MERCon 2025
MORATUWA ENGINEERING RESEARCH CONFERENCE

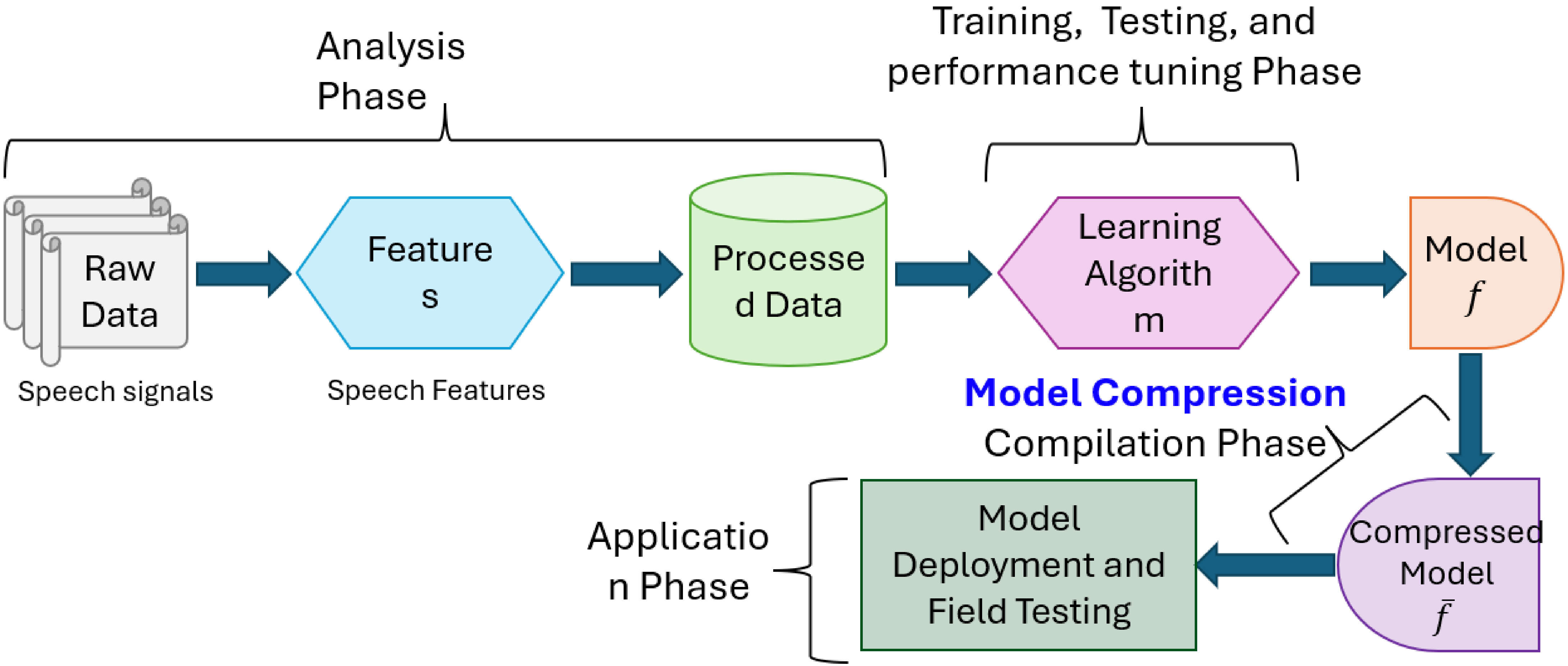
# TinyML: A Compact Revolution in Engineering AI

## Session 4: Parameter-Efficient Architeures for TinyML

Mr. Sanka Mohottala
Sri Lanka Institute of Information Technology
sanka.mo@slit.lk

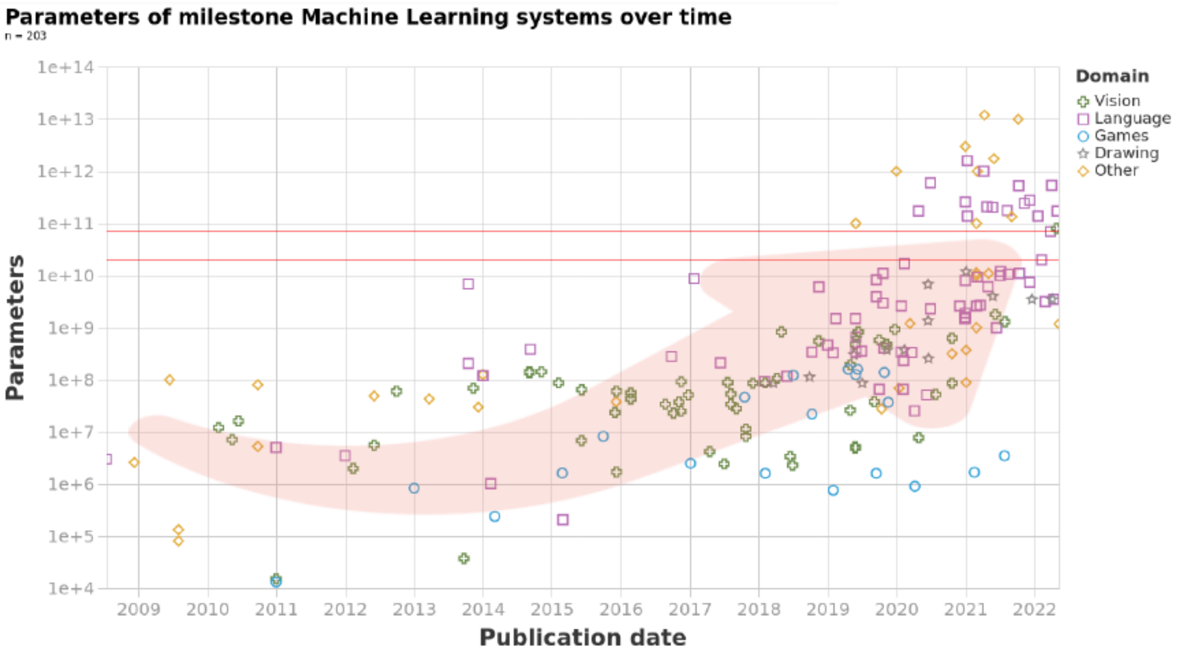
SLIIT UNI
THE KNOWLEDGE UNIVERSITY

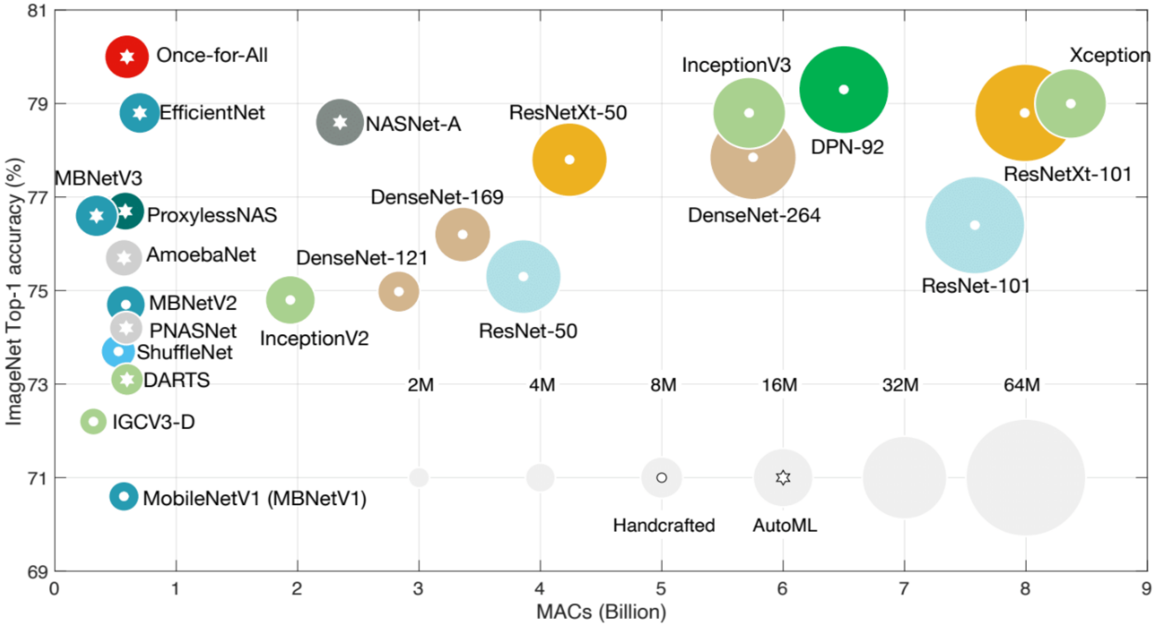
# A Schematic View of TinyML and Its Phases



