding to input tokens, encodes *absolute position* of each token in the image

Swin does not use positional embeddings, instead encodes *relative position* between patches when computing attention:

Attention with relative bias:

$$A = \text{Softmax} \left(\frac{QK^T}{\sqrt{D}} + B \right) V$$

$Q, K, V: M^2 \times D$ (Query, Key, Value)
 $B: M^2 \times M^2$ (learned biases)



Block L: Normal windows

Block L+1: Shifted Windows

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

slide credit: Justin Johnson



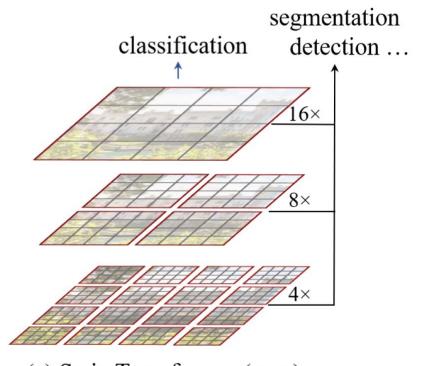
High Level Computer Vision | Bernt Schiele

29

Application to Various Tasks...

- General idea: think of (hierarchical tokens) as feature maps
 - allows to apply standard techniques for detection (e.g. Faster-RNN), instance segmentation (e.g. Mask-RCNN), semantic segmentation

	ImageNet		COCO		ADE20k
	top-1	top-5	AP ^{box}	AP ^{mask}	mIoU
w/o shifting	80.2	95.1	47.7	41.5	43.3
shifted windows	81.3	95.6	50.5	43.7	46.1
no pos.	80.1	94.9	49.2	42.6	43.8
abs. pos.	80.5	95.2	49.0	42.4	43.2
abs.+rel. pos.	81.3	95.6	50.2	43.4	44.0
rel. pos. w/o app.	79.3	94.7	48.2	41.9	44.1
rel. pos.	81.3	95.6	50.5	43.7	46.1



(a) Swin Transformer (ours)



High Level Computer Vision | Bernt Schiele

30

Overview of Today's Lecture

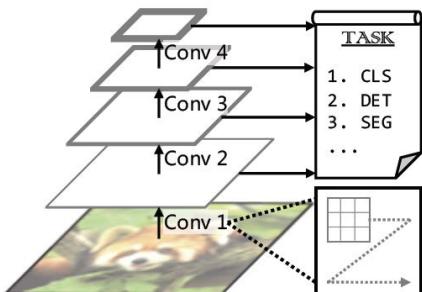
- Recap of Vision Transformer (ViT) for Image Classification
- Hierarchical Transformer Architectures
 - Swin Transformer — Hierarchical Vision Transformer using Shifted Windows
@ ICCV 2021 — <https://arxiv.org/abs/2103.14030>
 - SegFormer — Simple and Efficient Design for Semantic Segmentation with Transformers
@ NeurIPS 2021 — <https://arxiv.org/abs/2105.15203>
- Beyond Vanilla Gradient Descent
 - Adam Optimizer (and its components)
 - Second Order Optimization



Pyramid Vision Transformer Motivation

[22] He K, et al. "Deep residual learning for image recognition." in CVPR, 2016.
[54] Simonyan K, et al. "Very deep convolutional networks for large-scale image recognition." in ICLR, 2015.
[13] Dosovitskiy A, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." in ICLR 2021.

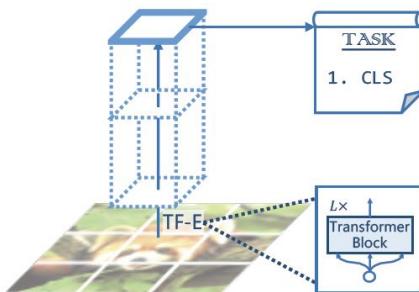
slide credit: Wenhui Wang & Enze Xie



(a) CNNs: VGG [54], ResNet [22], etc.

CNN's: Limitations

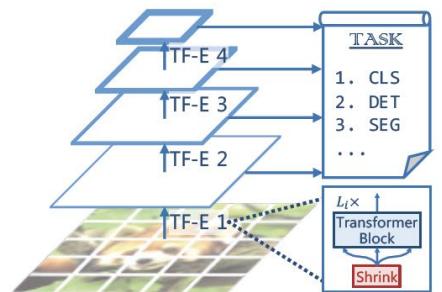
- Local Receptive Field
- Fixed Weights



(b) Vision Transformer [13]

ViT's Limitations

- Columnar Structure
- Low-Resolution Output
- Unsuitable for DET/SEG

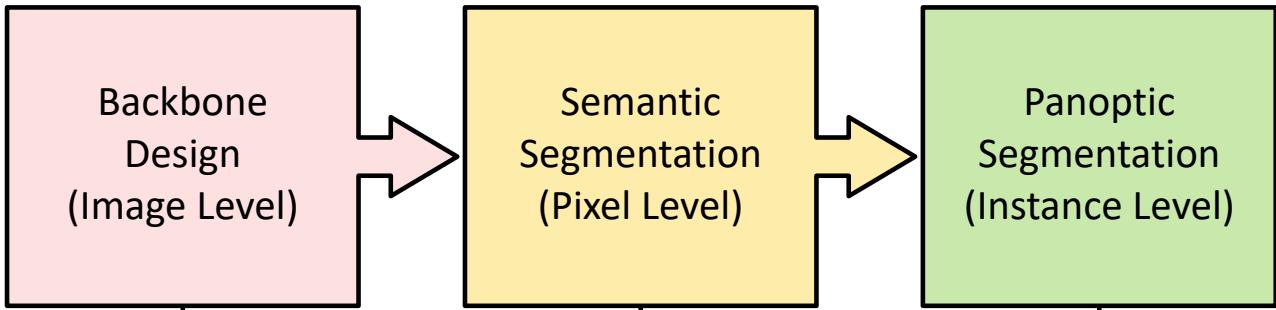


(c) Pyramid Vision Transformer (ours)
PVT (ours)

