Deep Learning for Computer Vision II: Advanced Topics

Efficient Networks and Parameter-Efficient Fine-Tuning (PEFT)

(held by Prof. Rainer Stiefelhagen, Zdravko Marinov)

Content

- Efficient Neural Networks
  - Main Metrics and Concerns
  - Efficient Building Blocks
  - Efficient Networks
  - Quantization & Mixed Precision
  - Pruning
- Introduction to Parameter Efficient Fine-Tuning (PEFT)
  - Adapter
  - Prefix Tuning
  - Prompt Tuning
  - Low Rank Adaptation (LoRA)

Learning with Less (Resources)

EFFICIENT NEURAL NETWORKS

Efficient Neural Networks

- Overview
  - Main Metrics and Concerns
  - Efficient Building Blocks
  - Efficient Networks
  - Quantization & Mixed Precision
  - Pruning

Efficient Neural Networks

- Why do we need efficient neural networks?

Productionization

Training on high-power clusters

Inference on low-power device

Efficient Neural Networks

- Large disparity between hardware used for training and inference

- Even the average **gaming PC** only has a quadcore CPU and a Nvidia GTX 1060 with 6 GB VRAM
- The average notebook/smartphone is even worse than that!
- A lot less powerful than server setups with >100 GB RAM and multiple GPUs

Efficient Neural Networks

- Additional concerns for mobile devices
  - Power consumption when running battery-powered
  - Heat generation
  - Model weight size when downloading over mobile networks and also when stored on local volume
    - The ImageNet-pretrained ResNet-101 weights are already 171 MB!
    - Might stop users from downloading and using an app
  - Runtime
    - Many applications have realtime demands, e.g. processing camera input
    - Mobile hardware – especially smartphones – usually has very little computational resources

Efficient Neural Networks

- Given these concerns, we can intuitively derive the main metrics that are used to compare the efficiency of neural networks
  - Number of parameters, sometimes given as MB or kB sizes
  - Number of floating point calculations, usually given as FLOPs or Multiply-Adds (sometimes called Multiply-Accumulate or MAC)
    - Note that many hardware accelerators can compute a Multiply-Add operation in a single clock cycle.
    - Many researchers consider 1 Multiply-Add = 2 FLOPs. Some papers might measure this differently however!
  - Inference time as duration in seconds or throughput as frames per second
  - Energy Efficiency measured in Watt or Joule

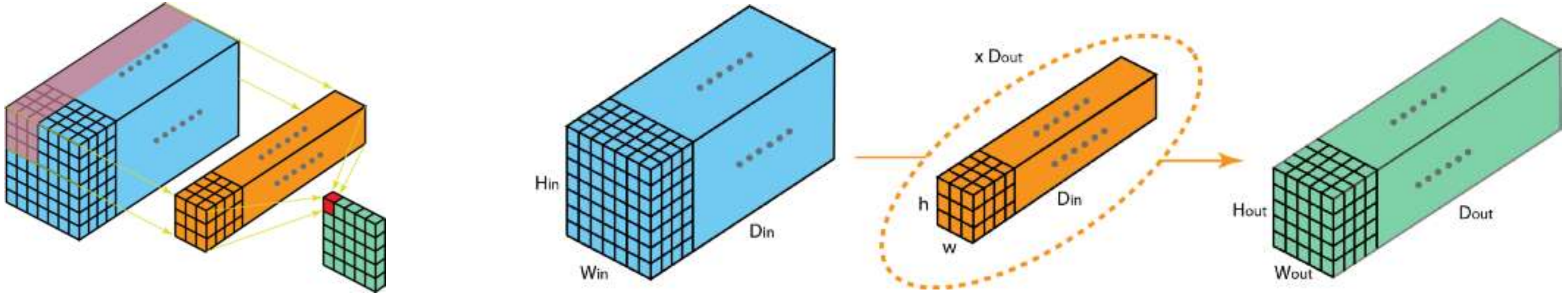
Faster ways to do convolution

EFFICIENT BUILDING BLOCKS

Efficient Building Blocks

- Standard convolution: Most commonly a $3x3xD_{in}$ filter kernel (h x w x D_{in})
- Single spatial position: multiply & add $3x3xD_{in}$ values of the input with those of the filter kernel

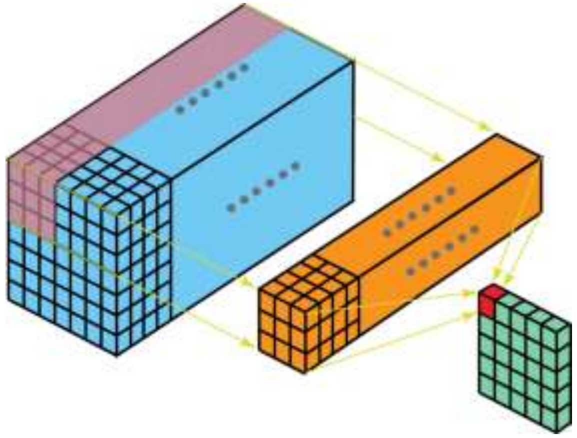
- Example below: input volume with H_{in}=W_{in}=7 and D_{in} channels and a filter with h=w=3 and D_{in} channels and no padding
- Outcome: **h x w x D_{in} x H_{out} x W_{out} x D_{out}** Multiply-Add operations and **h x w x D_{in} x D_{out}** weights

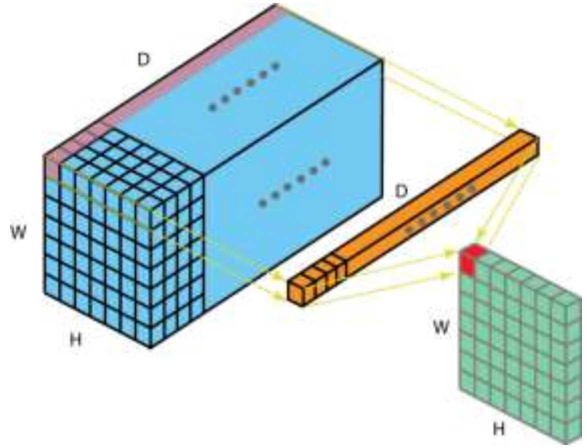


A single filter evaluation at a single spatial position and a full convolution [6]

Efficient Building Blocks

- Often h=w for a filter kernel, complexity is therefore quadratic w.r.t. h (or w)
- In terms of computations, h=w=3 is therefore 9 times as expensive as h=w=1!
- Takeaway: 1x1 convolutions are cheap!
- Problem: 1x1 filters lack spatial awareness, a CNN with **only** 1x1 filters would not perform well.
- But: we can use 1x1 convolution to reduce the input dimension D_{in} and apply 3x3 filters afterwards → the total number of 3x3 convolutions is reduced!

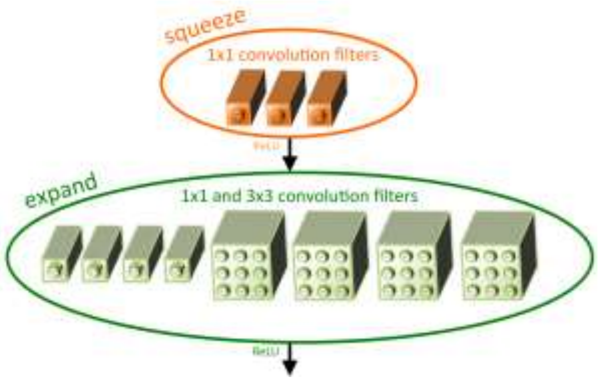




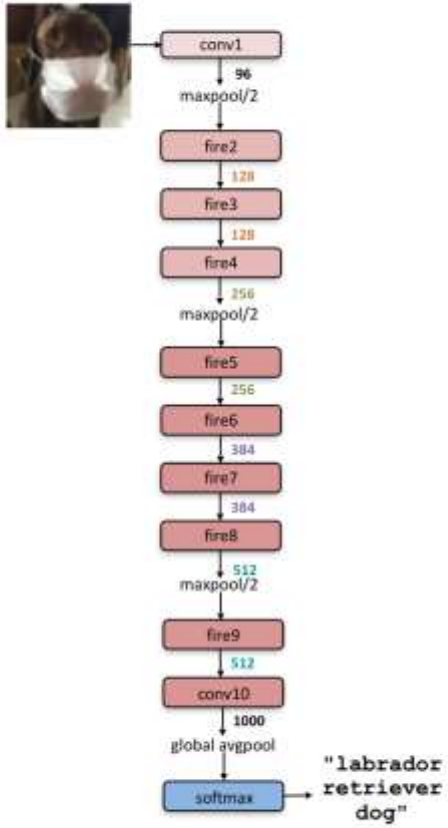
3x3 and 1x1 convolution in comparison [6]

SqueezeNet v1

- 1x1 convolutions extensively used in SqueezeNet v1 [5]
- Basic building block is the "Fire module"
  - First "squeeze" input: Reduce number of channels with **cheap 1x1 convolutions**
  - Then "expand" with a combination of 1x1 (cheap) and 3x3 (spatial information) filters
  - Concatenate output of 1x1 and 3x3 convolution
- Lowers both computation time and parameter count



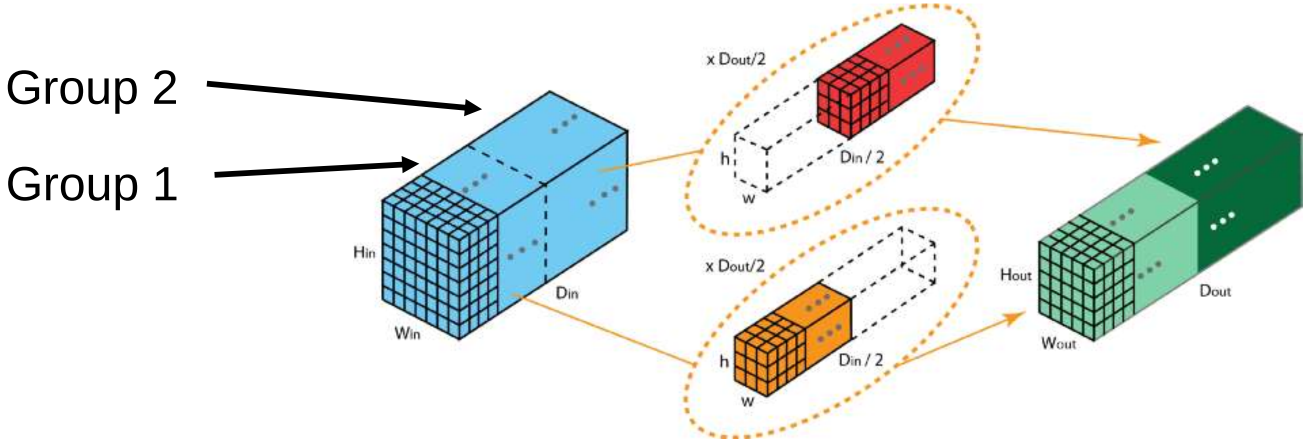
Fire module from [5]



SqueezeNet architecture

Grouped Convolution

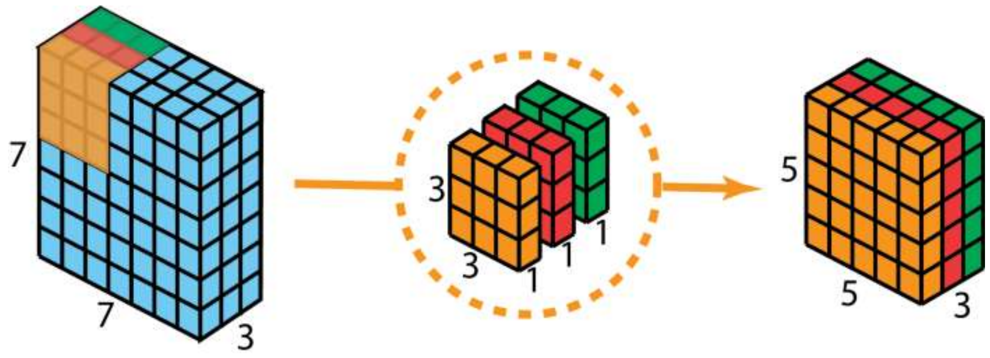
- Grouped convolution (sometimes called group convolution)
- First introduced in AlexNet [7] in 2012, at that time more an implementation detail, nowadays used for speeding up networks
- Main gist: divide input volume into groups. Filters only "work" on their group, in the example below number of groups g=2.
- Each filter only has 1/g amount of work and parameters
- But each filter also only sees 1/g channels and cannot work on all information



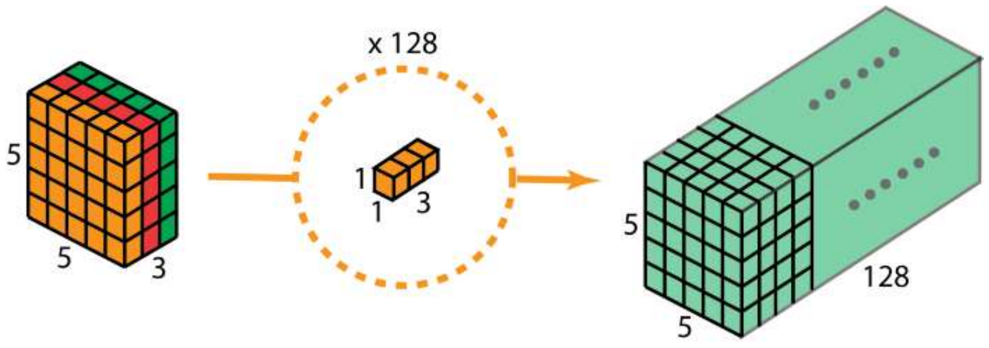
Grouped convolution [6]