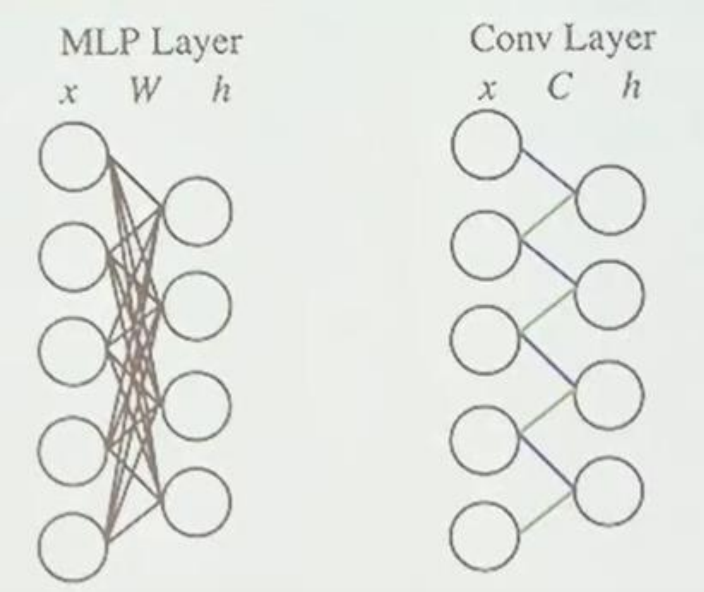
# Model Evolution

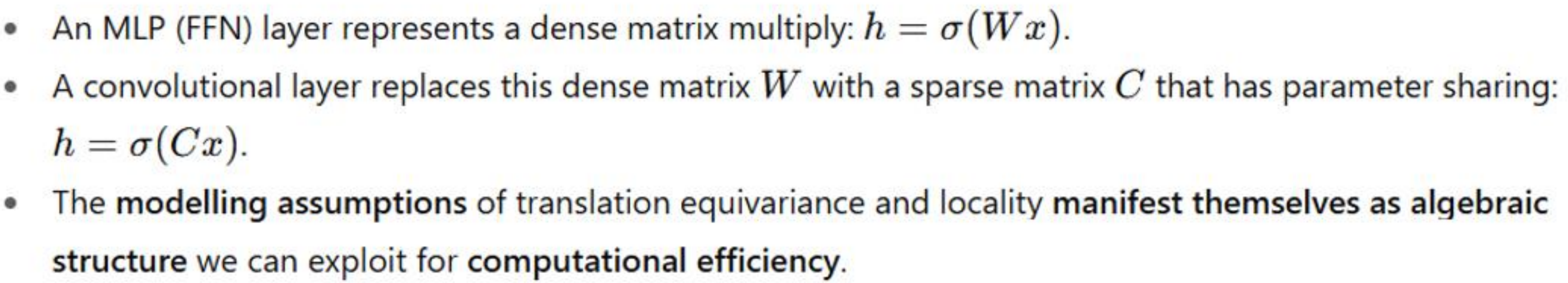
- The model parameters increase exponentially!
- We want model performance well while having fewer parameters





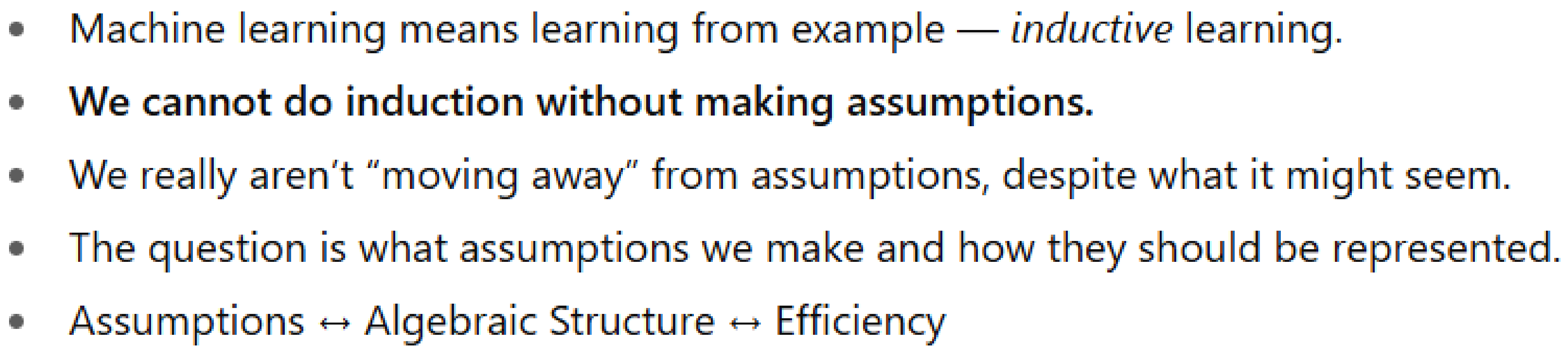
# ANN vs CNN





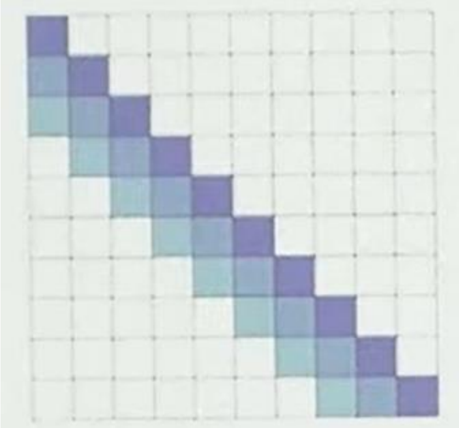
- An MLP (FFN) layer represents a dense matrix multiply: $h = \sigma(Wx)$.
- A convolutional layer replaces this dense matrix $W$ with a sparse matrix $C$ that has parameter sharing: $h = \sigma(Cx)$.
- The **modelling assumptions** of translation equivariance and locality **manifest themselves as algebraic structure** we can exploit for **computational efficiency**.

# We Can’t Get Away from Assumptions



- Machine learning means learning from example — *inductive* learning.
- **We cannot do induction without making assumptions.**
- We really aren’t “moving away” from assumptions, despite what it might seem.
- The question is what assumptions we make and how they should be represented.
- Assumptions ↔ Algebraic Structure ↔ Efficiency

# Convolutional & Toeplitz Structure

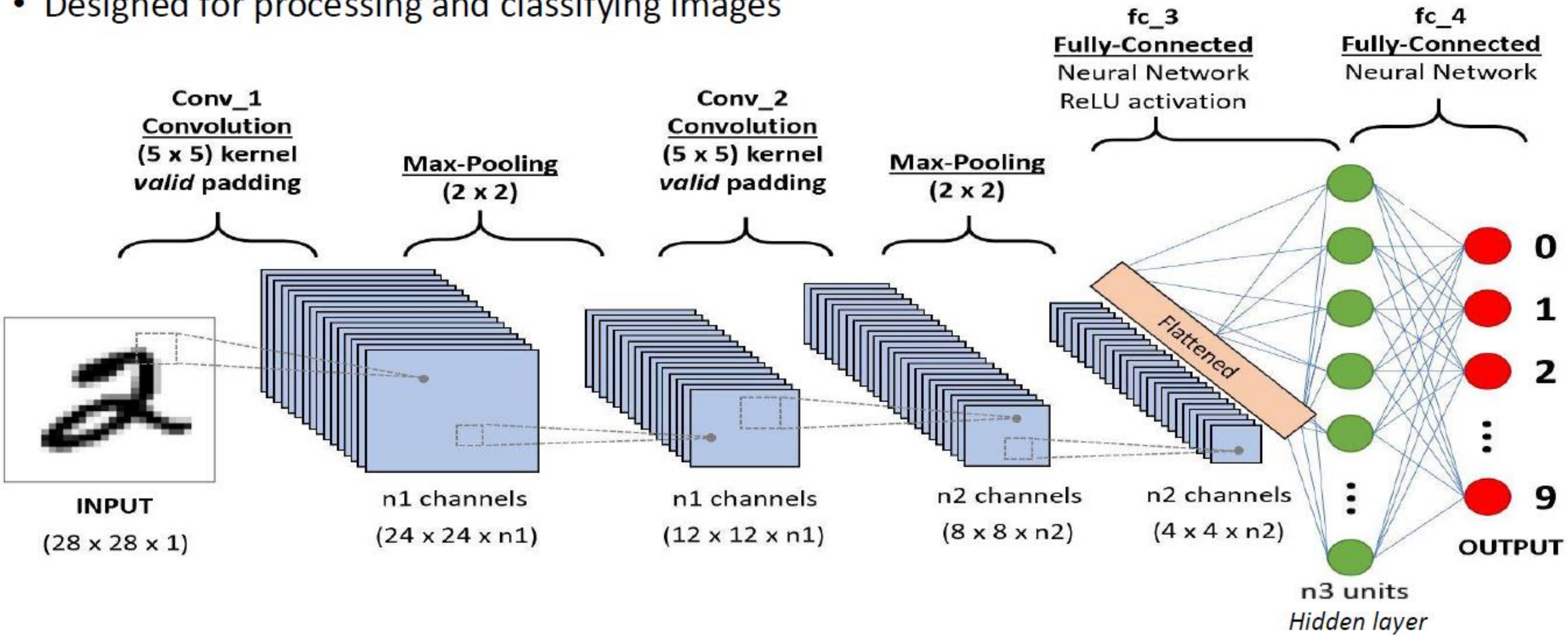


$$\begin{bmatrix} a_1 & a_2 & \dots & a_{d-1} & a_d \\ a_2 & a_1 & \dots & a_{d-2} & a_{d-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{d-1} & a_{d-2} & \dots & a_1 & a_2 \\ a_d & a_{d-1} & \dots & a_2 & a_1 \end{bmatrix}$$

- Multiplication with a $p \times p$ convolution kernel requires $\mathcal{O}(pd)$ flops. Each parameter is used $\mathcal{O}(d)$ times.
- Convolution matrices are special cases of Toeplitz matrices, which are constant on their diagonals.
- They frequently arise in systems with translational symmetry, such as images and time-series.
- **How much memory does it take to represent the symmetric Toeplitz matrix above?**