- A Transformer backbone as versatile as CNN



Pyramid Vision Transformer & SegFormer

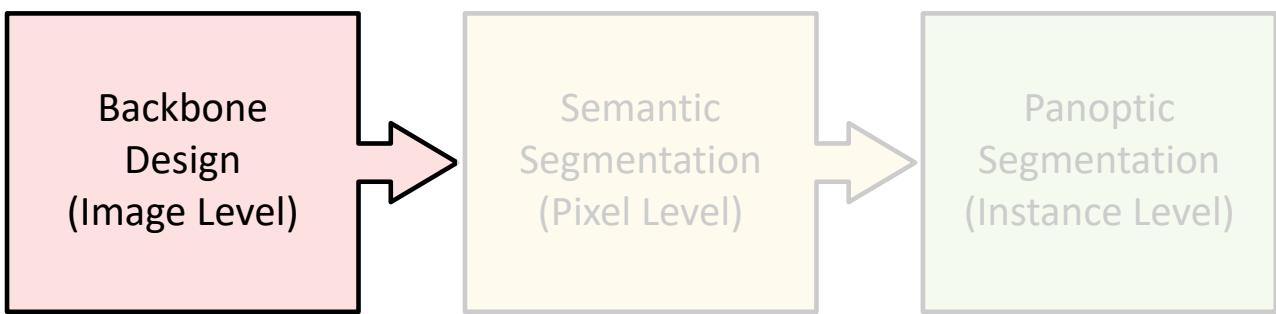


[1] **Pyramid Vision Transformer:** A Versatile Backbone for Dense Prediction without Convolutions. *In ICCV, 2021. (Oral)*

[2] **SegFormer:** Simple and Efficient Design for Semantic Segmentation with Transformers. *In NeurIPS, 2021.*

[3] **Panoptic SegFormer:** Delving Deeper into Panoptic Segmentation with Transformers. *In CVPR 2022.*

Backbone Design



[1] **Pyramid Vision Transformer:** A Versatile Backbone for Dense Prediction without Convolutions. *In ICCV, 2021. (Oral)*

[2] **SegFormer:** Simple and Efficient Design for Semantic Segmentation with Transformers. *In NeurIPS, 2021.*

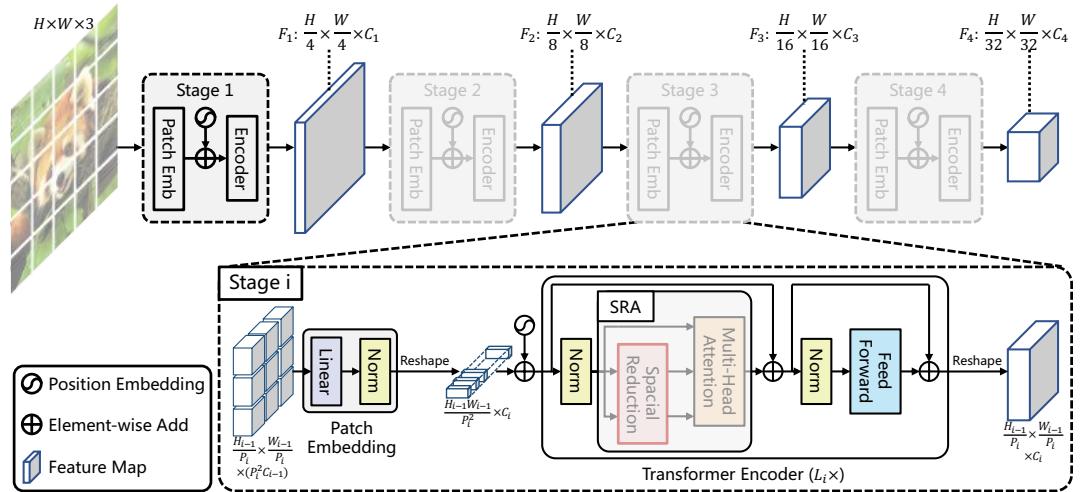
[3] **Panoptic SegFormer:** Delving Deeper into Panoptic Segmentation with Transformers. *In CVPR 2022.*

Pyramid Vision Transformer

- Overall Architecture

Key Points

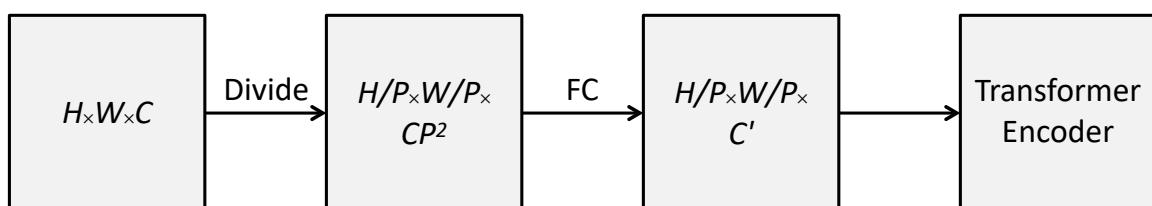
- Pyramid Structure*
- Four Stages:*
 - Patch Emb.*
 - Transformer Enc.*
- Spatial-Reduction Attention (SRA)*



Pyramid Vision Transformer

Q1: How to obtain the feature pyramid?

A: Patch embeddings with different **patch sizes (P)**



The process of the patch embedding in Stage i .



Pyramid Vision Transformer

Q2: How to reduce the computational cost of high-resolution feature maps?

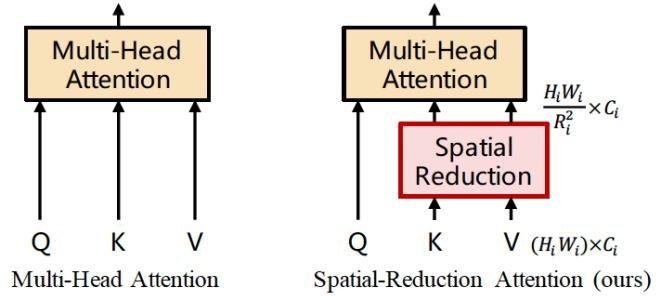
A: Spatial-Reduction Attention (SRA)

$$\text{SRA}(Q, K, V) = \text{Concat}(\text{head}_0, \dots, \text{head}_{N_i})W^O$$

$$\text{head}_j = \text{Attention}(QW_j^Q, \text{SR}(K)W_j^K, \text{SR}(V)W_j^V)$$

$$\text{SR}(\mathbf{x}) = \text{Norm}(\text{Reshape}(\mathbf{x}, R_i)W^S)$$

$$\text{Attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \text{Softmax}\left(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{d_{\text{head}}}}\right)\mathbf{v}$$



Compared to original multi-head attention, *the complexity of SRA is R_i^2 times lower!*



Pyramid Vision Transformer

Q2: How to reduce the computational cost of high-resolution feature maps?