Depthwise Separable Convolution

- Depthwise convolution is a special case of grouped convolution with $g=D_{in}$
- Every filter group only filters 1 channel of the input volume. This is very cheap computationally and has very few parameters.
- Depthwise separable convolution: depthwise convolution followed by a 1x1 convolution (1x1 convolution is also also referred to as pointwise convolution)



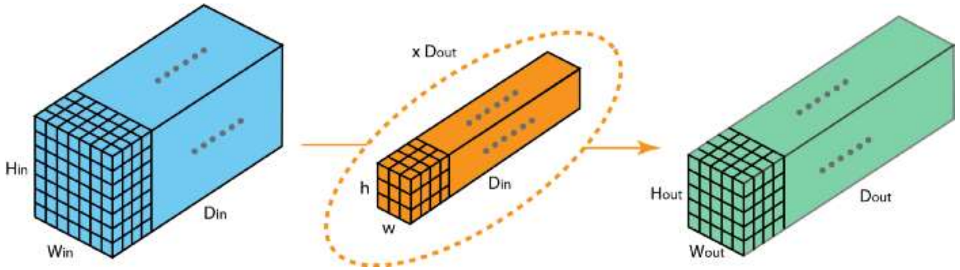
Depthwise convolution [6]



Pointwise convolution [6]

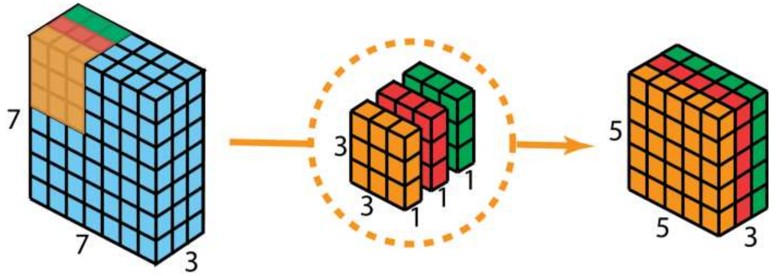
Question [5 minutes]

- (Reminder: standard convolution: $\mathbf{h \times w \times D_{in} \times H_{out} \times W_{out} \times D_{out}}$ Multiply-Add operations and $\mathbf{h \times w \times D_{in} \times D_{out}}$ weights)

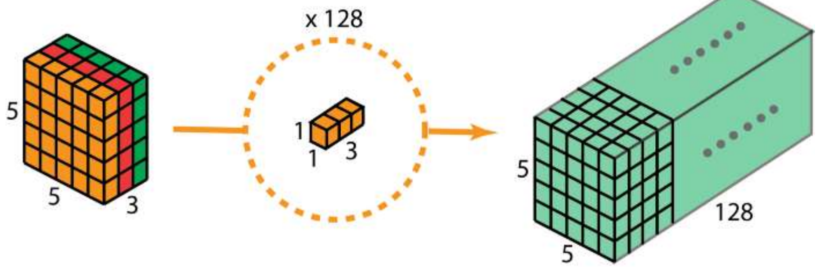


Standard convolution

- How many Multiply-Add operations and weights do depthwise and pointwise convolutions have? Given input: $\mathbf{H_{in} \times W_{in} \times D_{in}}$ output: $\mathbf{H_{out} \times W_{out} \times D_{out}}$ filter size: $\mathbf{h \times w \times 1}$ (for depthwise) and $\mathbf{1 \times 1 \times D_{in}}$ (for pointwise)



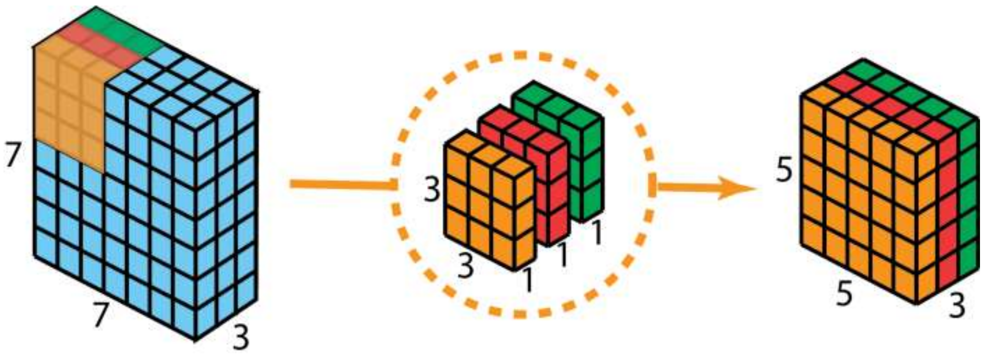
Depthwise convolution



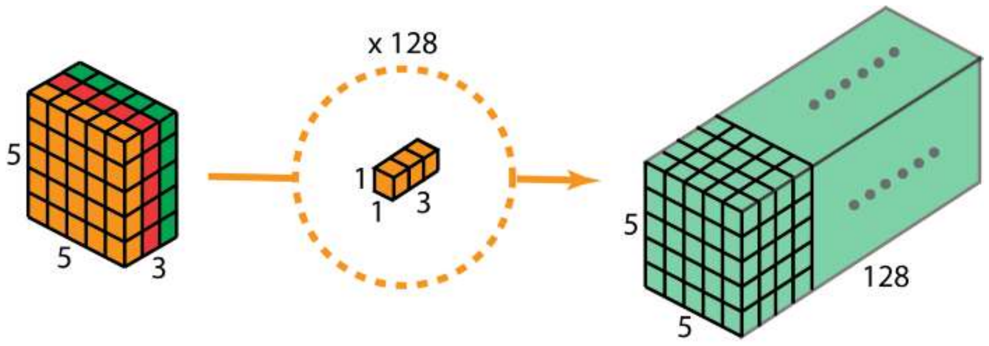
Pointwise convolution

Depthwise Separable Convolution

- (Reminder: standard convolution: $\mathbf{h \times w \times D_{in} \times H_{out} \times W_{out} \times D_{out}}$ Multiply-Add operations and $\mathbf{h \times w \times D_{in} \times D_{out}}$ weights)
- Depthwise part has $\mathbf{h \times w \times D_{in} \times H_{out} \times W_{out}}$ Multiply-Add operations and $\mathbf{h \times w \times D_{in}}$ weights
- Pointwise part has $\mathbf{D_{in} \times H_{out} \times W_{out} \times D_{out}}$ Multiply-Add operations and only $\mathbf{D_{in} \times D_{out}}$ weights
- For most inputs/outputs, even the combination of depthwise and pointwise part is more computationally efficient than a standard convolution



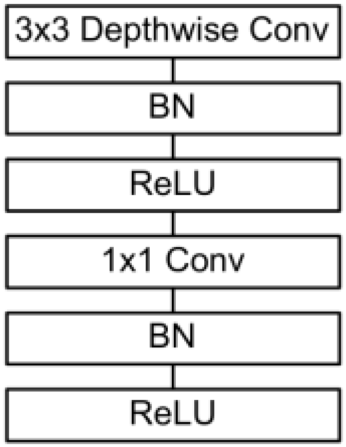
Depthwise convolution [6]



Pointwise convolution

MobileNets

- MobileNet v1 [9] is mostly based on depthwise separable convolution
- Basic building block is indeed very basic, but has been shown to work decently for many different tasks
- MobileNet v2 [10] expands on this basic unit and adds skip connections and inverted residual structures



MobileNet v1 building block

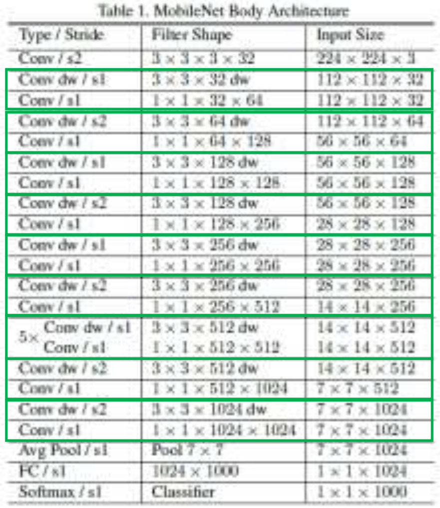
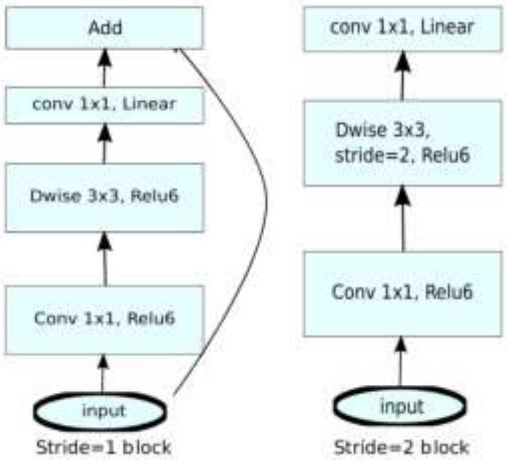


Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1	3 × 3 × 512 dw	14 × 14 × 512
5× Conv / s1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

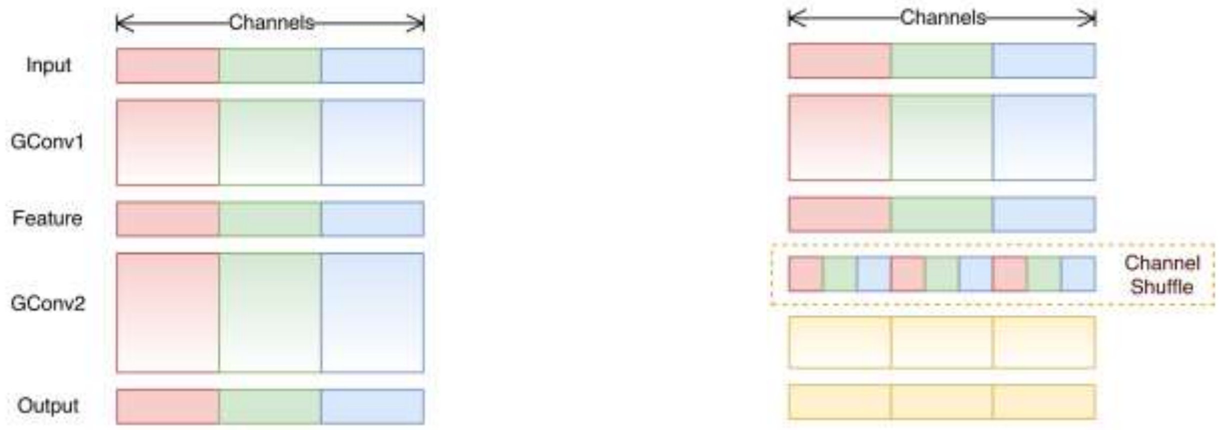
Stacked 13 times!



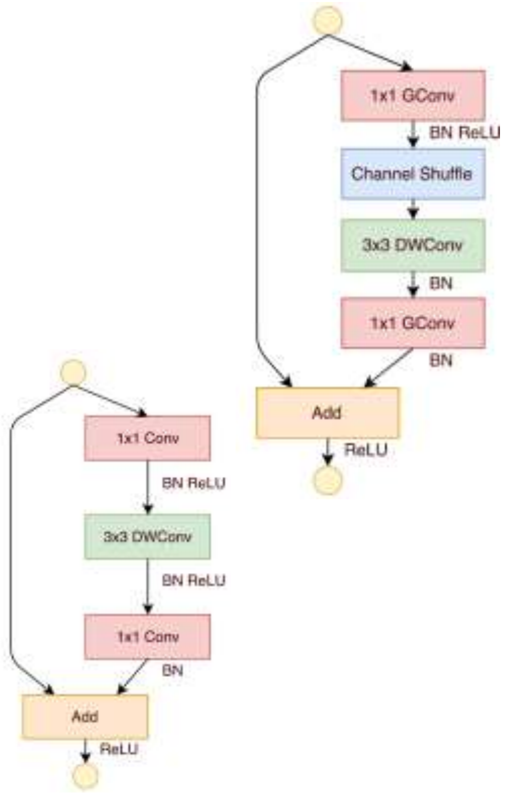
MobileNet v2 building blocks

ShuffleNet

- ShuffleNet [8] extensively uses grouped convolution
- Problem: When only using grouped convolution, information of the groups is never mixed (left). A red group filter would only work on information from previous red filters.
- Solution: **channel shuffle layer** (right). Channels are now mixed so that the next red filter can also consider information from the green and blue group



Visualization of the grouped convolution problem and its solution



ShuffleNet units

Efficient Building Blocks – Downsampling

- For CNNs, computational demand also depends on the size **h x w** of the input
- Filters have to be evaluated at every spatial position, which is expensive for large input sizes
- As often h=w, there is an obvious quadratic relationship between number of computations and the input size

- Thus, a common strategy of efficient neural networks is **downsampling fast**
  - Mostly handled by the top 2 layers ("stem cells")
  - Often a normal convolution with stride 2 (MobileNet v1) or a convolution with stride 2 followed by max pooling with stride 2 (SqueezeNet, ShuffleNet)
  - The latter reduces the common input size of **224x224** to **56x56** in only 2 layers!
  - This results in only 1/16th of spatial positions w.r.t. the input image

Mixed Precision, Quantization and Pruning

EFFICIENT TRAINING AND INFERENCE

Mixed Precision

- Commonly, neural networks are trained with 32-bit floating point (FP32) inputs and weight parameters
- This ensures a large range of representable numbers at the cost of storage space and computational power
- Using a smaller data type such as FP16 (half precision) would ensure more lightweight and more performant models and also faster training!