# Standard Convolutional Neural Networks (CNNs)

- Designed for processing and classifying images

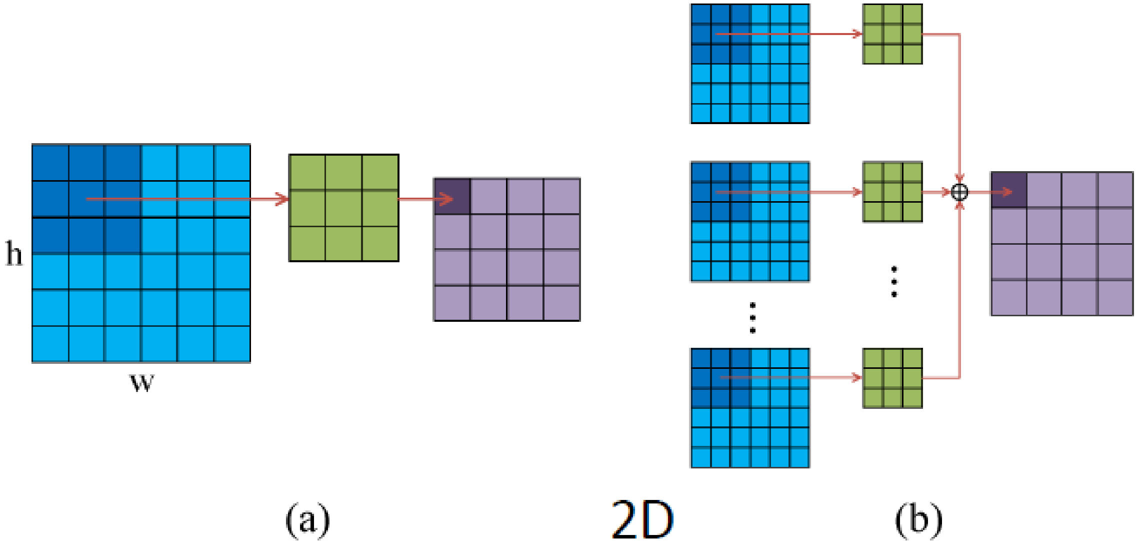


**Components of CNN:** Input layer; Convolutional layers; Pooling layers; Fully connected NNs; Output layer

# Standard Convolution

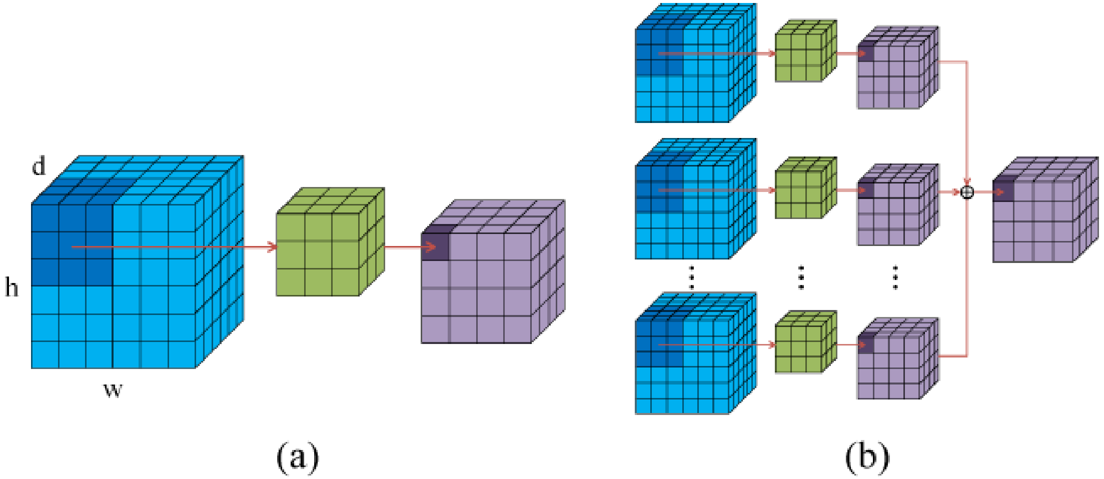
- Standard convolution slides over input data performing element-wise multiplication with the part of the input it is on, then summing the results into an output.

- Example: 2-D
- Input Feature Map
  - 6 × 6 × 1
  - Width × Height × Channel
- Kernel (Filter)
  - 3 × 3 × 1



# Standard Convolution

- Standard convolution slides over input data performing element-wise multiplication with the part of the input it is on, then summing the results into an output.
- Example: 3-D
- Input Feature Map
  - 6 × 6 × 6 × 1
  - Width × Height × Depth × Channel
- Kernel (Filter)
  - 3 × 3 × 3 × 1

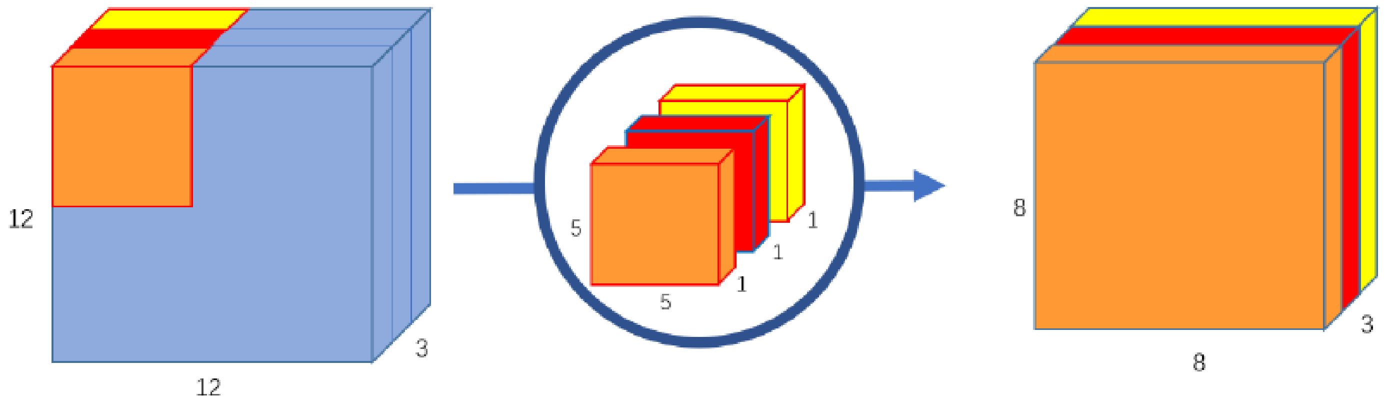


3D

# MobileNet – Depthwise Convolution

- Depthwise Convolution is a type of convolution where we apply a single convolutional filter for each input channel.

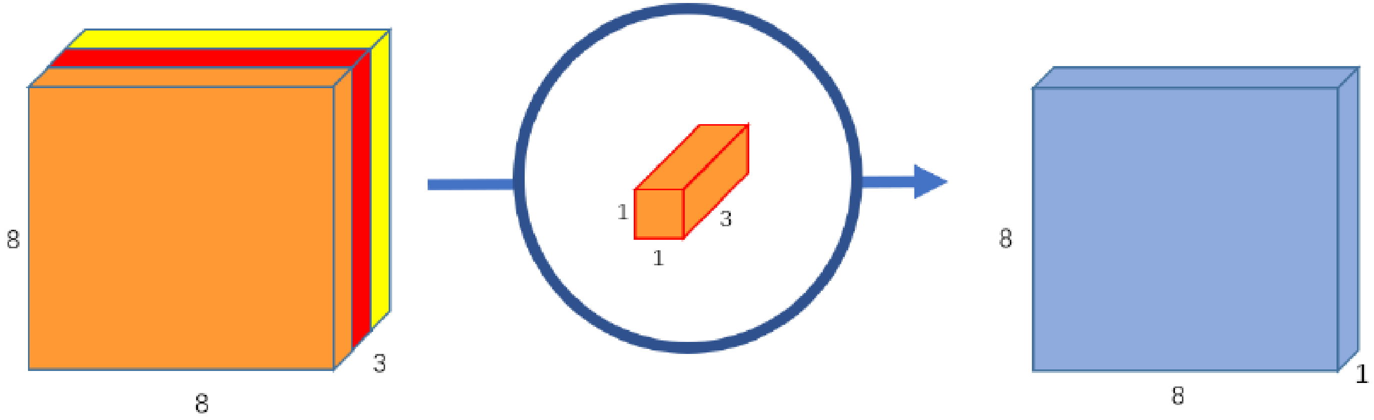
- Depthwise convolutions keep each channel separate.



# MobileNet – Pointwise Convolution

- Pointwise Convolution is a type of convolution that uses a **1x1 kernel**: a kernel that iterates through every single point.

- This kernel has a depth of however many channels the input image has.