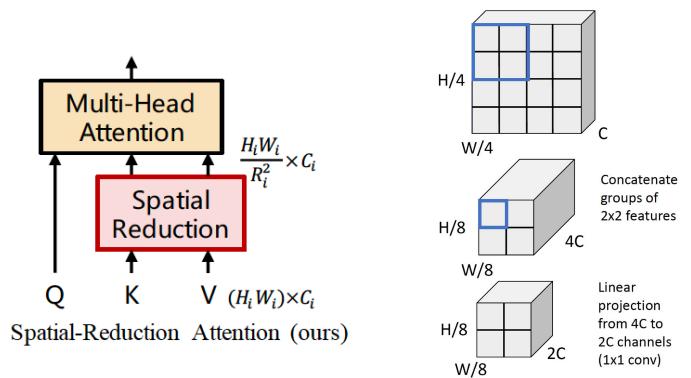
A: Spatial-Reduction Attention (SRA)

$$\text{SRA}(Q, K, V) = \text{Concat}(\text{head}_0, \dots, \text{head}_{N_i})W^O$$

$$\text{head}_j = \text{Attention}(QW_j^Q, \text{SR}(K)W_j^K, \text{SR}(V)W_j^V)$$

$$\text{SR}(\mathbf{x}) = \text{Norm}(\text{Reshape}(\mathbf{x}, R_i)W^S)$$

$$\text{Attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \text{Softmax}\left(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{d_{\text{head}}}}\right)\mathbf{v}$$



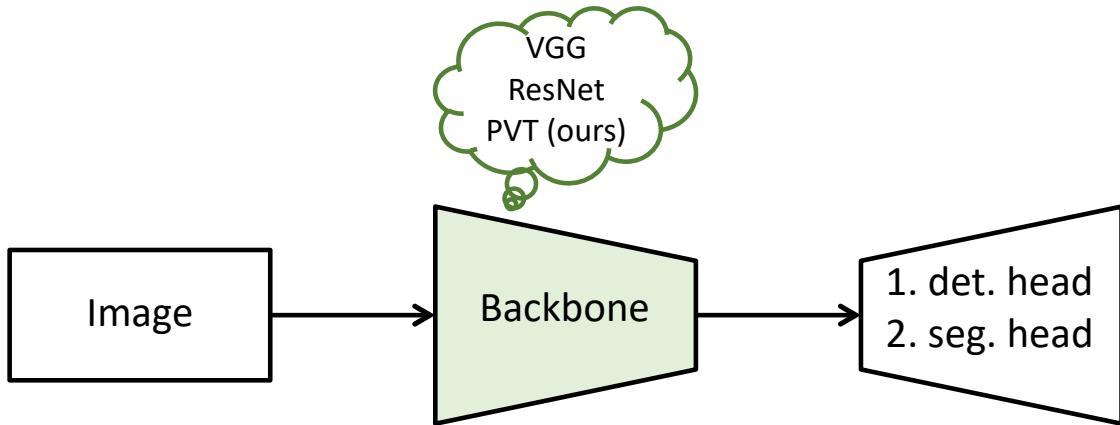
Compared to original multi-head attention, *the complexity of SRA is R_i^2 times lower!*



Pyramid Vision Transformer

Q3: How to apply PVT to detection and segmentation tasks?

A: Replacing the backbone.



Pyramid Vision Transformer

- Detailed Settings

- P_i : the patch size of the stage i ;
- C_i : the channel number of the output of the stage i ;
- L_i : the number of encoder layers in the stage i ;
- R_i : the reduction ratio of the SRA in the stage i ;
- N_i : the head number of the SRA in the stage i ;
- E_i : the expansion ratio of the feed-forward layer [51] in the stage i ;

	Output Size	Layer Name	PVT-Tiny	PVT-Small	PVT-Medium	PVT-Large
Stage 1	$\frac{H}{4} \times \frac{W}{4}$	Patch Embedding		$P_1 = 4; C_1 = 64$		
		Transformer Encoder	$R_1 = 8$ $N_1 = 1$ $E_1 = 8$	$\times 2$	$R_1 = 8$ $N_1 = 1$ $E_1 = 8$	$\times 3$
Stage 2	$\frac{H}{8} \times \frac{W}{8}$	Patch Embedding		$P_2 = 2; C_2 = 128$		
		Transformer Encoder	$R_2 = 4$ $N_2 = 2$ $E_2 = 8$	$\times 2$	$R_2 = 4$ $N_2 = 2$ $E_2 = 8$	$\times 3$
Stage 3	$\frac{H}{16} \times \frac{W}{16}$	Patch Embedding		$P_3 = 2; C_3 = 320$		
		Transformer Encoder	$R_3 = 2$ $N_3 = 5$ $E_3 = 4$	$\times 2$	$R_3 = 2$ $N_3 = 5$ $E_3 = 4$	$\times 18$
Stage 4	$\frac{H}{32} \times \frac{W}{32}$	Patch Embedding		$P_4 = 2; C_4 = 512$		
		Transformer Encoder	$R_4 = 1$ $N_4 = 8$ $E_4 = 4$	$\times 2$	$R_4 = 1$ $N_4 = 8$ $E_4 = 4$	$\times 3$

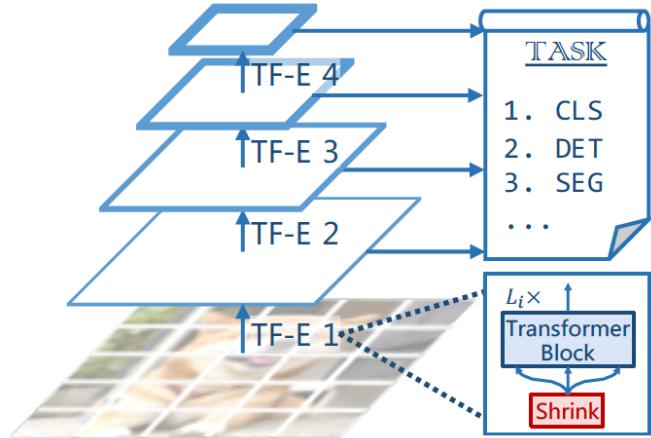
Table A1: Detailed settings of Pyramid Vision Transformer (PVT) series. The design follows the two rules of ResNet [5].