IEEE 754 32 bit float (single precision)

1	8	23

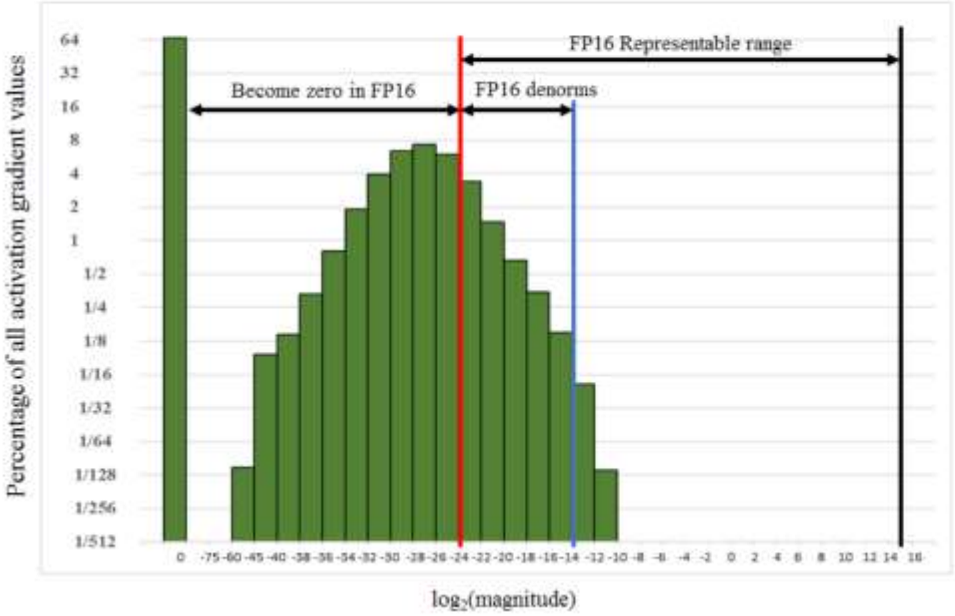
IEEE 754 16 bit float (half precision)

1	5	10

Sign, Exponent and Mantissa

Mixed Precision

- Problem: Representable range of FP16 is small, due to 5-bit exponent and 10-bit mantissa
- Gradients below 2^{-24} are rounded towards 0!
- This actually happens quite a lot during training



Histogram of activation gradient values during the training of Multibox SSD network [13]

Mixed Precision

- Result: Training diverges with FP16 although it would have converged with a FP32 data type
- Solution: Using a **mixed precision** approach with both FP16 and FP32 while also scaling the loss to an appropriate range

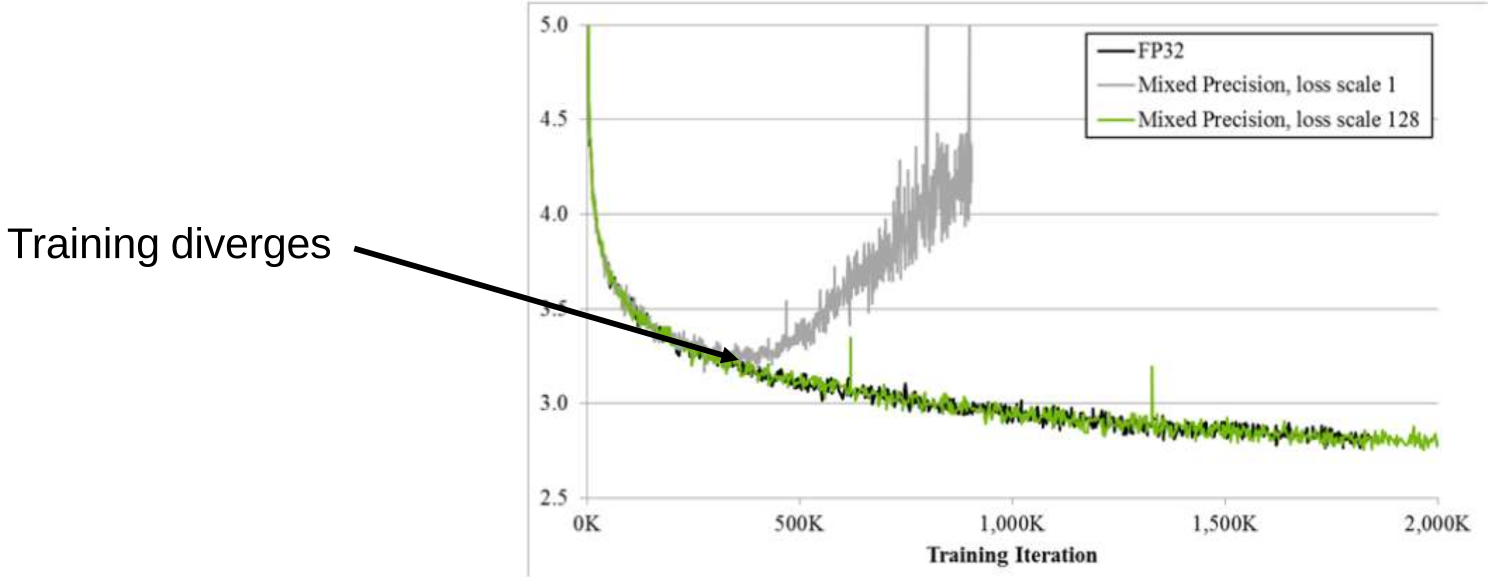


Image from https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/graphics/training-iteration.png

Mixed Precision

- Benefits of mixed precision training:
  - Half precision math throughput can be 2x-8x faster than single precision on modern GPUs
  - Weights stored on GPU take less space. Batch size can be increased!
  - Data transfers from/to the GPU are faster
  - Results mostly stay the same and can even increase in some cases
  - Easy to use in most deep learning frameworks such as PyTorch

Model	Baseline	Mixed Precision
AlexNet	56.77%	56.93%
VGG-D	65.40%	65.43%
GoogLeNet (Inception v1)	68.33%	68.43%
Inception v2	70.03%	70.02%
Inception v3	73.85%	74.13%
Resnet50	75.92%	76.04%

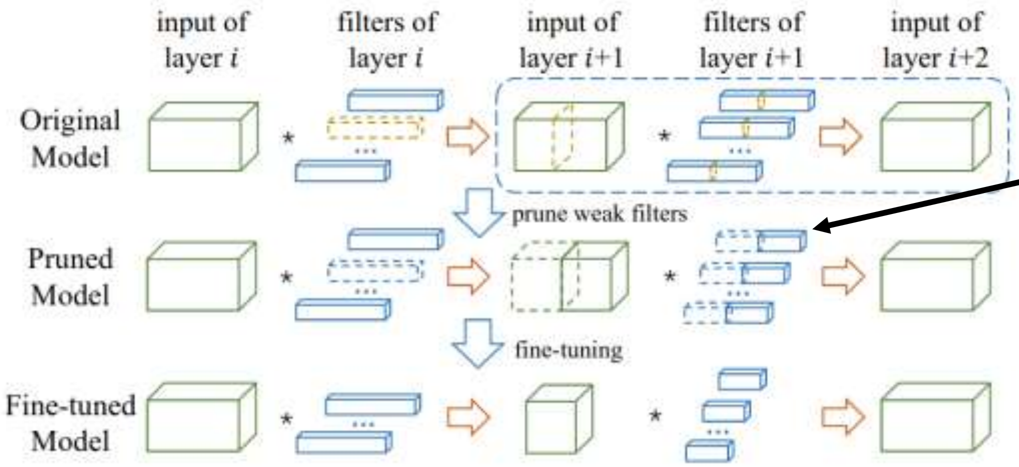
ILSVRC12 classification top-1 accuracy [13]

Pruning

- Pruning: removing redundancy/low value information from the network
- Pruning starts with a "bigger/heavier" network and tries to reduce the size
- Objective: Eliminate neurons or whole filters (in a CNN) while maintaining the metric (e.g. accuracy)
- Can help to remove e.g. multiple filters that learned (almost) the same feature like edge detection or color features
- Redundancy is actually quite common in NNs: Think about training with dropout, where often 50% of the values are randomly zeroed

Pruning

- There is not a singular pruning strategy that always works. Many different approaches can achieve a good pruning ratio
- However, a common setup is [17]:
  - Find unimportant filters according to some metric
  - Remove filters and adjust the filters of the subsequent layer
  - Finetune to "repair" the damage
  - Repeat until the target pruning percentage is achieved

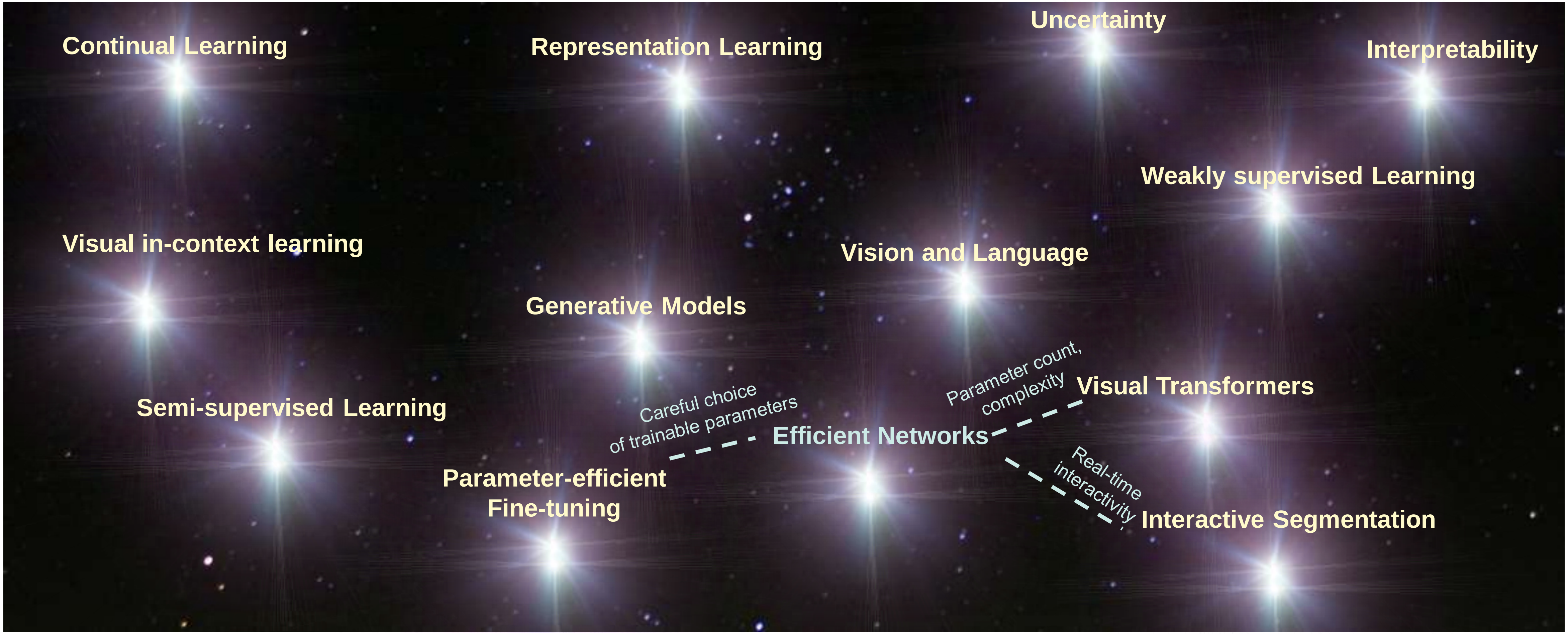


Subsequent filters have to be adjusted

Pruning

- How to determine which filter to remove?
- Common strategies and metrics:
  - Sum of absolute weight values in a filter. Small weights tend to produce weak activations and do not contribute much. ℓ_1 or ℓ_2 norms are commonly used.
  - Average Percentage of Zeros in a filter. Considers the sparsity of a filter, many zeros = information loss
  - Phrasing it as an optimization problem. [17] tries to find a filter that affects the output of the following layer the least, removes it and finetunes the network.
  - [18] uses an iterativ pruning approach, temporarily removing filters while monitoring the sensitivity metric of a detection task. Filters leading to the smallest drop are removed. No finetuning needed after every step.
- Differences in pruning setups:
  - Iterative vs. one-shot methods: Iterative setups only remove a small amount of filters per step.
  - Finetuning: Iterative methods often retrain after every pruning step, others only at the end.
  - Structured vs. Non-structured pruning: Structured pruning removes whole filters, non-structured removes single weights to induce sparsity. This often requires special hard- or software to handle.
  - Global vs. Local pruning: Global pruning considers all filters, local e.g. only a single layer.