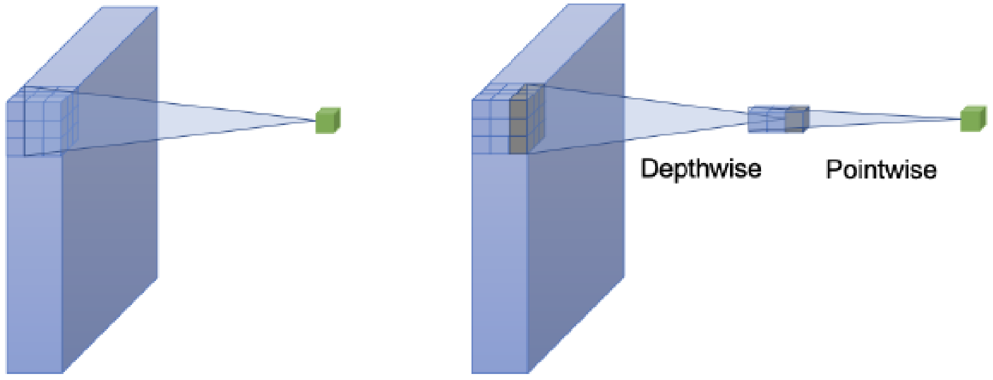


# MobileNet – Cascading Convolution Techniques

**Depthwise + Pointwise = Depthwise Separable Convolution**

A Two-Step-Algorithm:

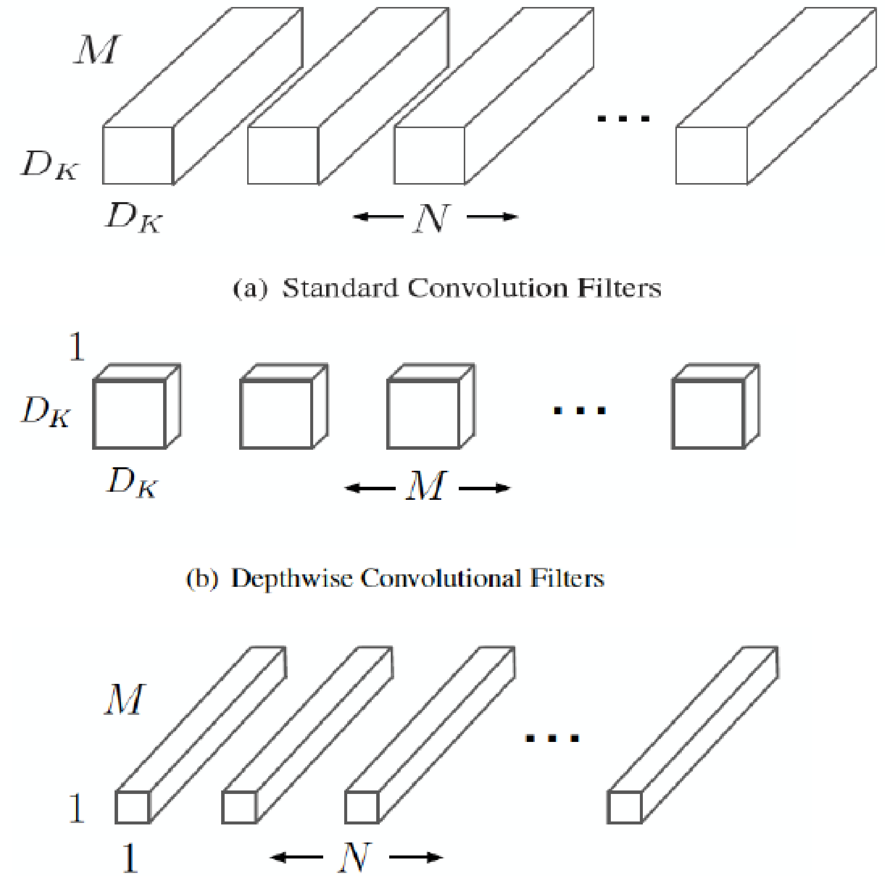
1. Depthwise convolution applies a single convolutional filter per each input channel
2. Pointwise convolution is then used to create a linear combination of the output of the depthwise convolution.



Standard Convolution

Depthwise Separable Convolution

# MobileNet – Cascading Convolution Techniques



(a) Standard Convolution Filters

(b) Depthwise Convolutional Filters

(c) $1 \times 1$ Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

$$\left.\begin{array}{c} D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \\ \\ D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \\ \\ M \cdot N \cdot D_F \cdot D_F \end{array}\right\} \textit{Number of Multiplications}$$

M - Number of Channels (R, G, B)
N - Number of Different Convolution Filter Sets.
$D_K \times D_K$ - Size of one Convolution Filter (Width × Height)
$D_F \times D_F$ - Size of one Feature map in channel (Width × Height)

# MobileNet – Cascading Convolution Techniques

## Depthwise + Pointwise = Depthwise Separable Convolution

**Benefits?**

- **Far fewer multiplications** than standard method (especially when using many filters)

$$\frac{\text{Depthwise Separable}}{\text{Standard Conv}} = \frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \; + \; M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \underset{\substack{\uparrow \\ \text{\# of filters}}}{\frac{1}{N}} + \underset{\substack{\nwarrow \\ \text{Filter (kernel) Dimensions}}}{\frac{1}{D_K^2}}$$

- As an example, consider $N = 100$ and $D_K = 512$, the ratio $\approx 0.01$!

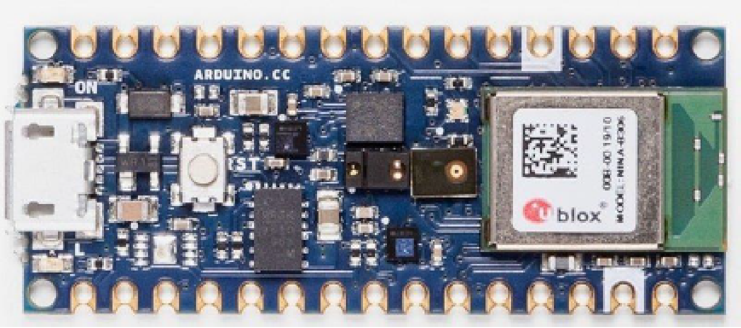
# MobileNet - Model Size Constraints

## However, the model size is still too large for our board

| Model | Size | Top-1 Accuracy |
|---|---|---|
| MobileNet v1 | 16 MB | 0.713 |

↑

Good for mobile devices with GB of RAM, but 64x microcontroller RAM



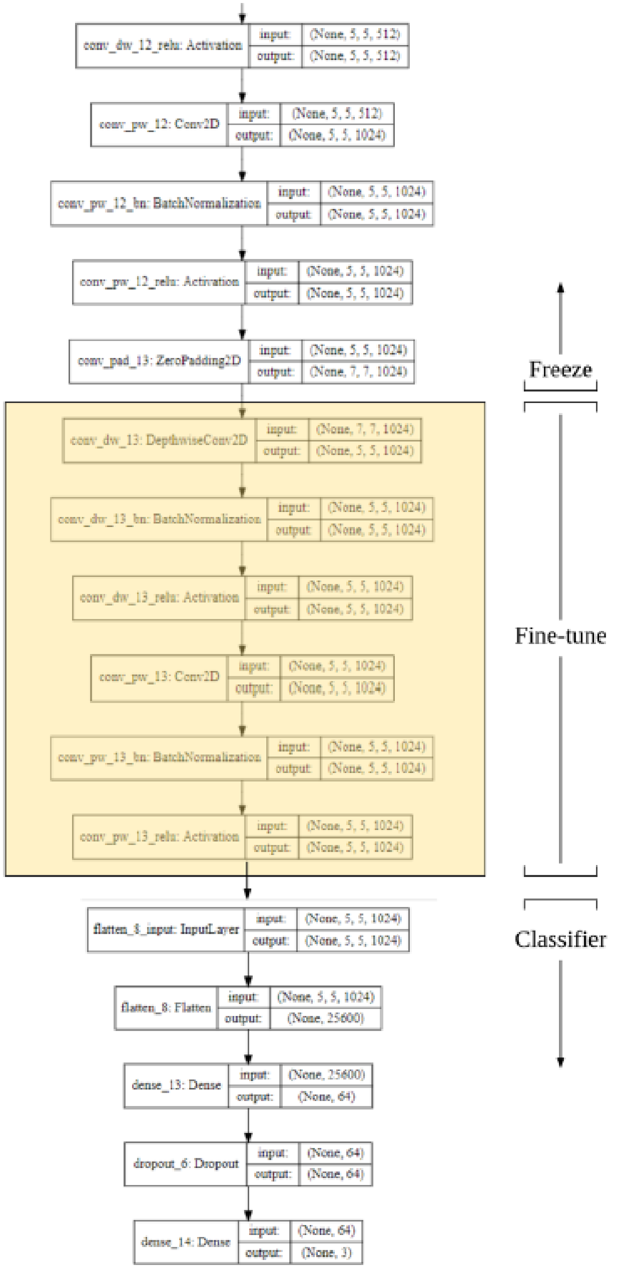
Our board only has **256 KB** RAM !

- The size of the model can be reduced further by **width multiplier**, $\boldsymbol{\alpha} \rightarrow (0,1]$
- $\alpha = 1$: baseline MobileNet; $\alpha < 1$: reduced MobileNet.

# Mobile Net and Transfer Learning

MobileNet is unique in that its convolutional block is a Depthwise-separable block.

- This speeds up the network and reduces the number of parameters it needs to store
- In the last step, combine Depthwise Conv2D and Conv2D blocks to perform the standard convolution operation.