(1) With the growth of network depth, the hidden dimension gradually increases, and the output resolution progressively shrinks; (2) The major computation resource is concentrated in Stage 3.



Pyramid Vision Transformer

- Advantages
 - Multi-Scale/High-Resolution Output
 - As versatile as CNN, can be applied to detection/segmentation
 - Making pure Transformer detection/segmentation possible, for example
 - (1) PVT + DETR [1]
 - (2) PVT + Trans2Seg [2]



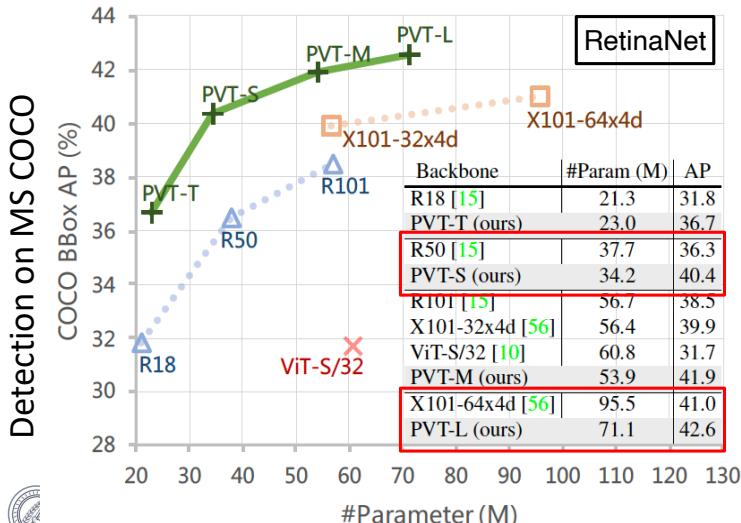
[1] Carion N, et al. "End-to-end object detection with transformers." in ECCV 2020.

[2] Xie E, et al. "Segmenting transparent object in the wild with transformer." in IJCAI 2021.



Pyramid Vision Transformer

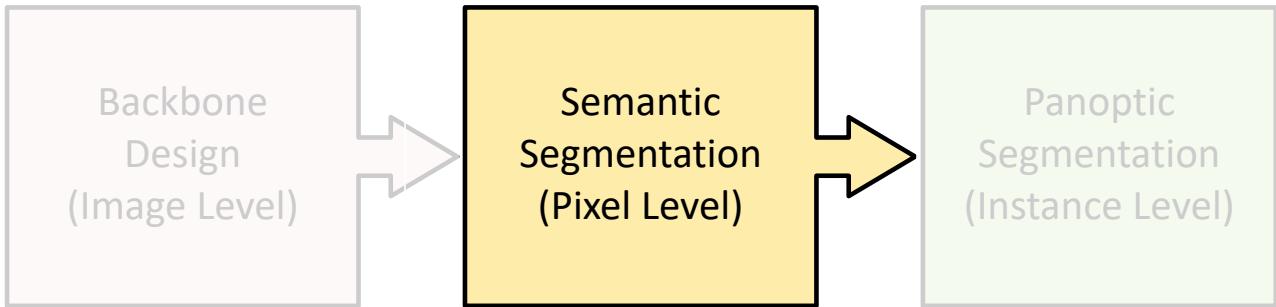
- Performance
 - Left: Object Detection — Right: Semantic Segmentation



Backbone	Semantic FPN		
	#Param (M)	GFLOPs	mIoU (%)
ResNet18 [22]	15.5	32.2	32.9
PVT-Tiny (ours)	17.0	33.2	35.7(+2.8)
ResNet50 [22]	28.5	45.6	36.7
PVT-Small (ours)	28.2	44.5	39.8(+3.1)
ResNet101 [22]	47.5	65.1	38.8
ResNeXt101-32x4d [73]	47.1	64.7	39.7(+0.9)
PVT-Medium (ours)	48.0	61.0	41.6(+2.8)
ResNeXt101-64x4d [73]	86.4	103.9	40.2
PVT-Large (ours)	65.1	79.6	42.1(+1.9)
PVT-Large* (ours)	65.1	79.6	44.8

Segmentation on ADE20K

Semantic Segmentation



[1] Pyramid Vision Transformer:
A Versatile Backbone for Dense
Prediction without Convolutions.
In ICCV, 2021. (Oral)

[2] SegFormer: Simple and
Efficient Design for Semantic
Segmentation with Transformers.
In NeurIPS, 2021.

[3] Panoptic SegFormer: Delving
Deeper into Panoptic Segmentation
with Transformers. *In CVPR 2022.*