Constellations in Efficient Networks



PARAMETER-EFFICIENT FINE-TUNING

Introduction to Parameter-Efficient Fine-Tuning (PEFT)

- LLMs have a lot of weights → Fine-tuning is expensive
  - More compute – large and multiple GPUs
  - File size – Checkpoints (GPT-3 – 800 GB)

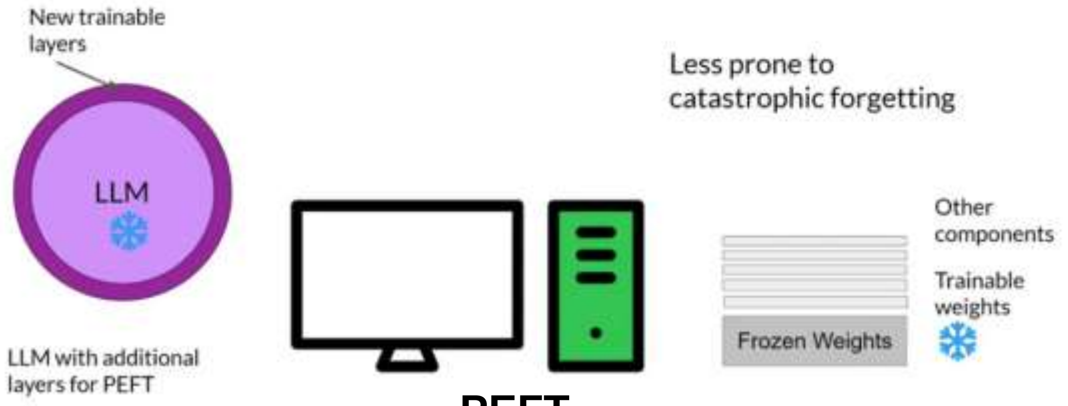
GPU	Tier	$ / hr (AWS)	VRAM (GiB)
H100	Enterprise	12.29	80
A100	Enterprise	5.12	80
V100	Enterprise	3.90	32
A10G	Enterprise	2.03	24
T4	Enterprise	0.98	16
RTX 4080	Consumer	N/A	16

Table taken from the DeepLearningAI 2023 workshop at https://www.youtube.com/watch?v=g68qlo9Izf0

Introduction to Parameter-Efficient Fine-Tuning (PEFT)

- Avoid tuning the whole model
  - Fine-tune only small subset of the model parameters
  - Allows fine-tuning large models on consumer GPUs
- Difference between full fine-tuning and PEFT
  - Pros (PEFT): computational and storage efficiency, and less prone to catastrophic forgetting

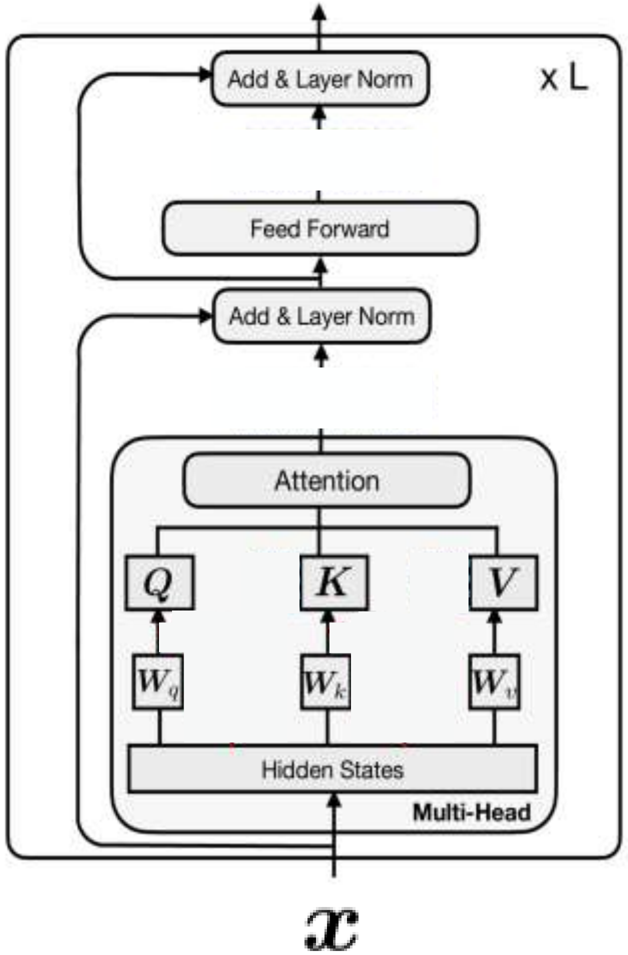
Full fine-tuning



PEFT

Images taken https://medium.com/@kanikaadik07/peft-parameter-efficient-fine-tuning-55e32c60c799

Recap: Transformer Models [1], [2]



$$\mathrm{Attn}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \mathrm{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V}$$

$$\mathrm{MHA}(\boldsymbol{x}) = \mathrm{Concat}(\mathrm{head}_1, \cdots, \mathrm{head}_h)\boldsymbol{W}_o,\ \mathrm{head}_i = \mathrm{Attn}(\boldsymbol{x}\boldsymbol{W}_q^{(i)}, \boldsymbol{x}\boldsymbol{W}_k^{(i)}, \boldsymbol{x}\boldsymbol{W}_v^{(i)}),\ \boldsymbol{x} \in \mathbb{R}^d$$

$$\mathrm{FFN}(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$

$$\boldsymbol{W}_q^{(i)}, \boldsymbol{W}_k^{(i)}, \boldsymbol{W}_v^{(i)} \in \mathbb{R}^{d \times d_h}$$

$$\boldsymbol{W}_o \in \mathbb{R}^{d \times d}$$

$$\boldsymbol{W}_1 \in \mathbb{R}^{d \times d_m}, \boldsymbol{W}_2 \in \mathbb{R}^{d_m \times d}$$

PARTIAL FINE-TUNING

Question [5 minutes]

- Partial Fine-tuning
  - Fine-tune part of the layers (usually the last ones)

- Why could this be a potential problem for large domain shifts in inference?

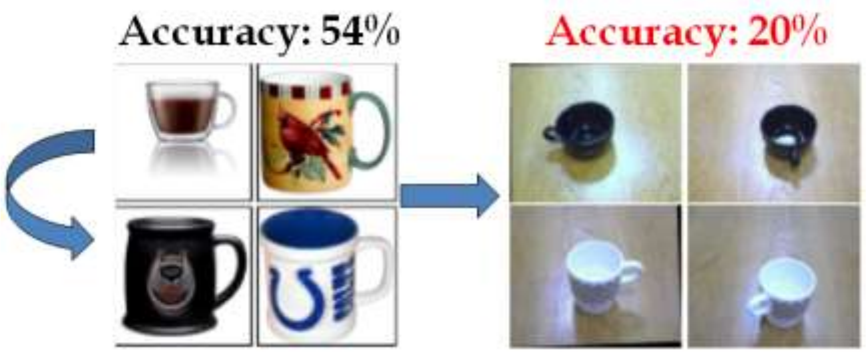


Image source: https://ai.bu.edu/adaptation.html

Partial Fine-tuning

- Partial Fine-tuning
  - Fine-tune part of the layers (usually the last ones)
  - Can be considered as PEFT
  - Does not mitigate large domain shifts
    - Adapters, Prompt Tuning, Prefix Tuning, and LoRA are better in practice
    - Adapt representation **at different levels** in the model
      - E.g. adapt low-level features in large appearance shifts

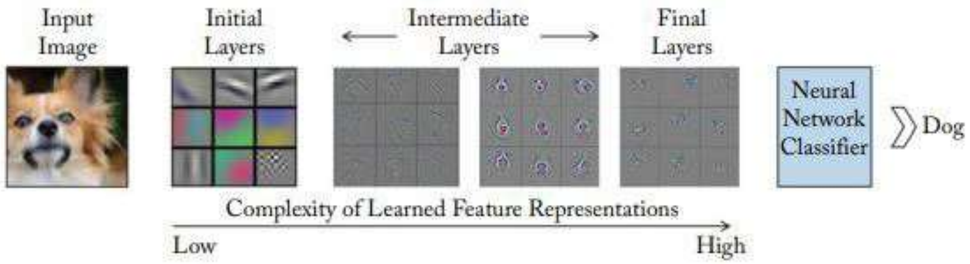
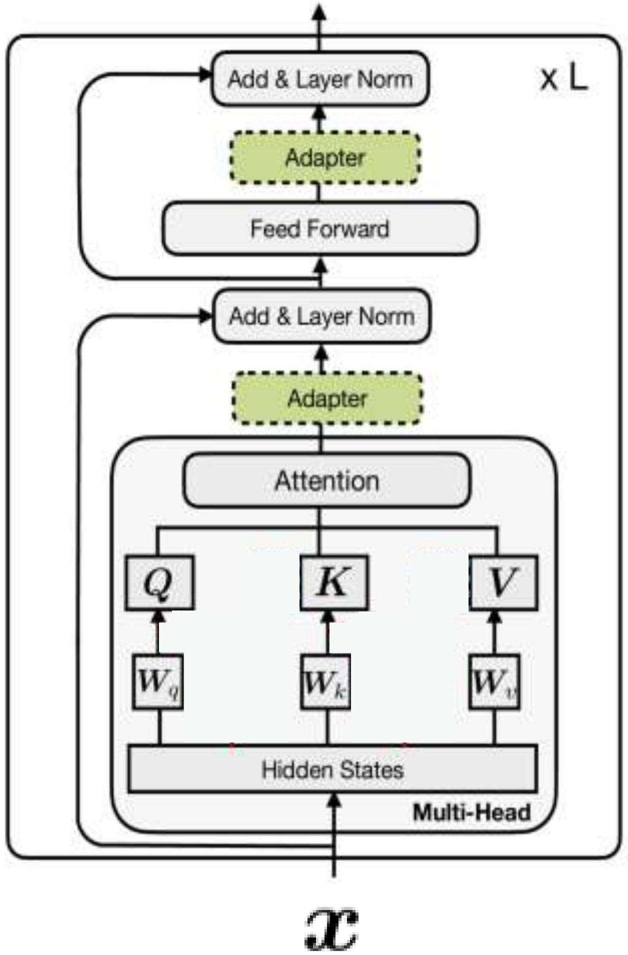


Image Source: Sayeed, Mohammed Azam, et al. "Detecting Malaria from Segmented Cell Images of Thin Blood Smear Dataset using Keras from Tensorflow." *International Journal for Research in Applied Science and Engineering Technology* 8.1 (2020): 597-607.

ADAPTERS

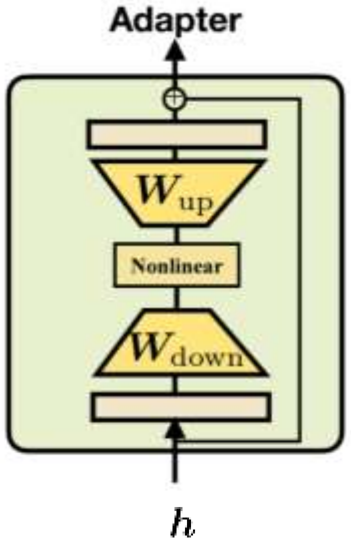
Adapters [2], [3]



$$\mathrm{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V}$$

$$\mathrm{MHA}(\boldsymbol{x}) = \mathrm{Concat}(\mathrm{head}_1, \cdots, \mathrm{head}_\mathrm{h})\boldsymbol{W}_o, \ \mathrm{head}_\mathrm{i} = \mathrm{Attn}(\boldsymbol{x}\boldsymbol{W}_q^{(i)}, \boldsymbol{x}\boldsymbol{W}_k^{(i)}, \boldsymbol{x}\boldsymbol{W}_v^{(i)}), \ \boldsymbol{x} \in \mathbb{R}^d$$

$$\mathrm{FFN}(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$



$$\boldsymbol{h} \leftarrow \boldsymbol{h} + f(\boldsymbol{h}\boldsymbol{W}_{\mathrm{down}})\boldsymbol{W}_{\mathrm{up}}$$

$$\boldsymbol{W}_q^{(i)}, \boldsymbol{W}_k^{(i)}, \boldsymbol{W}_v^{(i)} \in \mathbb{R}^{d \times d_h}$$

$$\boldsymbol{W}_o \in \mathbb{R}^{d \times d}$$

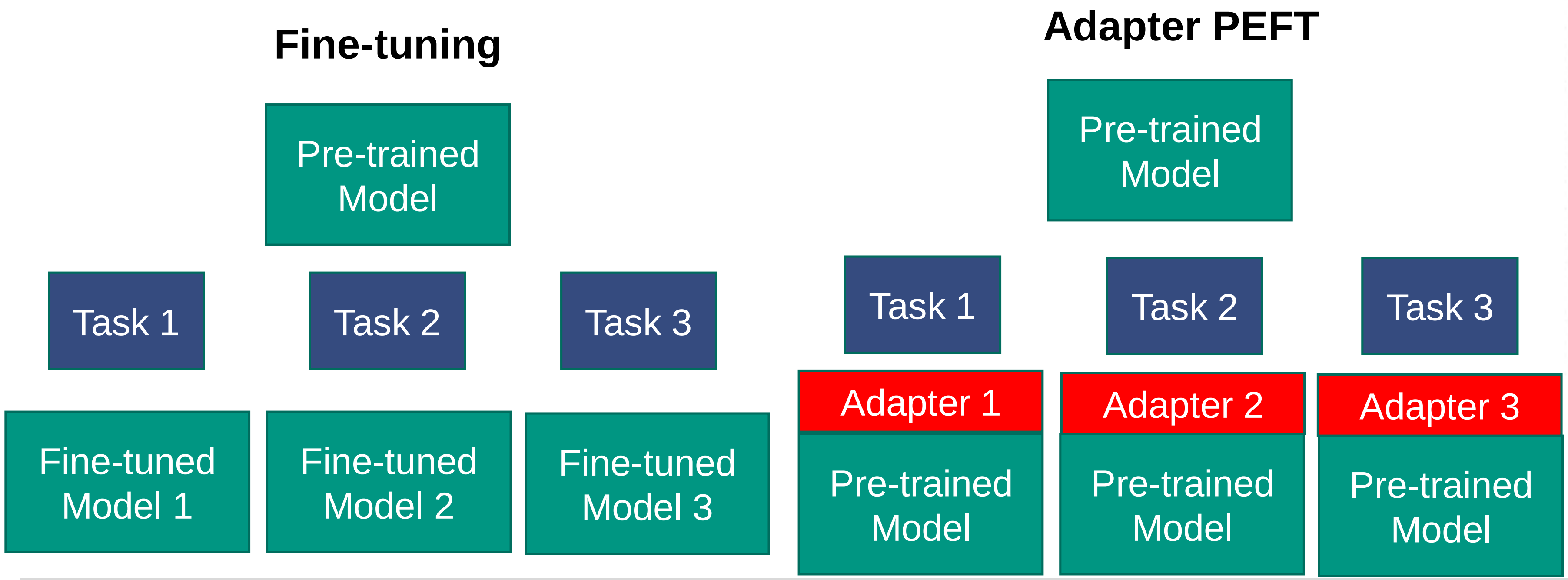
$$\boldsymbol{W}_1 \in \mathbb{R}^{d \times d_m}, \boldsymbol{W}_2 \in \mathbb{R}^{d_m \times d}$$