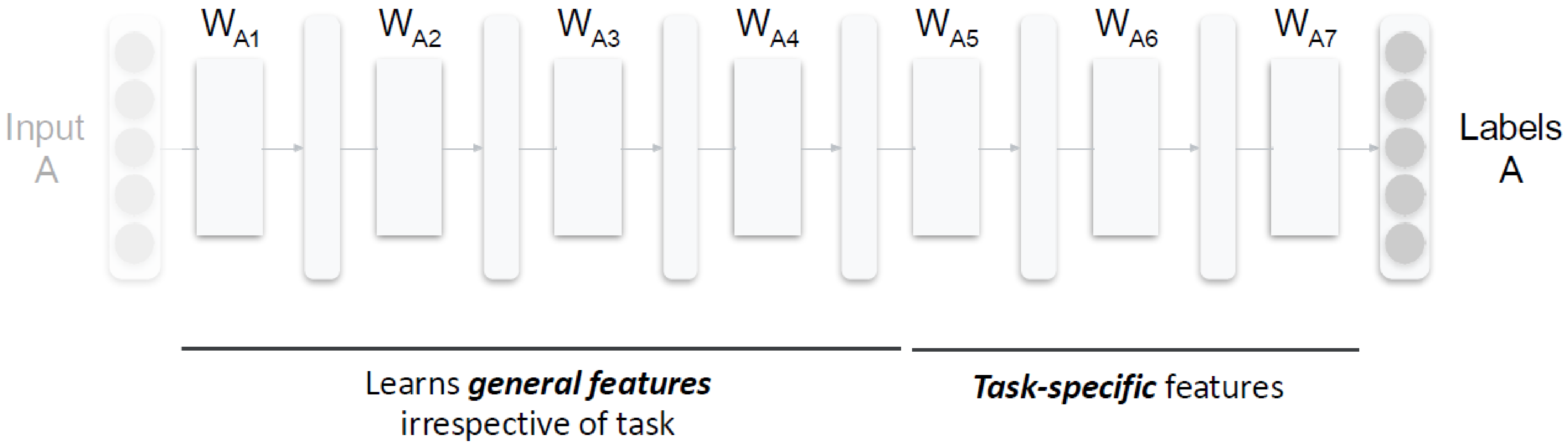

conv_dw_12_relu: Activation | input: (None, 5, 5, 512) | output: (None, 5, 5, 512)
conv_pw_12: Conv2D | input: (None, 5, 5, 512) | output: (None, 5, 5, 1024)
conv_pw_12_bn: BatchNormalization | input: (None, 5, 5, 1024) | output: (None, 5, 5, 1024)
conv_pw_12_relu: Activation | input: (None, 5, 5, 1024) | output: (None, 5, 5, 1024)
conv_pad_13: ZeroPadding2D | input: (None, 5, 5, 1024) | output: (None, 7, 7, 1024)
conv_dw_13: DepthwiseConv2D | input: (None, 7, 7, 1024) | output: (None, 5, 5, 1024)
conv_dw_13_bn: BatchNormalization | input: (None, 5, 5, 1024) | output: (None, 5, 5, 1024)
conv_dw_13_relu: Activation | input: (None, 5, 5, 1024) | output: (None, 5, 5, 1024)
conv_pw_13: Conv2D | input: (None, 5, 5, 1024) | output: (None, 5, 5, 1024)
conv_pw_13_bn: BatchNormalization | input: (None, 5, 5, 1024) | output: (None, 5, 5, 1024)
conv_pw_13_relu: Activation | input: (None, 5, 5, 1024) | output: (None, 5, 5, 1024)
flatten_8_input: InputLayer | input: (None, 5, 5, 1024) | output: (None, 5, 5, 1024)
flatten_8: Flatten | input: (None, 5, 5, 1024) | output: (None, 25600)
dense_13: Dense | input: (None, 25600) | output: (None, 64)
dropout_6: Dropout | input: (None, 64) | output: (None, 64)
dense_14: Dense | input: (None, 64) | output: (None, 3)
Freeze
Fine-tune
Classifier

# Transfer Learning

- Transfer Learning: a model trained on one task is re-purposed on another related task
- A model trained on one task will **have learned features** that are useful for other tasks
- E.g., a model trained to recognize cars could be used to recognize trucks as well

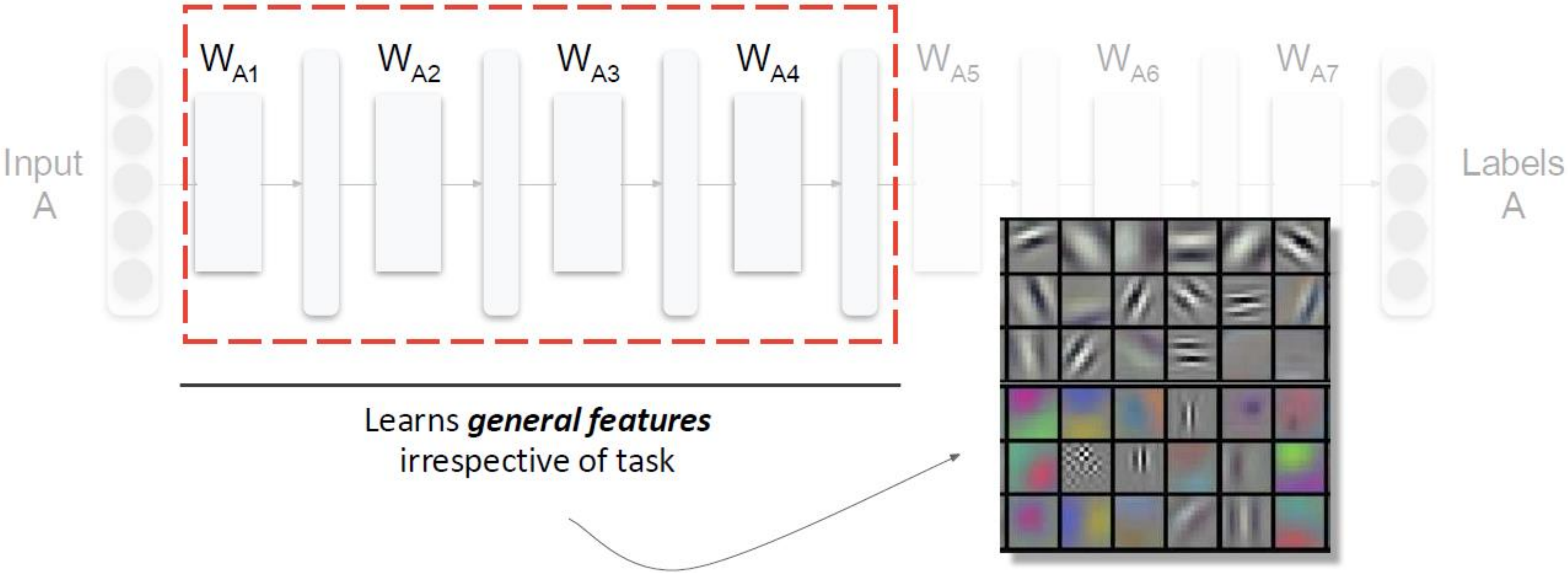
# Transfer Learning

- Do we have to train the model from scratch every single time?



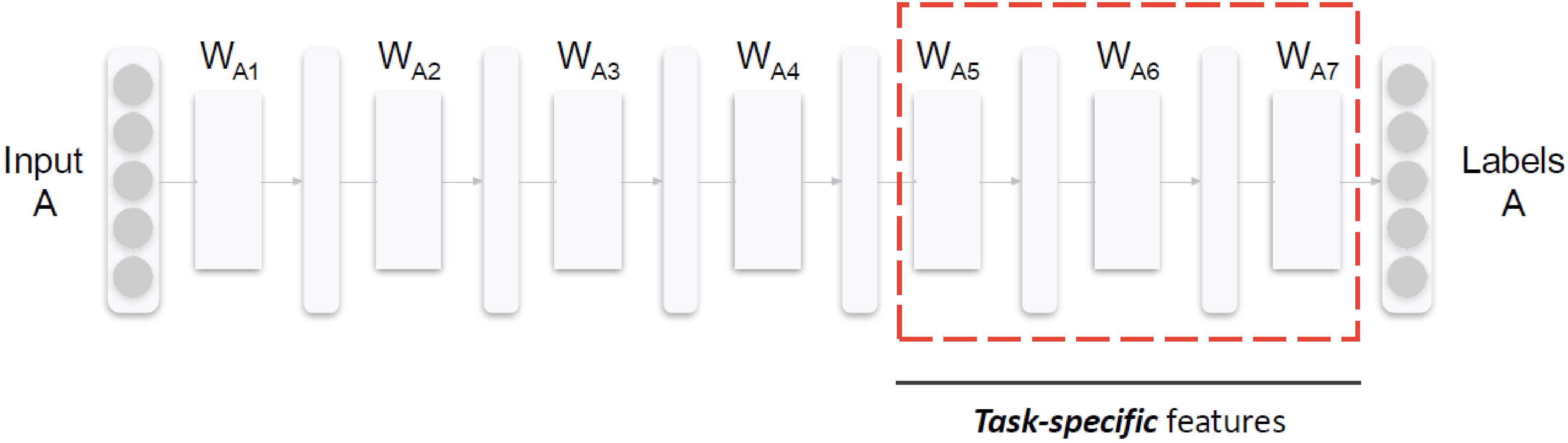
Input A
$W_{A1}$
$W_{A2}$
$W_{A3}$
$W_{A4}$
$W_{A5}$
$W_{A6}$
$W_{A7}$
Labels A
Learns ***general features*** irrespective of task
***Task-specific*** features

# Transfer Learning



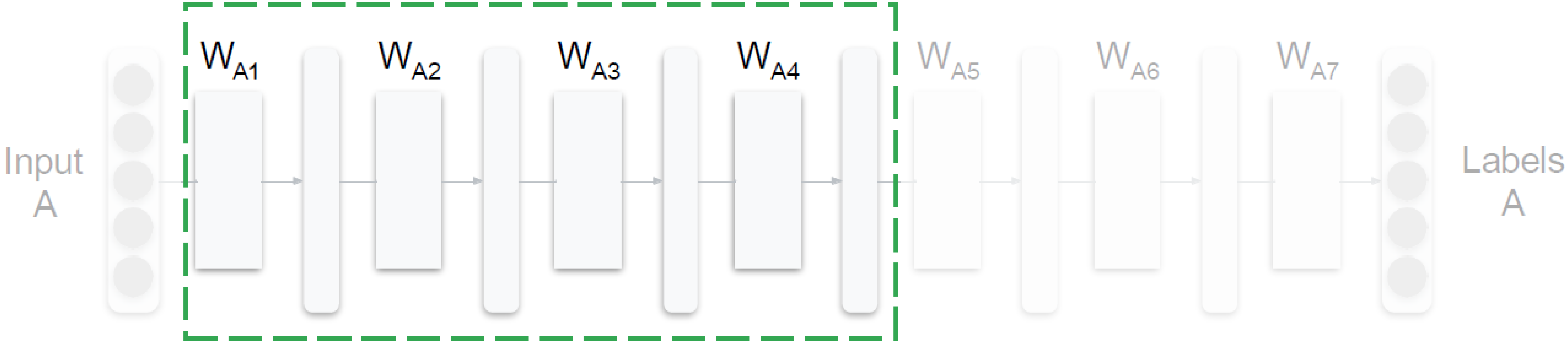
Learns ***general features*** irrespective of task