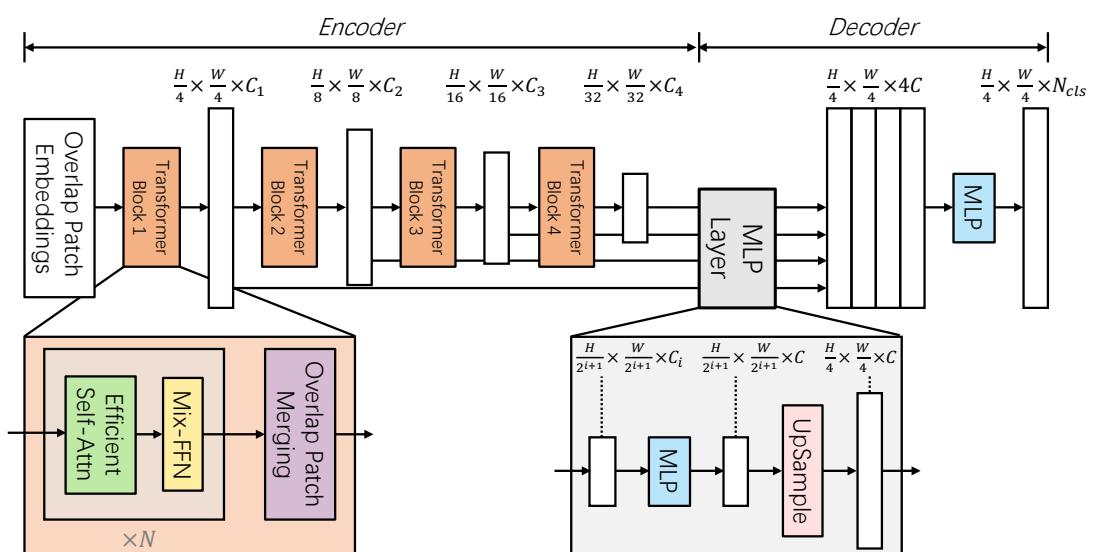


SegFormer

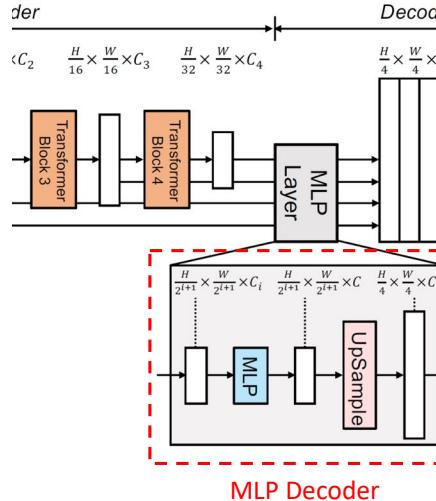
- Overall Architecture

Key Points

- **Encoder: PVT**
(No Pos. Encoding,
but Mix-FFN
(3×3 Conv + FFN))
- **MLP decoder &**
MLP head is
sufficient for seg.
Due to the large ERF
given by Transformer.



A Closer Look at MLP Decoder



$$\hat{F}_i = \text{Linear}(C_i, C)(F_i), \forall i$$

$$\hat{F}_i = \text{Upsample}\left(\frac{W}{4} \times \frac{W}{4}\right)(\hat{F}_i), \forall i$$

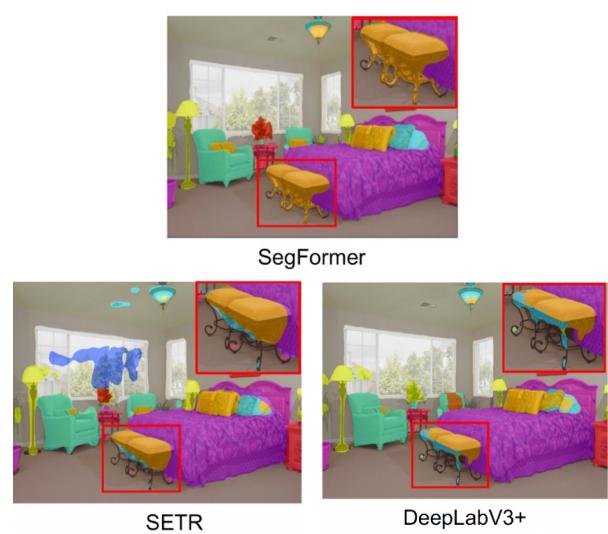
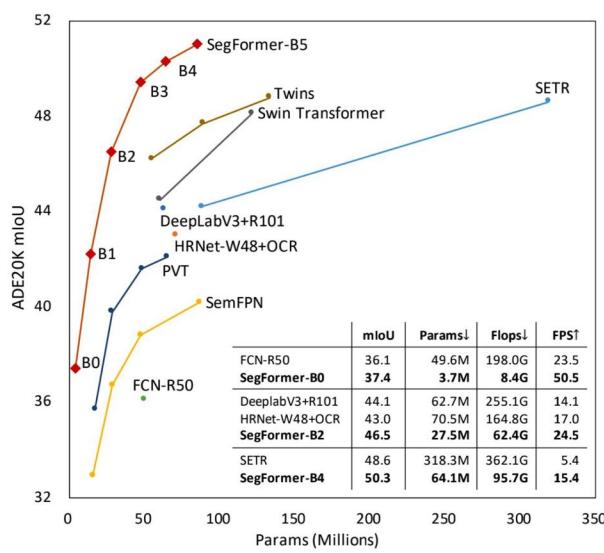
$$F = \text{Linear}(4C, C)(\text{Concat}(\hat{F}_i)), \forall i$$

$$M = \text{Linear}(C, N_{cls})(F),$$

1. unify to C channel dimensions for all transformer blocks
2. upsample
3. concatenate all blocks (4C channels) & project to C channels
4. predict the semantic classes from C channels

SegFormer

- Performance

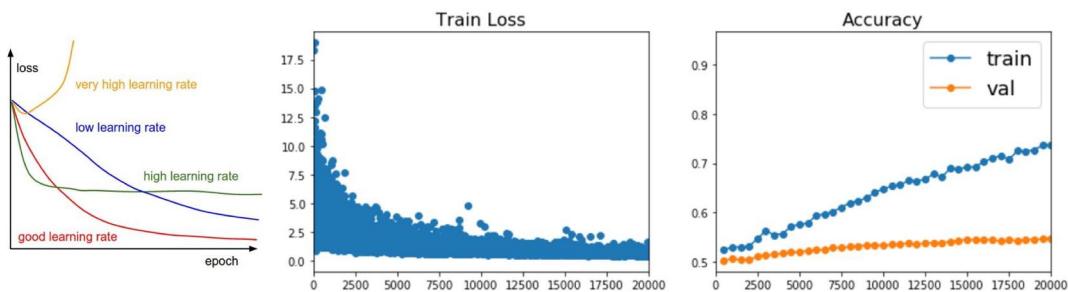


Overview of Today's Lecture

- Recap of Vision Transformer (ViT) for Image Classification
- Hierarchical Transformer Architectures
 - Swin Transformer — Hierarchical Vision Transformer using Shifted Windows
@ ICCV 2021 — <https://arxiv.org/abs/2103.14030>
 - SegFormer — Simple and Efficient Design for Semantic Segmentation with Transformers
@ NeurIPS 2021 — <https://arxiv.org/abs/2105.15203>
- Beyond Vanilla Gradient Descent
 - Adam Optimizer (and its components)
 - Second Order Optimization