Adapters

- Methodology
  - Adapt the pre-trained model at **multiple levels**
  - Insert adapter modules between pre-trained layers
    - Small set of additional parameters
    - Fine-tune only the task-specific adapter modules

Adapters [2], [3]

- Adds "corrections" to the learned representations of the pre-trained model
- Pre-trained model is unchanged
- New tasks → New adapters!
  - Reduced storage and training cost compared to fine-tuning
  - Only need to store the pre-trained model and the small task-specific adapters



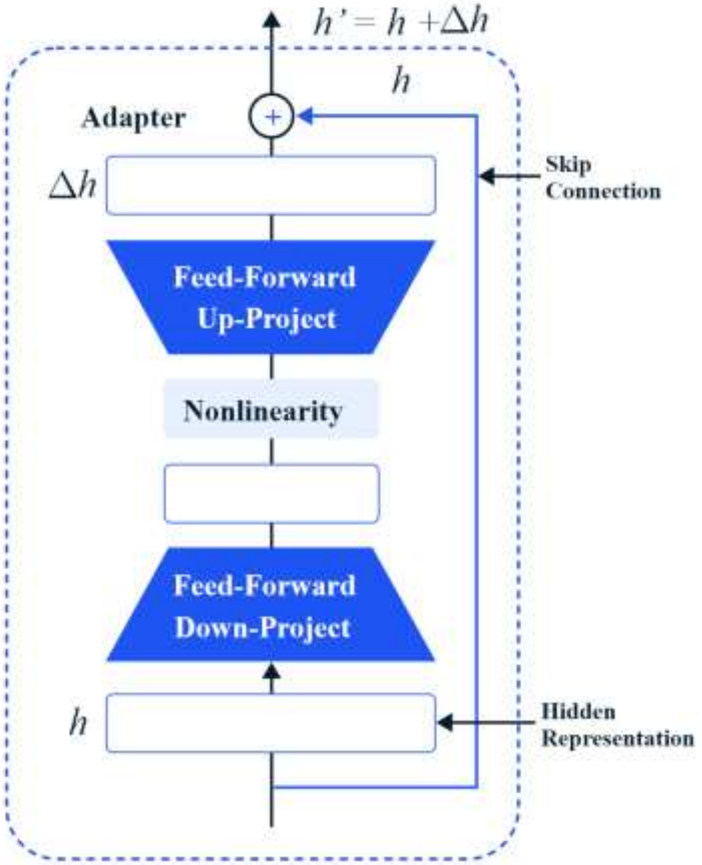


Image taken from: https://www.leewayhertz.com/parameter-efficient-fine-tuning/

Adapters – Example for 2D → 3D Segmentation [4]

- Given a model trained to segment cats and dogs (and other standard classes)

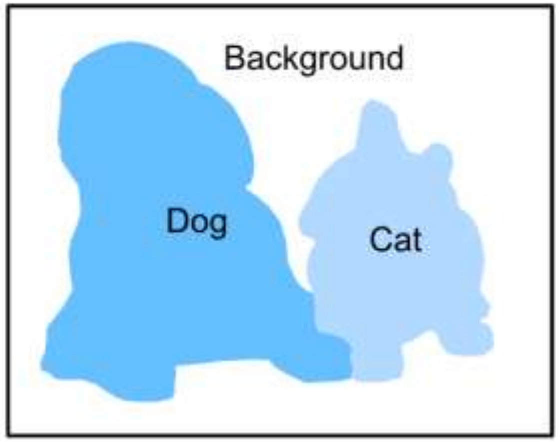


Image taken from: https://kiansoon.medium.com/semantic-segmentation-is-the-task-of-partitioning-an-image-into-multiple-segments-based-on-the-356a5582370e

Adapters – Example for 2D → 3D Segmentation [4]

- Given a model trained to segment cats and dogs (and other standard classes)
- Adapt it to segment volumetric brain tumors

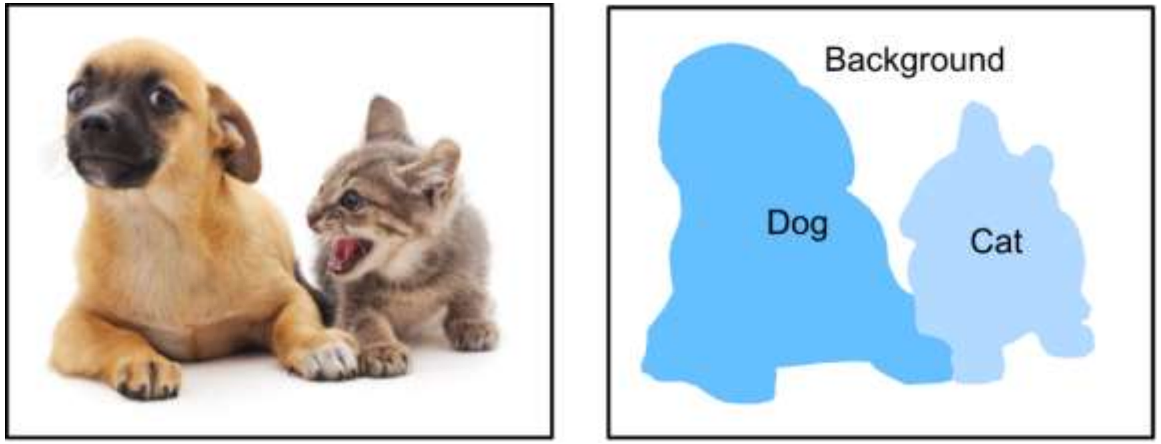


Image taken from: https://kiansoon.medium.com/semantic-segmentation-is-the-task-of-partitioning-an-image-into-multiple-segments-based-on-the-356a5582370e

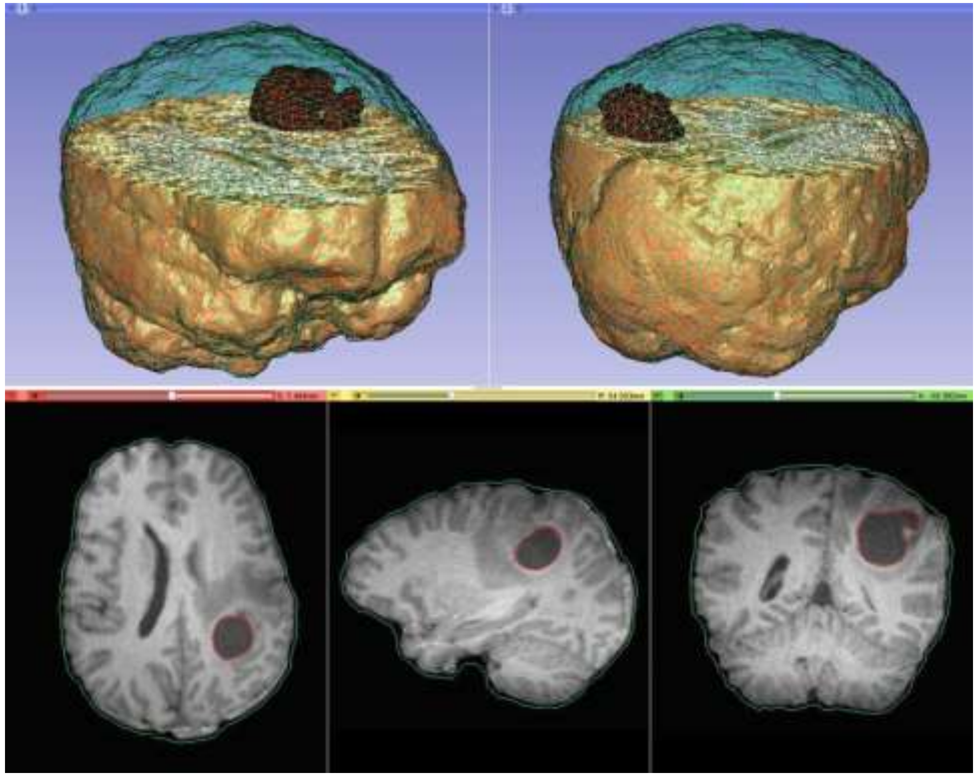


Image taken from [6]

Adapters – Example for 2D → 3D Segmentation [4]

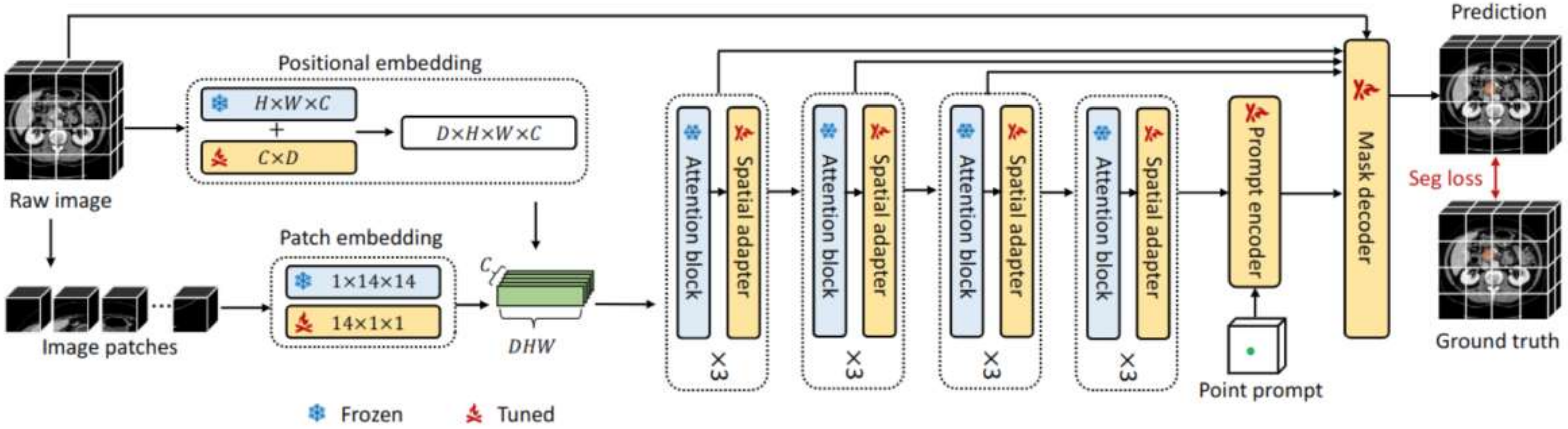
- Segment Anything Model (SAM) [7]
  - Pre-trained on a large-scale 2D dataset of natural images
  - Works well on out-of-domain data when fine-tuned
- However:
  - Can it be applied to 3D medical data?
  - Usually applied slice by slice (axial)
    - Extremely poor results
    - No spatial coherence in predictions
    - → Better: 3D convolutions!

3D MRI Image of the brain viewed from 3 different axes

Image taken from: https://submissions.mirasmart.com/ISMRM2022/itinerary/Files/PDFFiles/1860.html

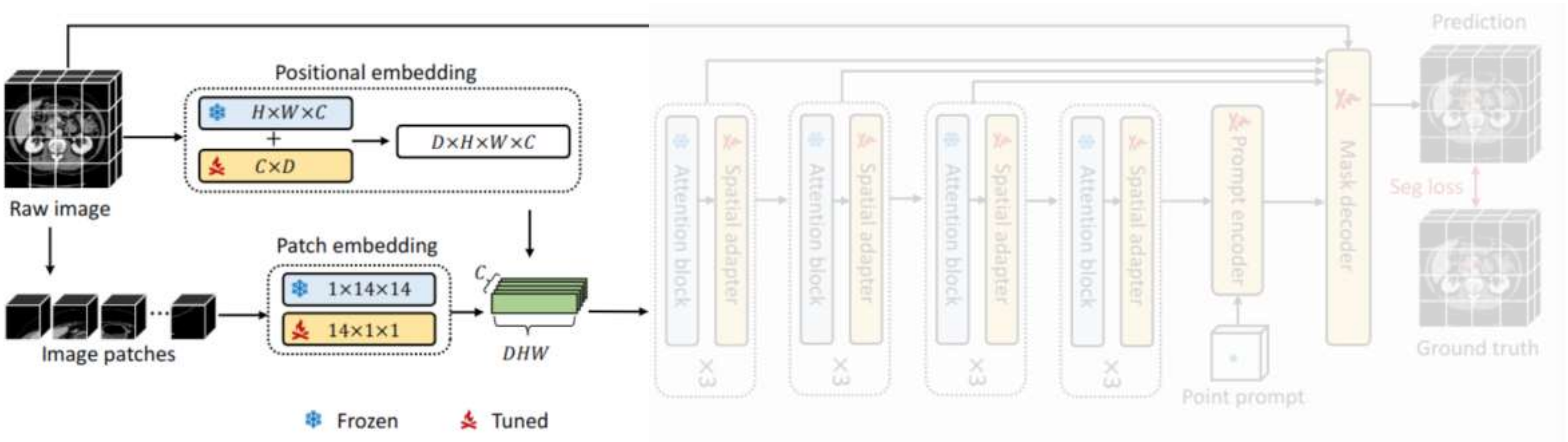
Adapters – Example for 2D → 3D Segmentation [4]

- Adapters at multiple locations



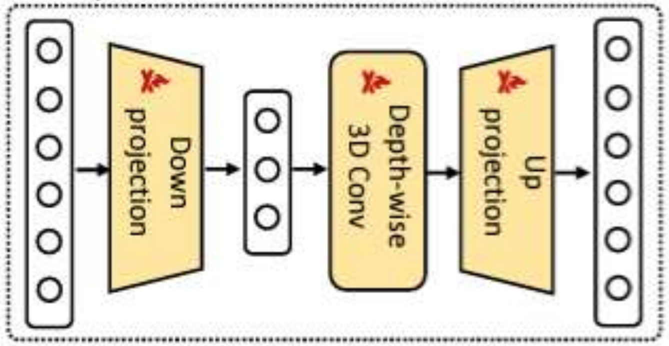
Adapters – Example for 2D → 3D Segmentation [4]