# Transfer Learning



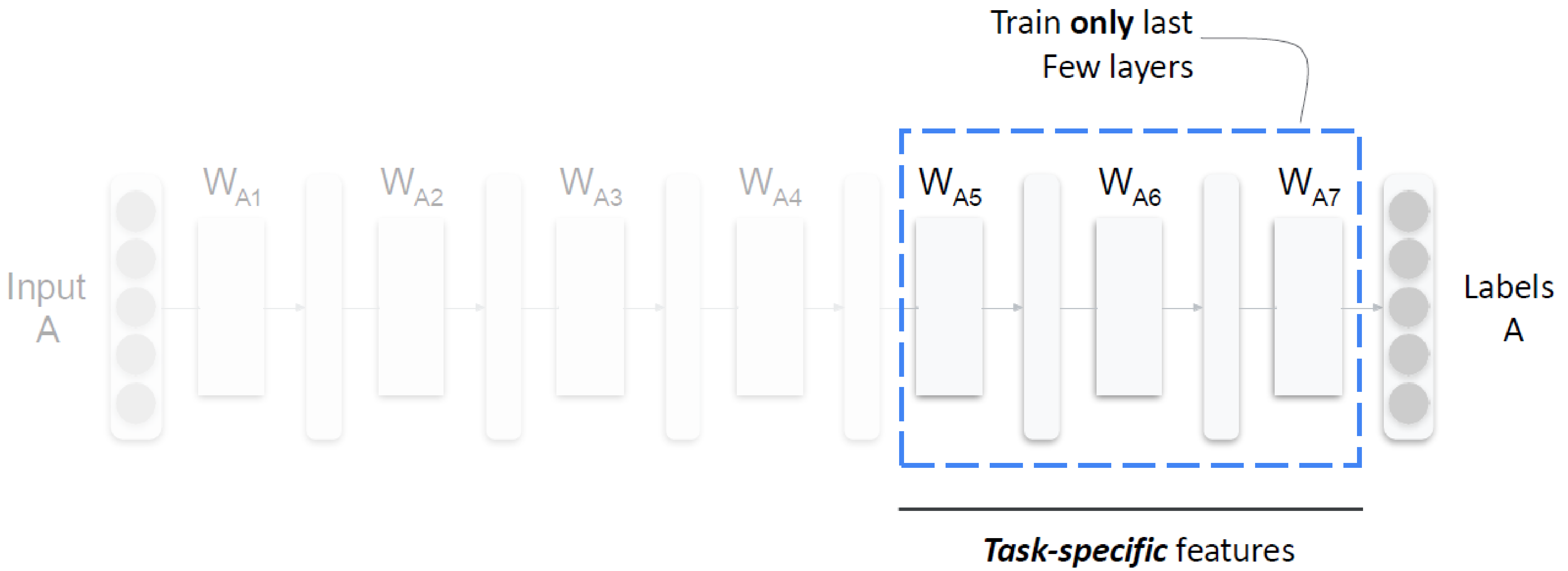
# Transfer Learning

**Reuse** (freeze general feature extraction)



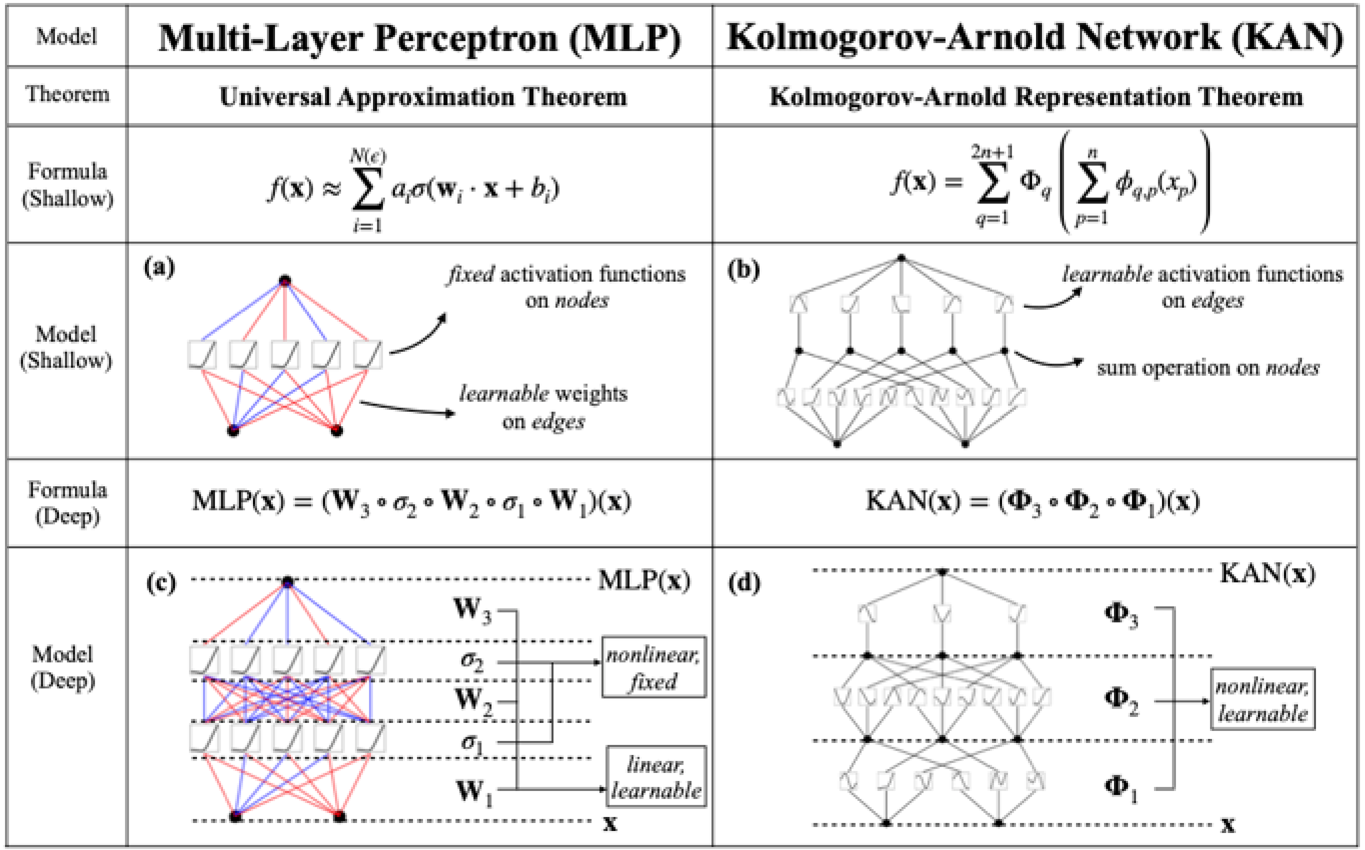
Learns ***general features*** irrespective of task

# Transfer Learning



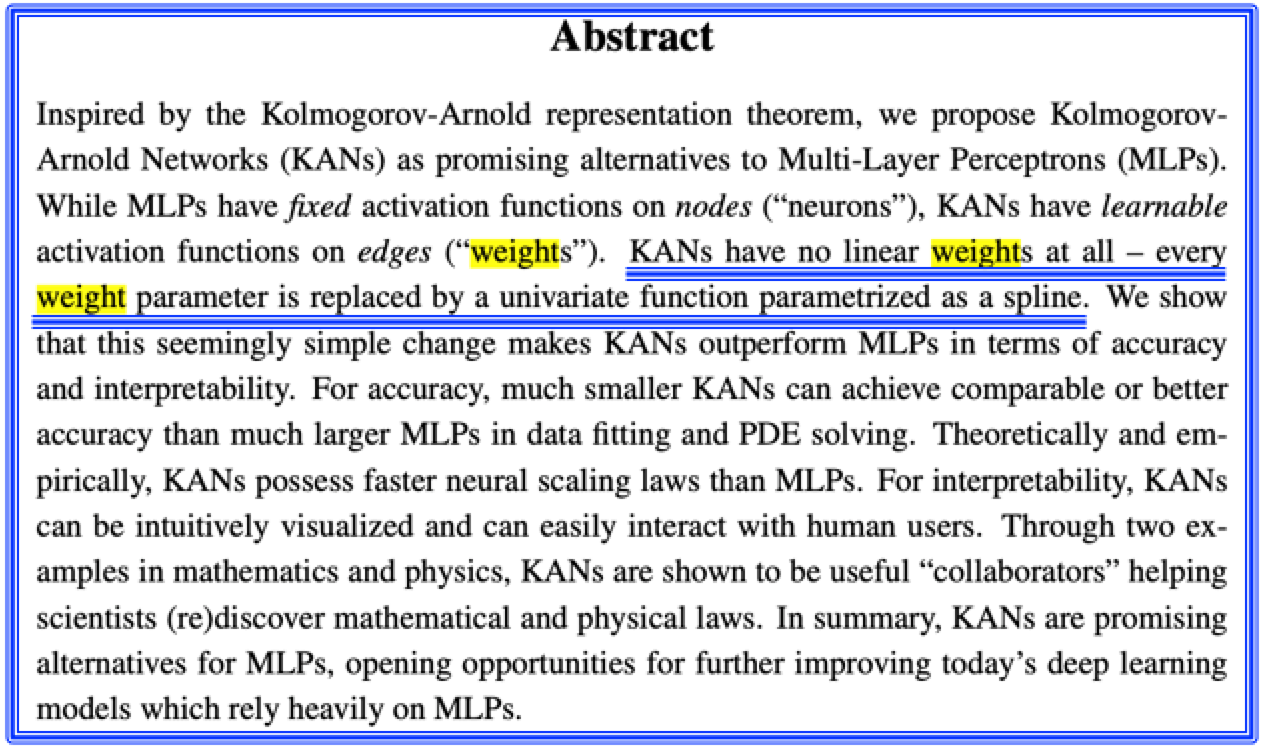
# KAN: Kolmogorov-Arnold Networks

- Liu, Ziming, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. "KAN: Kolmogorov-Arnold Networks." arXiv preprint arXiv:2404.19756 (2024).