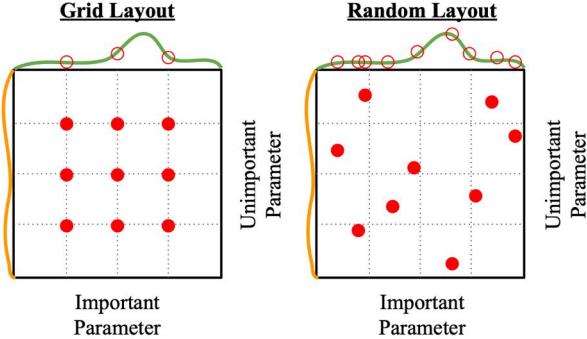
Babysitting Training



slide credit: Fei-Fei, Justin Johnson, Serena Yeung



Hyperparameter Search



Coarse to fine search

```
val acc: 0.412086, lr: 1.485200e-04, reg: 4.793564e-01, (1 / 100)
val acc: 0.214086, lr: 7.231880e-06, reg: 2.321281e-04, (2 / 100)
val acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val acc: 0.196000, lr: 1.551131e-05, reg: 4.374935e-05, (4 / 100)
val acc: 0.194000, lr: 1.181111e-05, reg: 3.000000e-01, (5 / 100)
val acc: 0.223000, lr: 4.215120e-05, reg: 4.198174e+01, (6 / 100)
val acc: 0.411000, lr: 1.750250e-04, reg: 2.110880e-04, (7 / 100)
val acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val acc: 0.482000, lr: 4.296863e-04, reg: 6.642355e-01, (9 / 100)
val acc: 0.079000, lr: 5.406162e-06, reg: 1.399350e+04, (10 / 100)
val acc: 0.154000, lr: 1.618980e-06, reg: 4.923252e-01, (11 / 100)
```

```
val acc: 0.527000, lr: 5.340517e-04, reg: 4.897824e-01, (0 / 100)
val acc: 0.492000, lr: 2.279484e-04, reg: 9.391130e-04, (1 / 100)
val acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val acc: 0.514000, lr: 1.113730e-04, reg: 1.000000e-01, (3 / 100)
val acc: 0.466000, lr: 1.113730e-04, reg: 5.244399e-02, (4 / 100)
val acc: 0.498000, lr: 9.477776e-04, reg: 2.801293e-03, (5 / 100)
val acc: 0.500000, lr: 1.113730e-04, reg: 1.000000e-01, (6 / 100)
val acc: 0.522000, lr: 5.586761e-04, reg: 2.312685e-01, (7 / 100)
val acc: 0.538000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val acc: 0.495000, lr: 1.479149e-04, reg: 1.000000e-01, (9 / 100)
val acc: 0.479000, lr: 2.421623e-04, reg: 2.287097e-01, (11 / 100)
val acc: 0.466000, lr: 1.135527e-04, reg: 3.905640e-02, (12 / 100)
val acc: 0.511000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val acc: 0.514000, lr: 6.438349e-04, reg: 3.633781e-01, (16 / 100)
val acc: 0.515000, lr: 1.113730e-04, reg: 1.000000e-01, (17 / 100)
val acc: 0.509000, lr: 9.752279e-04, reg: 2.856085e-03, (18 / 100)
val acc: 0.508000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (28 / 100)
val acc: 0.510000, lr: 8.039327e-04, reg: 1.282391e-02, (21 / 100)
```