- Adapters at multiple locations
  - **Positional embeddings** → Extend lookup table with depth
  - **Patch embeddings** → Use pre-trained 14x14 2D convolution as 1x14x14 3D convolution
    - Extend with 14x1x1 depth-wise convolution to approximate 14x14x14 3D convolution

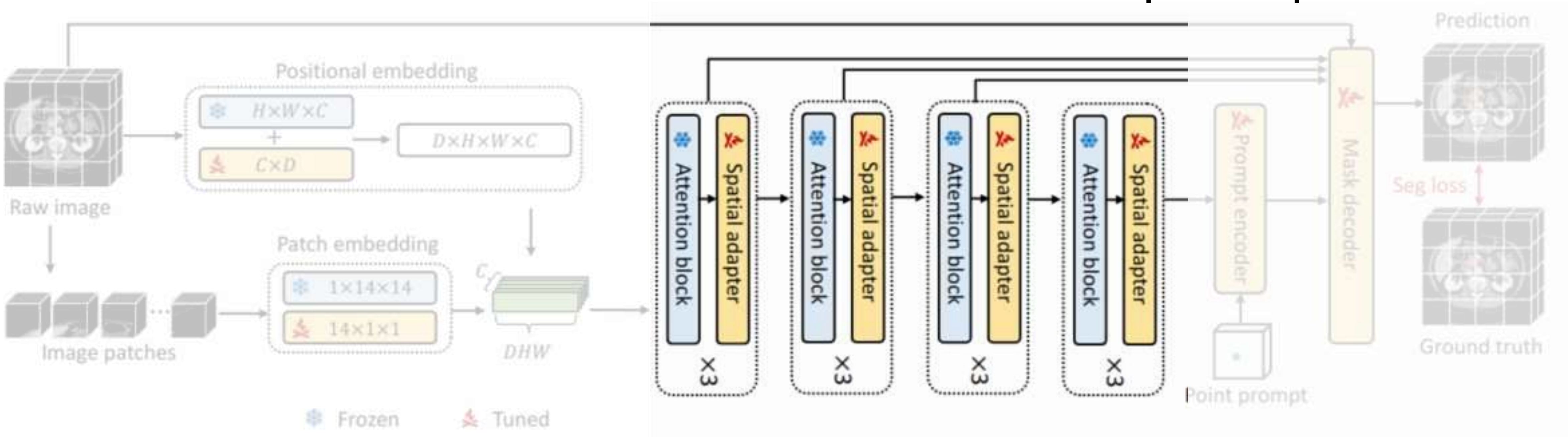


Adapters – Example for 2D → 3D Segmentation [4]

- Adapters at multiple locations
  - **Spatial Adapter**
    - Additional depth-wise 3D convolution before up-projection
    - Adapters can learn 3D spatial information

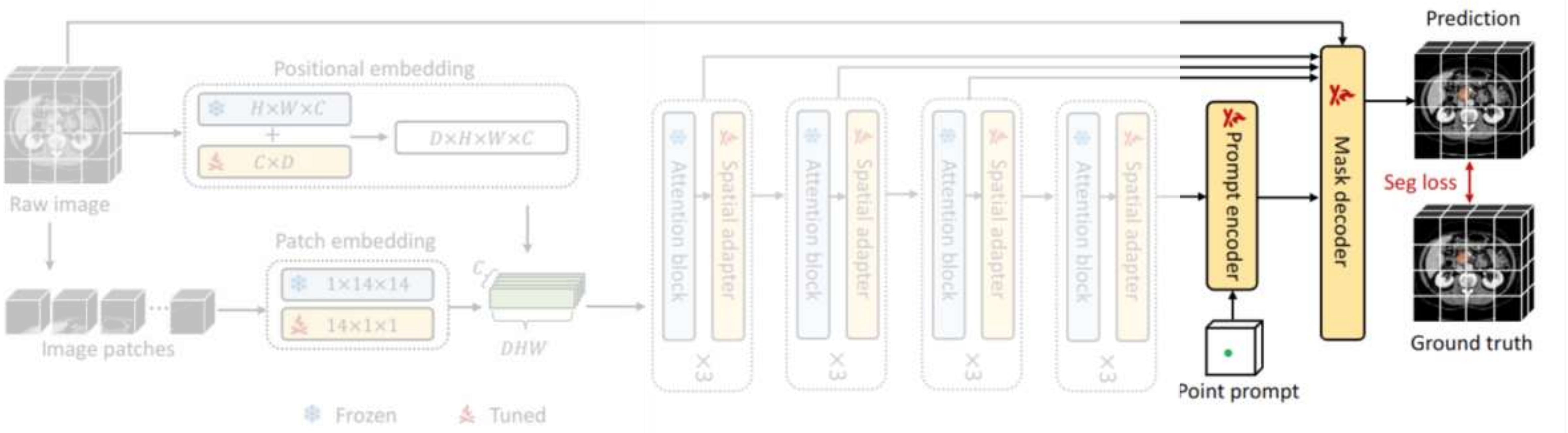


Spatial adapter



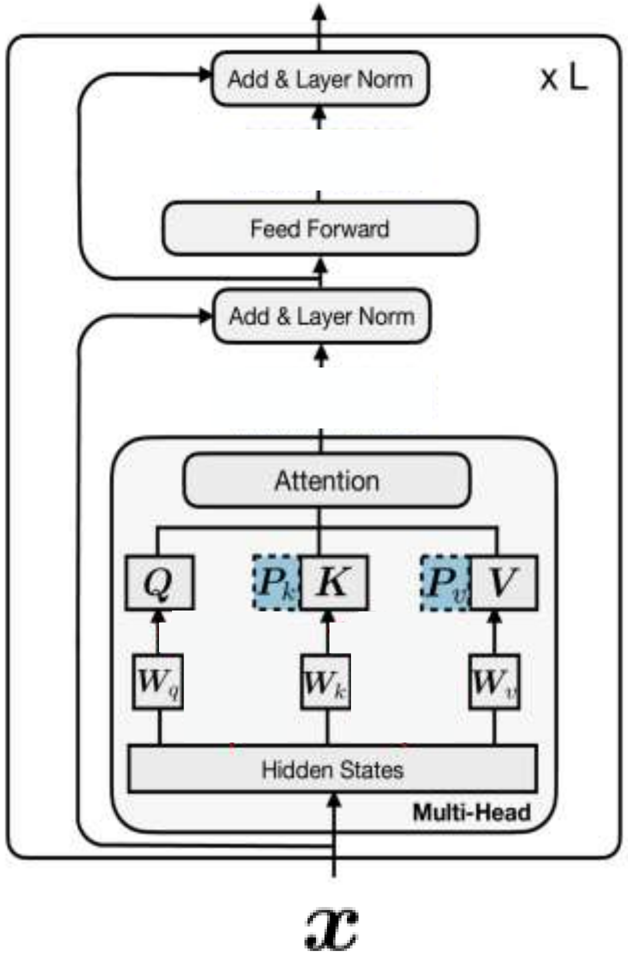
Adapters – Example for 2D → 3D Segmentation [4]

- Adapters at multiple locations
  - **Mask Decoder** and **Point Encoder** are trained from scratch with 3D convolutions
  - They are already lightweight and have few parameters



PREFIX TUNING

Prefix Tuning [5]

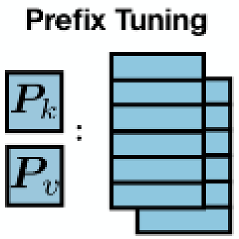


$$\mathrm{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}$$

$$\mathrm{MHA}(\boldsymbol{x}) = \mathrm{Concat}(\mathrm{head}_1, \cdots, \mathrm{head}_h)\boldsymbol{W}_o, \ \cancel{\mathrm{head}_i = \mathrm{Attn}(\boldsymbol{x}\boldsymbol{W}_q^{(i)}, \boldsymbol{x}\boldsymbol{W}_k^{(i)}, \boldsymbol{x}\boldsymbol{W}_v^{(i)})}; \quad \boldsymbol{x} \in \mathbb{R}^d$$

$$\mathrm{head}_i = \mathrm{Attn}(\boldsymbol{x}\boldsymbol{W}_q^{(i)}, \mathrm{concat}(\boldsymbol{P}_k^{(i)}, \boldsymbol{x}\boldsymbol{W}_k^{(i)}), \mathrm{concat}(\boldsymbol{P}_v^{(i)}, \boldsymbol{x}\boldsymbol{W}_v^{(i)})) \quad \boldsymbol{x} \in \mathbb{R}^d$$

$$\mathrm{FFN}(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$



$$\boldsymbol{P}_k, \boldsymbol{P}_v \in \mathbb{R}^{l \times d}$$

$$\boldsymbol{W}_q^{(i)}, \boldsymbol{W}_k^{(i)}, \boldsymbol{W}_v^{(i)} \in \mathbb{R}^{d \times d_h}$$

$$\boldsymbol{W}_o \in \mathbb{R}^{d \times d}$$

$$\boldsymbol{W}_1 \in \mathbb{R}^{d \times d_m}, \boldsymbol{W}_2 \in \mathbb{R}^{d_m \times d}$$

Prefix Tuning [5]

- Only update the concatenated prefixes
- Intuition: Let the model learn how to “steer” itself
  - Prefixes encode task-specific knowledge
- Why not learn which prompt works best (prompt engineering)?

Prefix Tuning [5]

- Why not learn which prompt works best (prompt engineering)?
  - Optimization over discrete space is not flexible
    - Solution is forced to choose words from the vocabulary
    - Model is only adapted at the input layer

$$w_1, w_2 = \underset{w_1', w_2' \in \text{Vocab}}{\text{argmax}} \; \mathbb{E}_{x,y}[\log P_{\text{GPT2}}(y \mid w_1', w_2', x)]$$

Optimal prompts (prompt engineering)

Equation taken from: https://medium.com/@musicalchemist/prefix-tuning-lightweight-adaptation-of-large-language-models-for-customized-natural-language-a8a93165c132

Prefix Tuning [5]

- Why not learn which prompt works best (prompt engineering)?
  - Optimization over discrete space is not flexible
    - Solution is forced to choose words from the vocabulary
    - Model is only adapted at the input layer

$$w_1, w_2 = \underset{w'_1, w'_2 \in \text{Vocab}}{\operatorname{argmax}} \mathbb{E}_{x,y}[\log P_{\text{GPT2}}(y \mid w'_1, w'_2, x)]$$

Optimal prompts (prompt engineering)

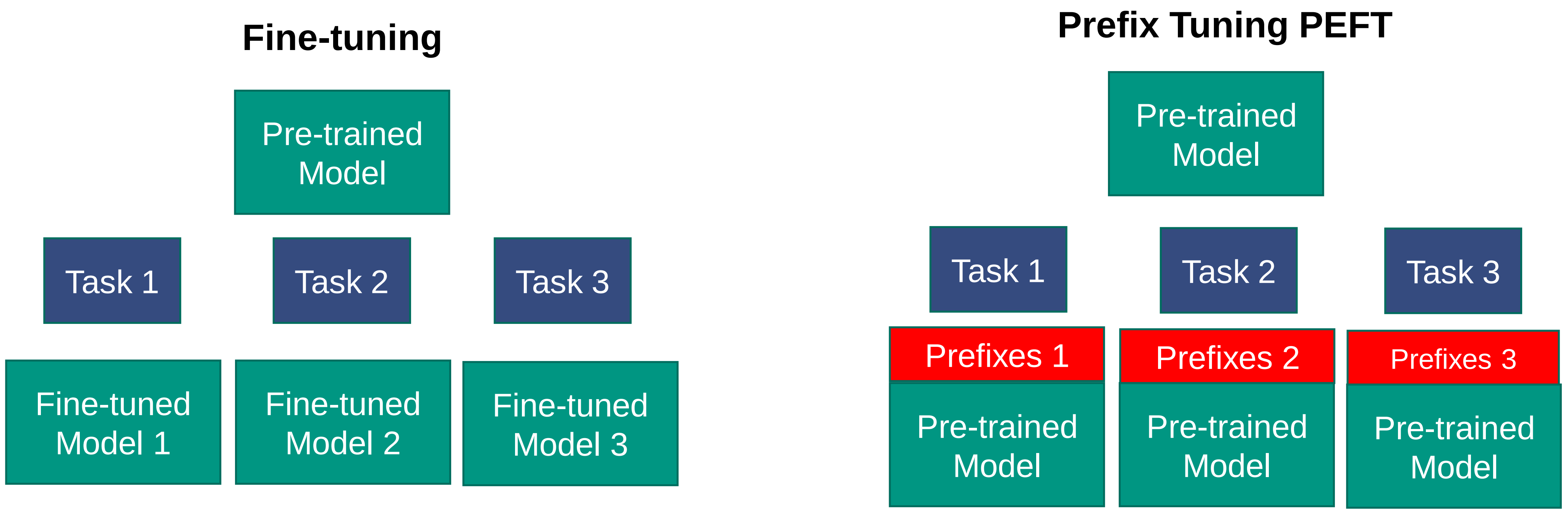
- Prefix tuning:
  - Optimization over continuous variables directly with gradient descent
    - Solution is flexible and task-specific
    - Model is adapted in all layers

$$p_1, p_2 = \underset{p'_1, p'_2 \in \mathbb{R}^{l \times d}}{\operatorname{argmax}} \mathbb{E}_{x,y}[\log P_{\text{GPT2}}(y \mid p'_1, p'_2, x)]$$

Equations taken from: https://medium.com/@musicalchemist/prefix-tuning-lightweight-adaptation-of-large-language-models-for-customized-natural-language-a8a93165c132