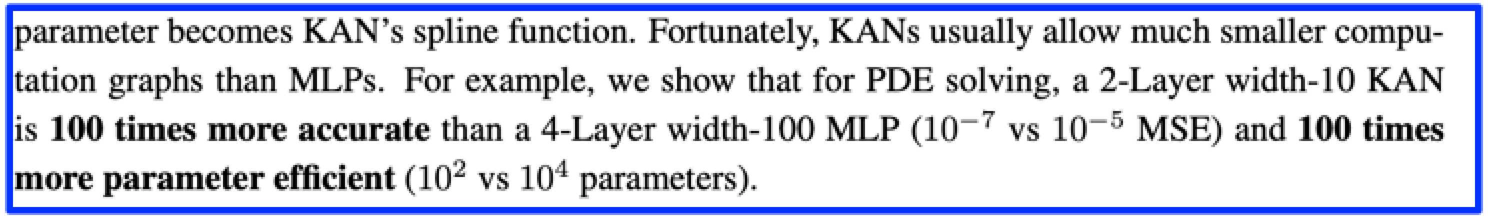
| Model | Multi-Layer Perceptron (MLP) | Kolmogorov-Arnold Network (KAN) |
|---|---|---|
| Theorem | Universal Approximation Theorem | Kolmogorov-Arnold Representation Theorem |
| Formula (Shallow) | $f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$ | $f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$ |
| Model (Shallow) | (a)<br>*fixed* activation functions on *nodes*<br>*learnable* weights on *edges* | (b)<br>*learnable* activation functions on *edges*<br>sum operation on *nodes* |
| Formula (Deep) | $\text{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$ | $\text{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$ |
| Model (Deep) | (c)<br>MLP(**x**)<br>$\mathbf{W}_3$<br>$\sigma_2$<br>$\mathbf{W}_2$<br>$\sigma_1$<br>$\mathbf{W}_1$<br>*nonlinear, fixed*<br>*linear, learnable*<br>**x** | (d)<br>KAN(**x**)<br>$\mathbf{\Phi}_3$<br>$\mathbf{\Phi}_2$<br>$\mathbf{\Phi}_1$<br>*nonlinear, learnable*<br>**x** |

# KAN: Kolmogorov-Arnold Networks



## Abstract

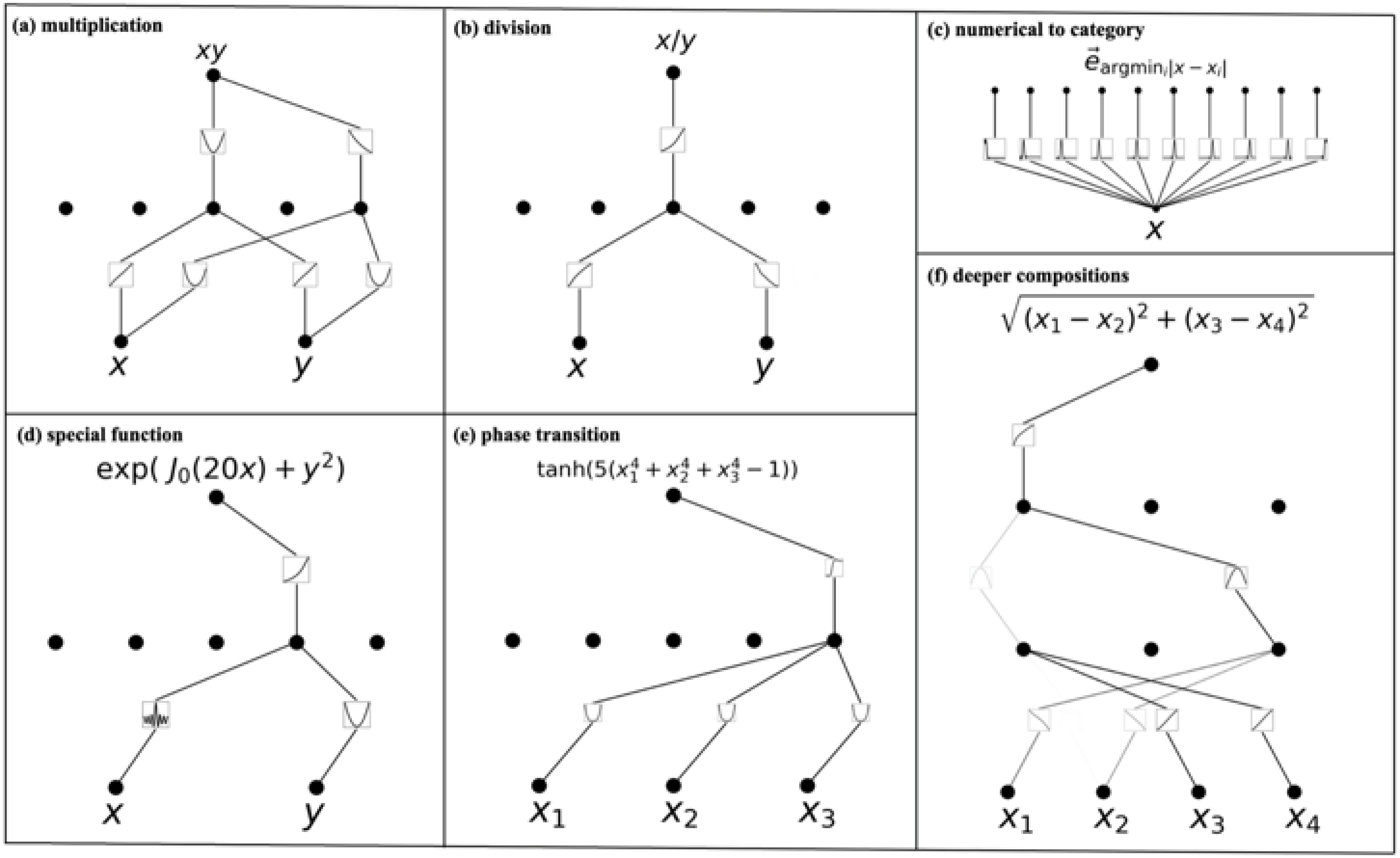
Inspired by the Kolmogorov-Arnold representation theorem, we propose Kolmogorov-Arnold Networks (KANs) as promising alternatives to Multi-Layer Perceptrons (MLPs). While MLPs have *fixed* activation functions on *nodes* (“neurons”), KANs have *learnable* activation functions on *edges* (“weights”). KANs have no linear weights at all – every weight parameter is replaced by a univariate function parametrized as a spline. We show that this seemingly simple change makes KANs outperform MLPs in terms of accuracy and interpretability. For accuracy, much smaller KANs can achieve comparable or better accuracy than much larger MLPs in data fitting and PDE solving. Theoretically and empirically, KANs possess faster neural scaling laws than MLPs. For interpretability, KANs can be intuitively visualized and can easily interact with human users. Through two examples in mathematics and physics, KANs are shown to be useful “collaborators” helping scientists (re)discover mathematical and physical laws. In summary, KANs are promising alternatives for MLPs, opening opportunities for further improving today’s deep learning models which rely heavily on MLPs.



parameter becomes KAN’s spline function. Fortunately, KANs usually allow much smaller computation graphs than MLPs. For example, we show that for PDE solving, a 2-Layer width-10 KAN is **100 times more accurate** than a 4-Layer width-100 MLP ($10^{-7}$ vs $10^{-5}$ MSE) and **100 times more parameter efficient** ($10^2$ vs $10^4$ parameters).