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

49

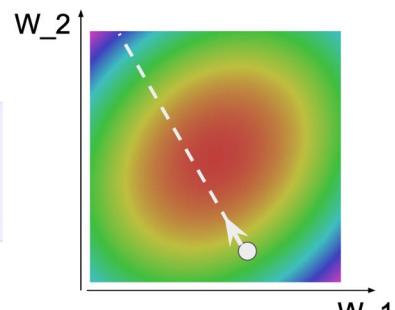
Optimization

```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$



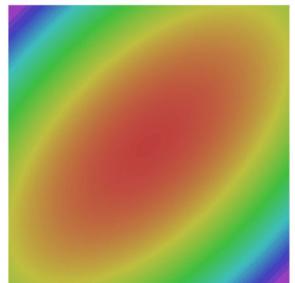
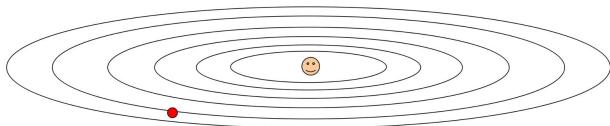
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

50

Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?



Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

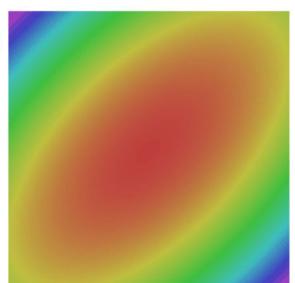
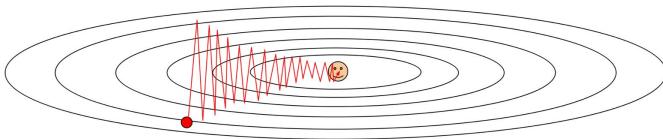
High Level Computer Vision | Bernt Schiele

51

Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?

Very slow progress along shallow dimension, jitter along steep direction



Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

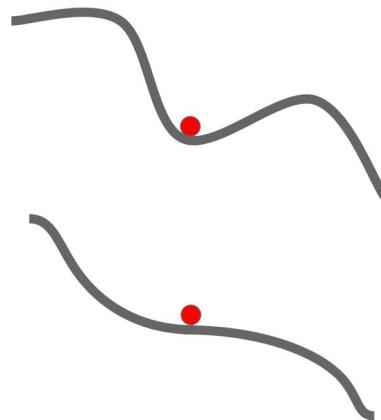
High Level Computer Vision | Bernt Schiele

52

Optimization: Problems with SGD

What if the loss
function has a
local minima or
saddle point?

Zero gradient,
gradient descent
gets stuck



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

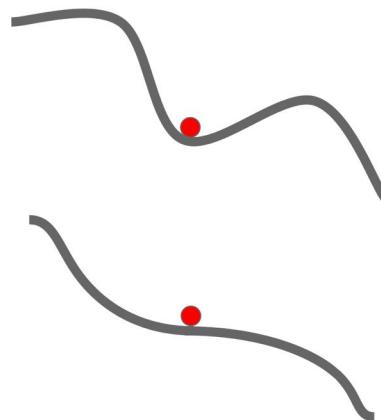
High Level Computer Vision | Bernt Schiele

53

Optimization: Problems with SGD

What if the loss
function has a
local minima or
saddle point?

Saddle points much
more common in
high dimension



Dauphin et al, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", NIPS 2014

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

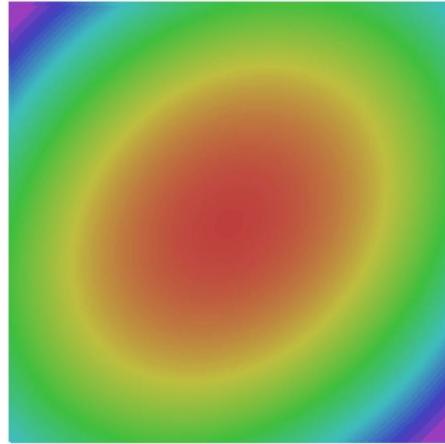
High Level Computer Vision | Bernt Schiele

54

Optimization: Problems with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W)$$
$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W)$$



slide credit: Fei-Fei, Justin Johnson, Serena Yeung



High Level Computer Vision | Bernt Schiele

55

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:  
    dx = compute_gradient(x)  
    x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0  
while True:  
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x -= learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

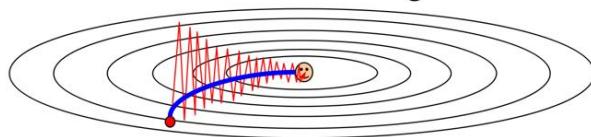
56

SGD + Momentum

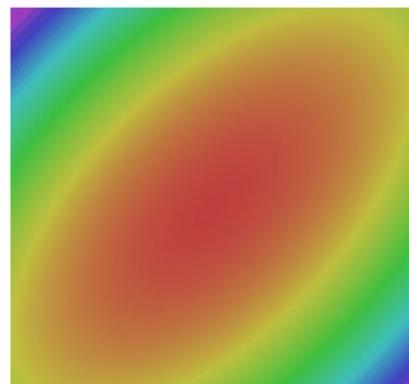
Local Minima Saddle points



Poor Conditioning



Gradient Noise



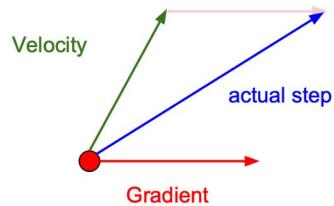
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

57

SGD + Momentum

Momentum update:



Combine gradient at current point with velocity to get step used to update weights

Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$ ", 1983
Nesterov, "Introductory lectures on convex optimization: a basic course", 2004
Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

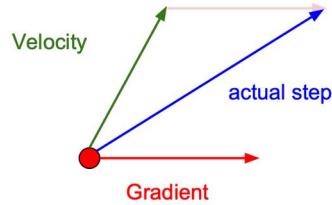
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

58

Nesterov Momentum

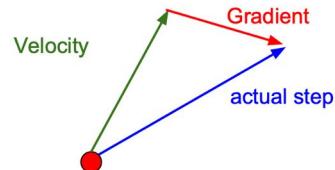
Momentum update:



Combine gradient at current point with velocity to get step used to update weights

Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$ ", 1983
 Nesterov, "Introductory lectures on convex optimization: a basic course", 2004
 Sutskever et al., "On the importance of initialization and momentum in deep learning", ICML 2013

Nesterov Momentum



"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction



Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

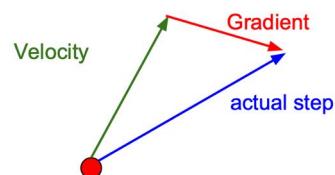
$$x_{t+1} = x_t + v_{t+1}$$

Annoying, usually we want update in terms of $x_t, \nabla f(x_t)$

Change of variables $\tilde{x}_t = x_t + \rho v_t$ and rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\begin{aligned} \tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1 + \rho)v_{t+1} \\ &= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t) \end{aligned}$$



"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction



AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

“Per-parameter learning rates”
or “adaptive learning rates”

Duchi et al, “Adaptive subgradient methods for online learning and stochastic optimization”, JMLR 2011

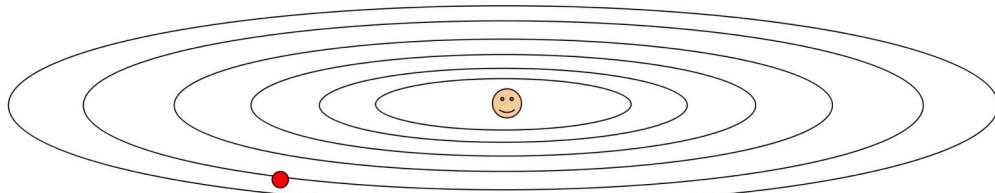
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

61

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Q: What happens with AdaGrad? Progress along “steep” directions is damped; progress along “flat” directions is accelerated

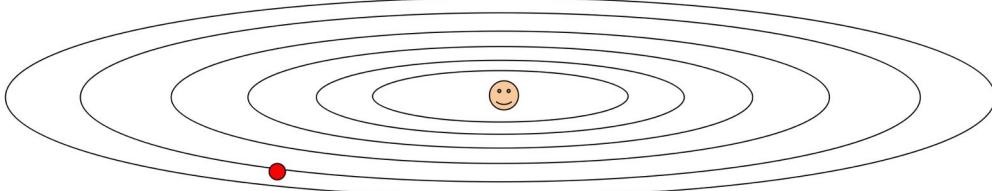
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

62

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Q2: What happens to the step size over long time? Decays to zero



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

63

RMSProp

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Tieleman and Hinton, 2012



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

64

Adam (almost)

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7)
```

Momentum

AdaGrad / RMSProp

Sort of like RMSProp with momentum

Q: What happens at first timestep?

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

65

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7)
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that
first and second moment
estimates start at zero

Adam with beta1 = 0.9,
beta2 = 0.999, and learning_rate = 1e-3 or 5e-4
is a great starting point for many models!

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

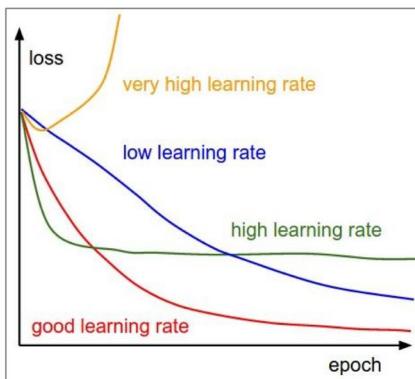
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

66

Learning Rate Schedule...

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



Q: Which one of these learning rates is best to use?



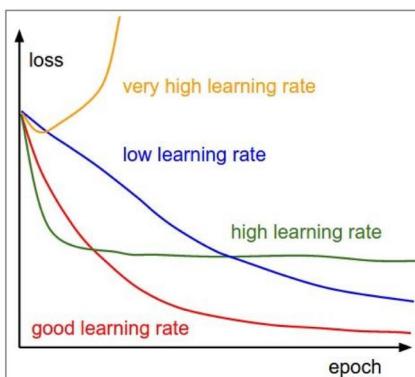
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

67

Learning Rate Schedule...