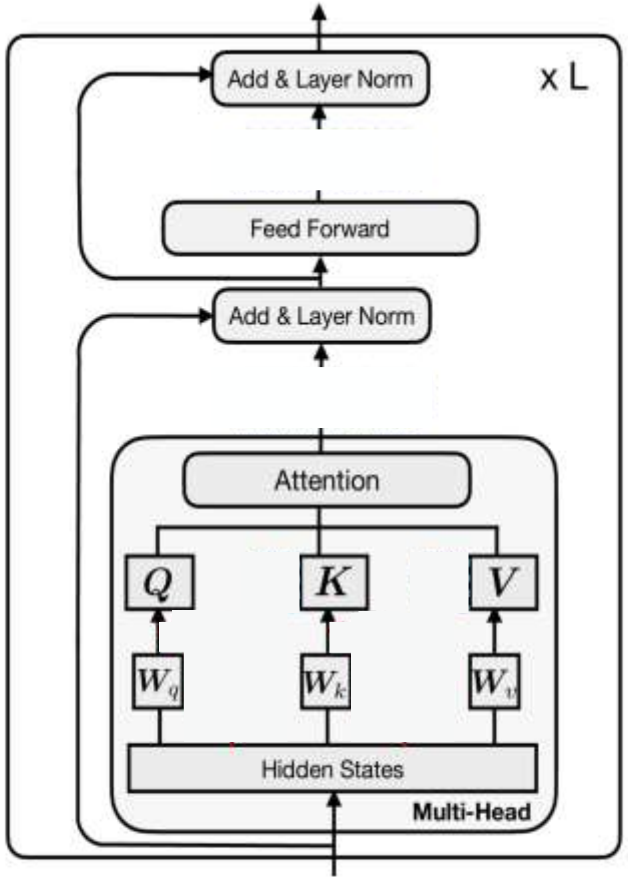
Prefix Tuning [5]

- Adds additional context to the learned representations in the sequence
- Pre-trained model is unchanged
- New tasks → New prefixes!
  - Very similar to adapters but usually requires fewer parameters



PROMPT TUNING

Prompt Tuning (soft prompts) [10]



$\mathbf{concat}(x, p)$

$$\mathrm{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V}$$

$$\mathrm{MHA}(\boldsymbol{x}) = \mathrm{Concat}(\mathrm{head}_1, \cdots, \mathrm{head}_\mathrm{h})\boldsymbol{W}_o, \ \mathrm{head}_\mathrm{i} = \mathrm{Attn}(\boldsymbol{x}\boldsymbol{W}_q^{(i)}, \boldsymbol{x}\boldsymbol{W}_k^{(i)}, \boldsymbol{x}\boldsymbol{W}_v^{(i)}), \ \cancel{\boldsymbol{x} \in \mathbb{R}^d}$$

$$\mathrm{FFN}(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$

$$x = \mathbf{concat}(x, p) \in \mathbb{R}^{d+l}$$

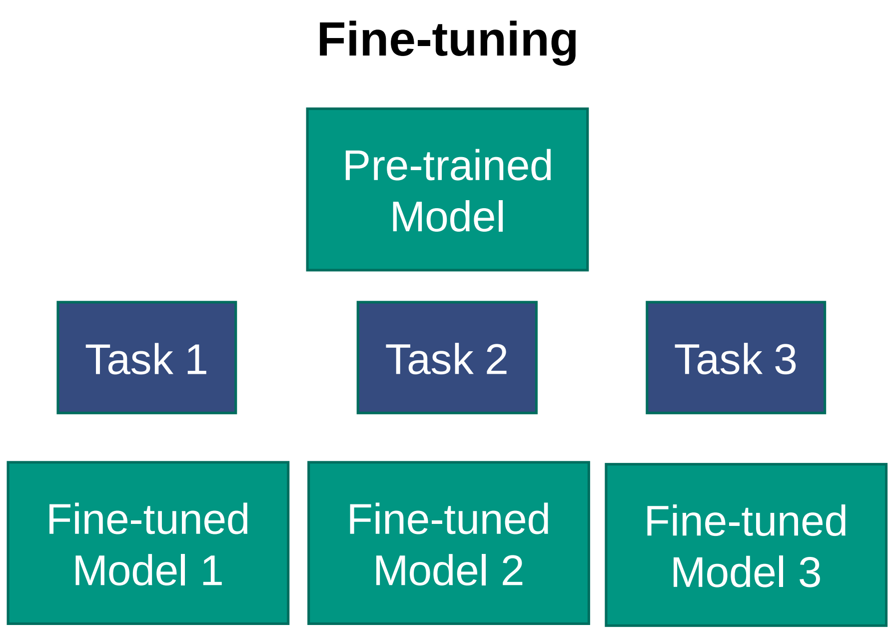
$$\boldsymbol{W}_q^{(i)}, \boldsymbol{W}_k^{(i)}, \boldsymbol{W}_v^{(i)} \in \mathbb{R}^{d \times d_h}$$

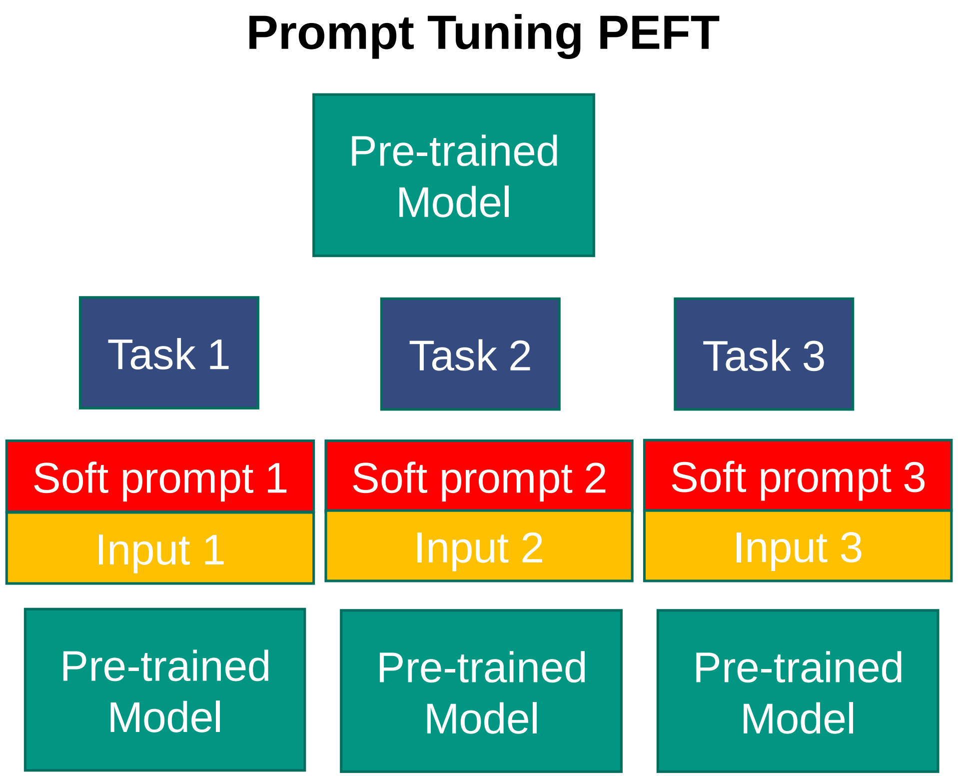
$$\boldsymbol{W}_o \in \mathbb{R}^{d \times d}$$

$$\boldsymbol{W}_1 \in \mathbb{R}^{d \times d_m}, \boldsymbol{W}_2 \in \mathbb{R}^{d_m \times d}$$

Prompt Tuning (soft prompts) [10]

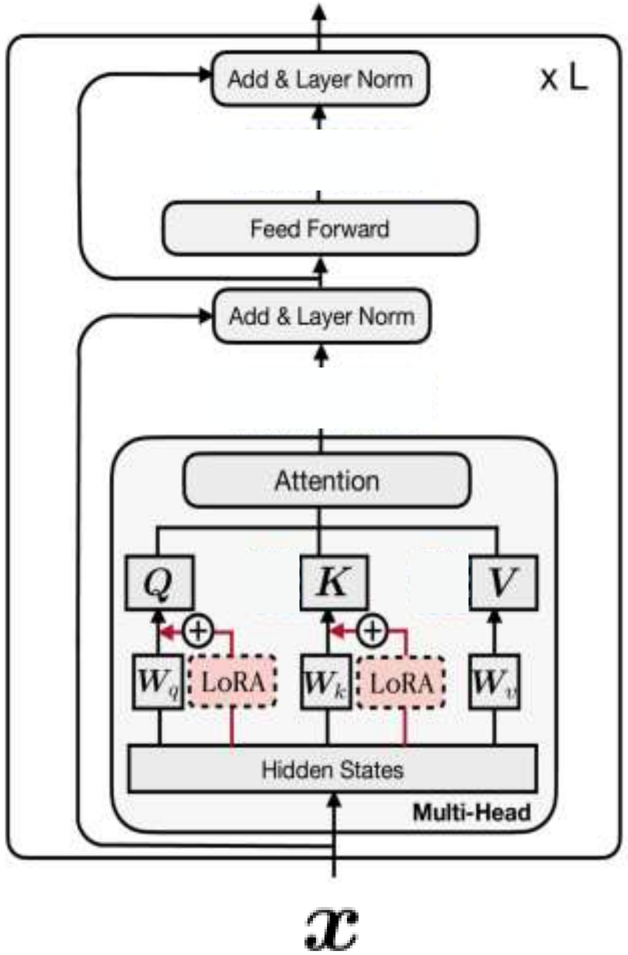
- Adds additional context to the **inputs** in the sequence
  - Instead of the intermediate representations
- Pre-trained model is unchanged
- Similar to prefix tuning but only at input level
- Soft prompts → Continuous values





LOW RANK ADAPTATION (LORA)

Low Rank Adaptation (LoRA) [8]



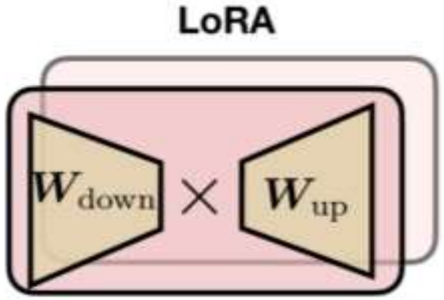
$$\mathrm{Attn}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}$$

$$\mathrm{MHA}(\boldsymbol{x}) = \mathrm{Concat}(\mathrm{head}_1,\cdots,\mathrm{head}_\mathrm{h})\boldsymbol{W}_o,\ \mathrm{head}_\mathrm{i} = \mathrm{Attn}(\boldsymbol{x}\boldsymbol{W}_q^{(i)}, \boldsymbol{x}\boldsymbol{W}_k^{(i)}, \boldsymbol{x}\boldsymbol{W}_v^{(i)}),\ \boldsymbol{x}\in\mathbb{R}^d$$

$$\mathrm{FFN}(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$

$$\boldsymbol{W}_q^{(i)} + \Delta\boldsymbol{W}_q^{(i)} \approx \boldsymbol{W}_q^{(i)} + \boldsymbol{W}_{q-up}^{(i)} \cdot \boldsymbol{W}_{q-down}^{(i)}$$

$$\boldsymbol{W}_k^{(i)} + \Delta\boldsymbol{W}_k^{(i)} \approx \boldsymbol{W}_k^{(i)} + \boldsymbol{W}_{k-up}^{(i)} \cdot \boldsymbol{W}_{k-down}^{(i)}$$



$$\Delta\boldsymbol{W}_q^{(i)}, \Delta\boldsymbol{W}_k^{(i)} \in \mathbb{R}^{d\times d_h}$$

$$\boldsymbol{W}_{q-up}^{(i)}, \boldsymbol{W}_{k-up}^{(i)} \in \mathbb{R}^{d\times r}$$

$$\boldsymbol{W}_{q-down}^{(i)}, \boldsymbol{W}_{k-down}^{(i)} \in \mathbb{R}^{r\times d_h}$$

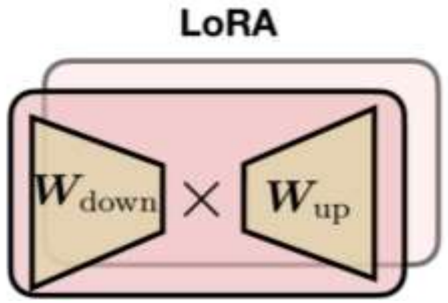
$$\boldsymbol{W}_q^{(i)}, \boldsymbol{W}_k^{(i)}, \boldsymbol{W}_v^{(i)} \in \mathbb{R}^{d\times d_h}$$

$$\boldsymbol{W}_o \in \mathbb{R}^{d\times d}$$

$$\boldsymbol{W}_1 \in \mathbb{R}^{d\times d_m}, \boldsymbol{W}_2 \in \mathbb{R}^{d_m\times d}$$

Low Rank Adaptation (LoRA) [8]

- Intuition behind LoRA
  - Pre-trained models already have good features
  - Gradient updates are sparse on new tasks
    - The model has only a little to learn to adapt to the new task
  - The "update matrices" $\Delta \boldsymbol{W}_q^{(i)}, \Delta \boldsymbol{W}_k^{(i)}$ have an inherently low rank
- Reparameterization of update matrices
  - $\Delta \boldsymbol{W}_q^{(i)}, \Delta \boldsymbol{W}_k^{(i)} \in \mathbb{R}^{d \times d_h}$
  - Downscale: $\boldsymbol{W}_{q-down}^{(i)}, \boldsymbol{W}_{k-down}^{(i)} \in \mathbb{R}^{r \times d_h}$
  - Upscale: $\boldsymbol{W}_{q-up}^{(i)}, \boldsymbol{W}_{k-up}^{(i)} \in \mathbb{R}^{d \times r}$
  - Low-rank $r << \min(d_h, d)$



$$\boldsymbol{W}_q^{(i)} + \Delta \boldsymbol{W}_q^{(i)} \approx \boldsymbol{W}_q^{(i)} + \boldsymbol{W}_{q-up}^{(i)} \cdot \boldsymbol{W}_{q-down}^{(i)}$$