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



=> Learning rate decay over time!

step decay:

e.g. decay learning rate by half every few epochs.

exponential decay:

$$\alpha = \alpha_0 e^{-kt}$$

1/t decay:

$$\alpha = \alpha_0 / (1 + kt)$$



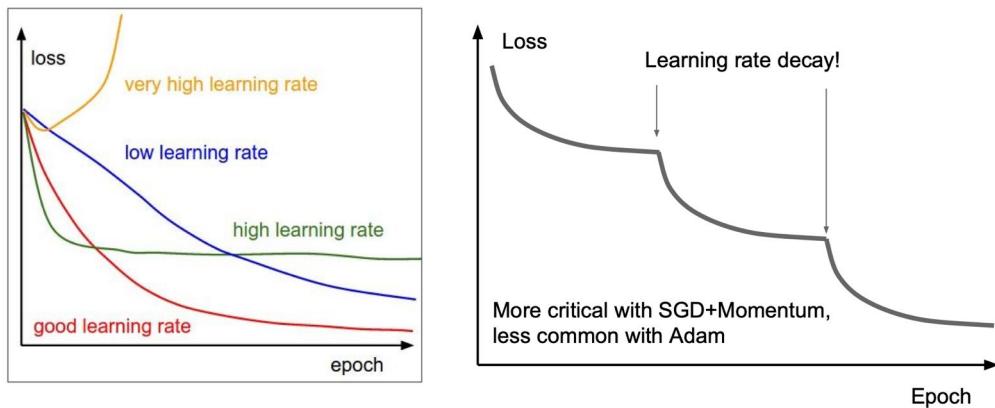
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

68

Learning Rate Schedule...

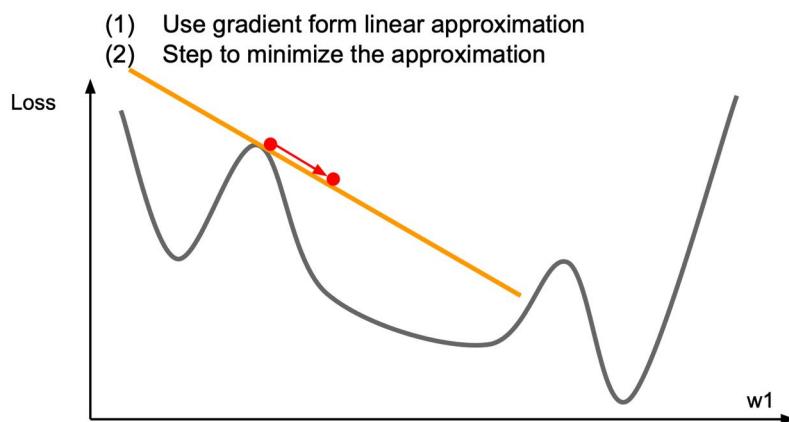
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



slide credit: Fei-Fei, Justin Johnson, Serena Yeung
High Level Computer Vision | Bernt Schiele

69

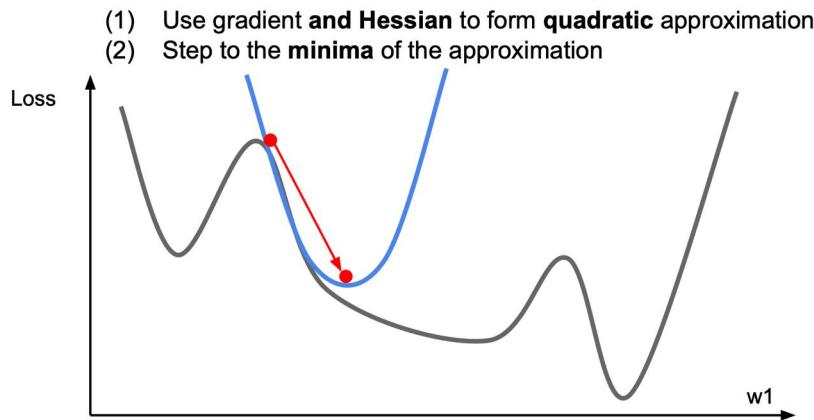
First-Order Optimization



slide credit: Fei-Fei, Justin Johnson, Serena Yeung
High Level Computer Vision | Bernt Schiele

70

Second-Order Optimization



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

71

Second-Order Optimization

second-order Taylor expansion:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Solving for the critical point we obtain the Newton parameter update:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

No hyperparameters!
No learning rate!
(Though you might use one in practice)

Q: What is nice about this update?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

72

Second-Order Optimization

second-order Taylor expansion:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Solving for the critical point we obtain the Newton parameter update:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Hessian has $O(N^2)$ elements
Inverting takes $O(N^3)$
 $N = (\text{Tens or Hundreds of}) \text{ Millions}$

Q2: Why is this bad for deep learning?



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

73

Second-Order Optimization

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

- Quasi-Newton methods (**BGFS** most popular):
instead of inverting the Hessian ($O(n^3)$), approximate inverse Hessian with rank 1 updates over time ($O(n^2)$ each).
- **L-BFGS** (Limited memory BFGS):
Does not form/store the full inverse Hessian.



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

High Level Computer Vision | Bernt Schiele

74

L-BFGS

- **Usually works very well in full batch, deterministic mode**
i.e. if you have a single, deterministic $f(x)$ then L-BFGS will probably work very nicely
- **Does not transfer very well to mini-batch setting.** Gives bad results. Adapting second-order methods to large-scale, stochastic setting is an active area of research.

Le et al, "On optimization methods for deep learning, ICML 2011"
Ba et al, "Distributed second-order optimization using Kronecker-factored approximations", ICLR 2017

slide credit: Fei-Fei, Justin Johnson, Serena Yeung



High Level Computer Vision | Bernt Schiele

75

In Practice:

- **Adam** is a good default choice in many cases
- **SGD+Momentum** with learning rate decay often outperforms Adam by a bit, but requires more tuning
- If you can afford to do full batch updates then try out **L-BFGS** (and don't forget to disable all sources of noise)

slide credit: Fei-Fei, Justin Johnson, Serena Yeung



High Level Computer Vision | Bernt Schiele

76