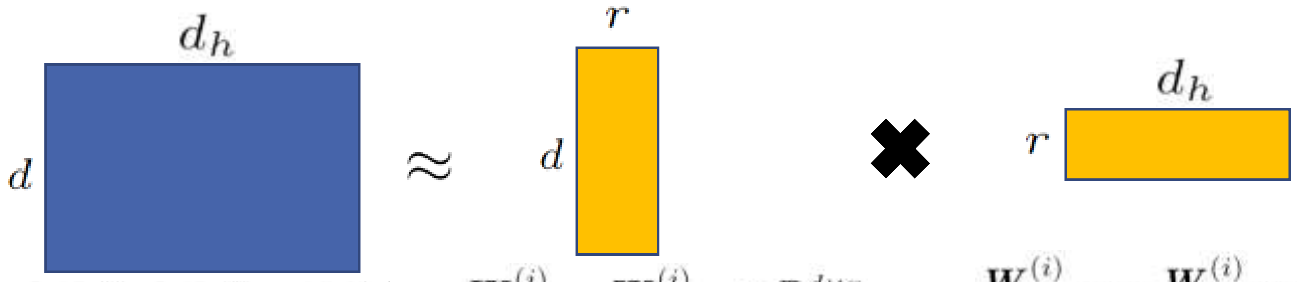
$$\boldsymbol{W}_k^{(i)} + \Delta \boldsymbol{W}_k^{(i)} \approx \boldsymbol{W}_k^{(i)} + \boldsymbol{W}_{k-up}^{(i)} \cdot \boldsymbol{W}_{k-down}^{(i)}$$

$$\boldsymbol{W}_q^{(i)}, \boldsymbol{W}_k^{(i)}, \boldsymbol{W}_v^{(i)} \in \mathbb{R}^{d \times d_h}$$

$$\Delta \boldsymbol{W}_q^{(i)}, \Delta \boldsymbol{W}_k^{(i)} \in \mathbb{R}^{d \times d_h}$$

$$\boldsymbol{W}_{q-up}^{(i)}, \boldsymbol{W}_{k-up}^{(i)} \in \mathbb{R}^{d \times r}$$

$$\boldsymbol{W}_{q-down}^{(i)}, \boldsymbol{W}_{k-down}^{(i)} \in \mathbb{R}^{r \times d_h}$$



$\Delta \boldsymbol{W}_q^{(i)}, \Delta \boldsymbol{W}_k^{(i)} \in \mathbb{R}^{d \times d_h}$ $\boldsymbol{W}_{q-up}^{(i)}, \boldsymbol{W}_{k-up}^{(i)} \in \mathbb{R}^{d \times r}$ $\boldsymbol{W}_{q-down}^{(i)}, \boldsymbol{W}_{k-down}^{(i)} \in \mathbb{R}^{r \times d_h}$

Low Rank Adaptation (LoRA) [8]

- Reparameterization of update matrices
  - During inference → Just **add the update matrices** to the pre-trained model

$$\boldsymbol{W}_q^{(i)} \leftarrow \boldsymbol{W}_q^{(i)} + \boldsymbol{W}_{q-up}^{(i)} \cdot \boldsymbol{W}_{q-down}^{(i)}$$
$$\boldsymbol{W}_k^{(i)} \leftarrow \boldsymbol{W}_k^{(i)} + \boldsymbol{W}_{k-up}^{(i)} \cdot \boldsymbol{W}_{k-down}^{(i)}$$

  - No additional parameters → No latency
    - Adapters and Prefix tuning require additional parameters

Low Rank Adaptation (LoRA) [8]

- Reparameterization of update matrices
  - During inference → Just **add the update matrices** to the pre-trained model
  - Update matrices for different tasks can be combined by addition (Example: DreamBooth[9])

Dog in a big red bucket

$$W_q^{(i)} \leftarrow W_q^{(i)} + W_{q-up}^{(i)} \cdot W_{q-down}^{(i)}$$
$$W_k^{(i)} \leftarrow W_k^{(i)} + W_{k-up}^{(i)} \cdot W_{k-down}^{(i)}$$

“Dog” LoRA update matrices

Superman, close-up portrait

$$W_q^{(i)} \leftarrow W_q^{(i)} + W_{q-up}^{(i)} \cdot W_{q-down}^{(i)}$$
$$W_k^{(i)} \leftarrow W_k^{(i)} + W_{k-up}^{(i)} \cdot W_{k-down}^{(i)}$$

“Toy” LoRA update matrices

Dog, close-up portrait

$$W_q^{(i)} \leftarrow W_q^{(i)} + W_{q-up}^{(i)} \cdot W_{q-down}^{(i)} + W_{q-up}^{(i)} \cdot W_{q-down}^{(i)}$$
$$W_k^{(i)} \leftarrow W_k^{(i)} + W_{k-up}^{(i)} \cdot W_{k-down}^{(i)} + W_{k-up}^{(i)} \cdot W_{k-down}^{(i)}$$

“Dog” + “Toy” LoRA update matrices

COMPARISON OF PEFT APPROACHES

Comparison of Fine-tuning Approaches

- Full fine-tuning
  - Pros
    - Completely adapts model to the new task – best performance given enough data
  - Cons
    - Catastrophic forgetting as many parameters are updated
    - Computationally infeasible for large models
    - Storage inefficient
    - Slow training

- PEFT
  - Pros
    - Computationally efficient: only a small portion of the parameters is updated
    - Storage efficient
    - Fast training on consumer GPUs
  - Cons
    - Requires careful engineering for a specific task
      - Where to put adapters
      - How to set r in LoRA
      - How large should the prefix be in Prefix tuning, etc.

Comparison of Fine-tuning Approaches

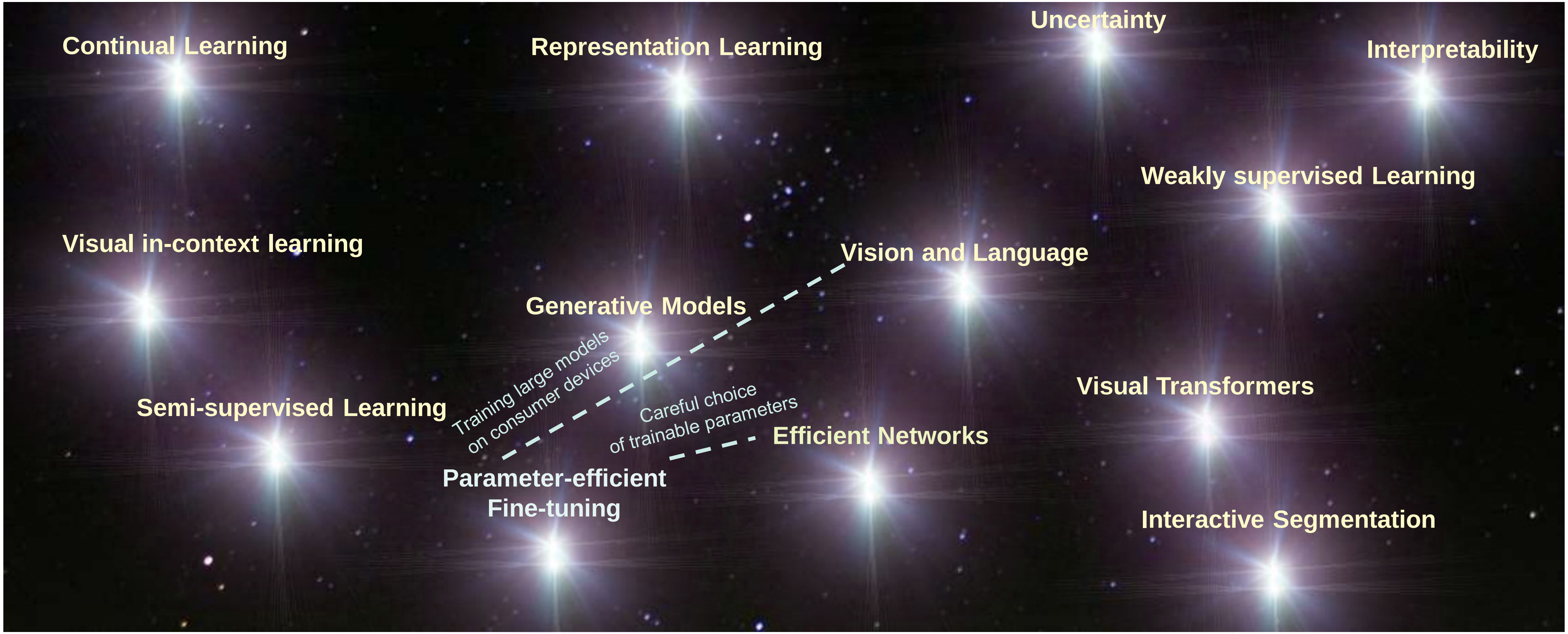
- Adapters and Prefix and Prompt Tuning
  - Pros
    - Can “transform” the model to fit another domain
      - Example: 2D → 3D inputs
  - Cons
    - Inference latency
      - Adapter and additional prefix parameters make the model larger
      - Often not parallelizable
- LoRA
  - Pros
    - No latency – just add the learned weights to the pre-trained model during inference
    - Usually better performant
  - Cons
    - Model architecture stays the same → Cannot be applied on domains from other dimensions

Conclusion: Parameter-Efficient Fine-Tuning

- PEFT allows to train huge models on consumer GPUs with little performance loss

- Different ways to achieve this:
  - Adapters, LoRA, Prefix and Prompt tuning, Partial Fine-tuning, Full Fine-tuning
  - Choice depends on the task at hand

Constellations in Efficient Networks



References [Efficient Networks]

- [1] West, Jeremy; Ventura, Dan; Warnick, Sean (2007). "Spring Research Presentation: A Theoretical Foundation for Inductive Transfer". Brigham Young University, College of Physical and Mathematical Sciences.
- [2] Peng, Xingchao, et al. "Visda: The visual domain adaptation challenge." *arXiv preprint arXiv:1710.06924* (2017).
- [3] Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." *IEEE Transactions on knowledge and data engineering* 22.10 (2009): 1345-1359.
- [4] Ringwald, Tobias, and Rainer Stiefelhagen. "Adaptiope: A modern benchmark for unsupervised domain adaptation." *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021.
- [5] Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size." *arXiv preprint arXiv:1602.07360* (2016).
- [6] A Comprehensive Introduction to Different Types of Convolutions in Deep Learning, from https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
- [7] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012): 1097-1105.
- [8] Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.
- [9] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." *arXiv preprint arXiv:1704.04861* (2017).
- [10] Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.
- [11] Ganin, Yaroslav, et al. "Domain-adversarial training of neural networks." *The journal of machine learning research* 17.1 (2016): 2096-2030.
- [12] Chang, Woong-Gi, et al. "Domain-specific batch normalization for unsupervised domain adaptation." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.
- [13] Micikevicius, Paulius, et al. "Mixed precision training." *arXiv preprint arXiv:1710.03740* (2017).
- [14] Zhu, Feng, et al. "Towards unified int8 training for convolutional neural network." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.
- [15] Intel® oneAPI Deep Neural Network Library Developer Guide and Reference Developer Guide and Reference, https://software.intel.com/content/www/us/en/develop/documentation/onednn-developer-guide-and-reference/top/programming-model/inference-and-training-aspects/int8-inference.html
- [16] Rastegari, Mohammad, et al. "Xnor-net: Imagenet classification using binary convolutional neural networks." *European conference on computer vision*. Springer, Cham, 2016.
- [17] Luo, Jian-Hao, Jianxin Wu, and Weiyao Lin. "Thinet: A filter level pruning method for deep neural network compression." *Proceedings of the IEEE international conference on computer vision*. 2017.
- [18] Ringwald, Tobias, et al. "UAV-Net: A fast aerial vehicle detector for mobile platforms." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019.
- [19] Hoffman, Judy, et al. "Cycada: Cycle-consistent adversarial domain adaptation." *International conference on machine learning*. PMLR, 2018.
- [20] Sun, Baochen, and Kate Saenko. "Deep coral: Correlation alignment for deep domain adaptation." *European conference on computer vision*. Springer, Cham, 2016.
- [21] You, Kaichao, et al. "Universal domain adaptation." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.

References [PEFT]

- [1] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).
- [2] He, Junxian, et al. "Towards a unified view of parameter-efficient transfer learning." *arXiv preprint arXiv:2110.04366* (2021).
- [3] Houlsby, Neil, et al. "Parameter-efficient transfer learning for NLP." *International Conference on Machine Learning*. PMLR, 2019.
- [4] Gong, Shizhan, et al. "3DSAM-adapter: Holistic Adaptation of SAM from 2D to 3D for Promptable Medical Image Segmentation." *arXiv preprint arXiv:2306.13465* (2023).
- [5] Li, Xiang Lisa, and Percy Liang. "Prefix-Tuning: Optimizing Continuous Prompts for Generation." *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2021.
- [6] Drakopoulos, Fotis, and Nikos P. Chrisochoides. "Accurate and fast deformable medical image registration for brain tumor resection using image-guided neurosurgery." *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization* 4.2 (2016): 112-126.
- [7] Kirillov, Alexander, et al. "Segment anything." *arXiv preprint arXiv:2304.02643* (2023).
- [8] Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).
- [9] Ruiz, Nataniel, et al. "Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023.
- [10] Lester, Brian, Rami Al-Rfou, and Noah Constant. "The power of scale for parameter-efficient prompt tuning." *arXiv preprint arXiv:2104.08691* (2021).