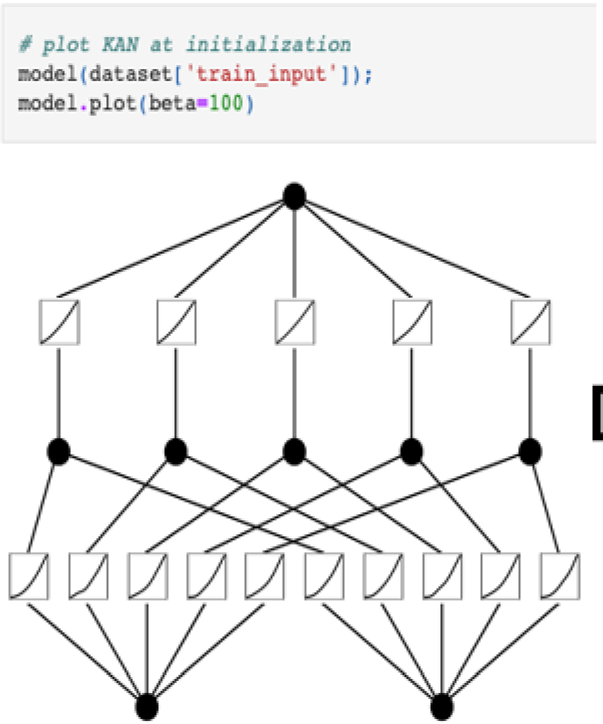
# Interpretability of KAN and Pruning

- https://github.com/KindXiaoming/pykan

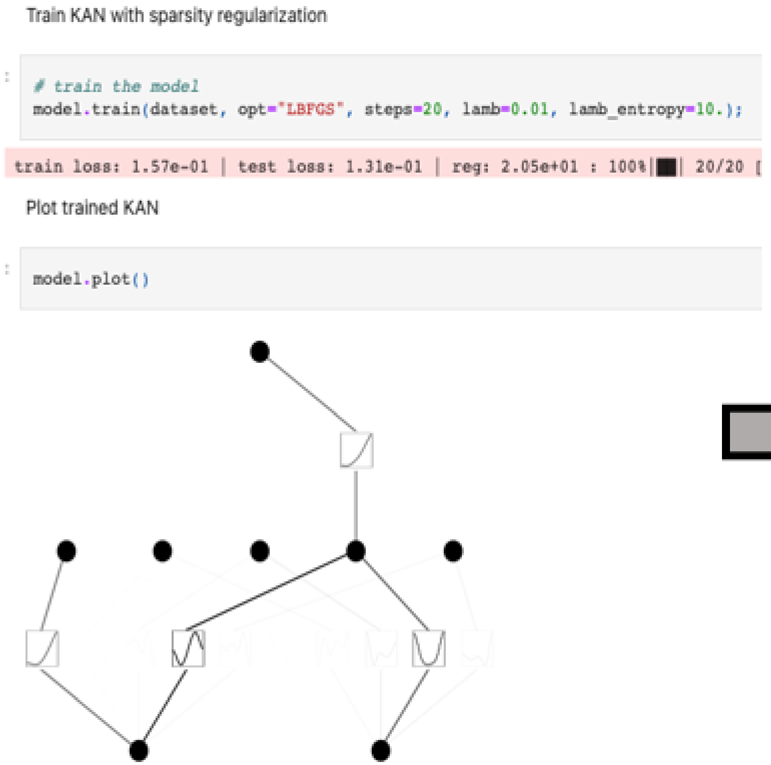


# Pruning in KAN

- https://github.com/KindXiaoming/pykan



```
# plot KAN at initialization
model(dataset['train_input']);
model.plot(beta=100)
```



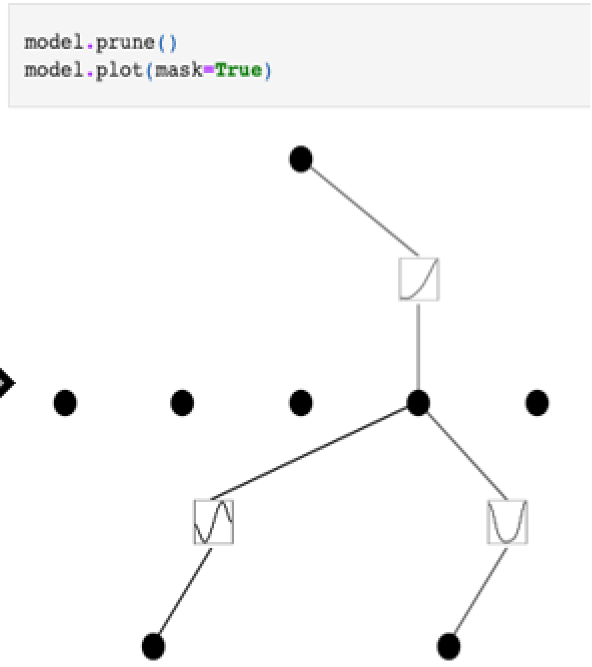
Train KAN with sparsity regularization

```
# train the model
model.train(dataset, opt="LBFGS", steps=20, lamb=0.01, lamb_entropy=10.);
```

```
train loss: 1.57e-01 | test loss: 1.31e-01 | reg: 2.05e+01 : 100%|██| 20/20 [
```

Plot trained KAN

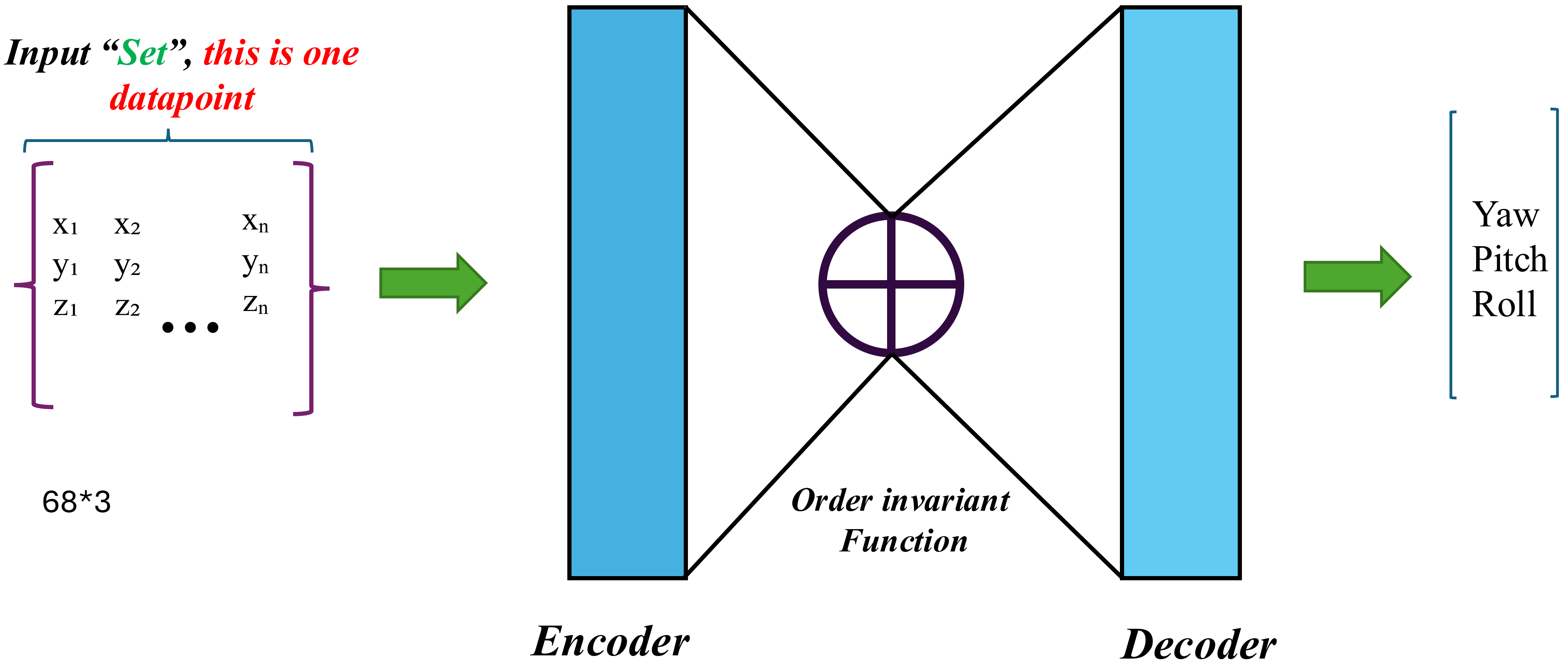
```
model.plot()
```



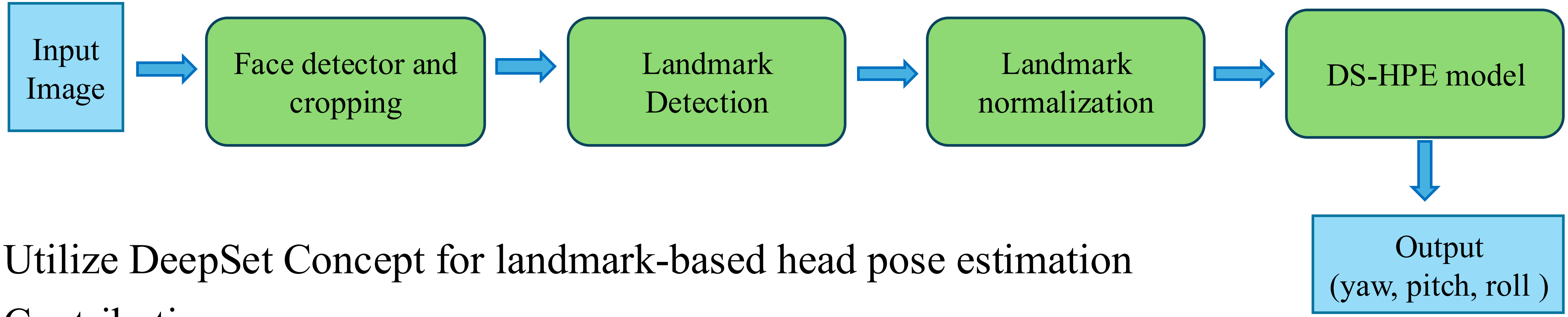
```
model.prune()
model.plot(mask=True)
```

Prune KAN and replot (get a smaller shape)

# Deep-Set Architecture



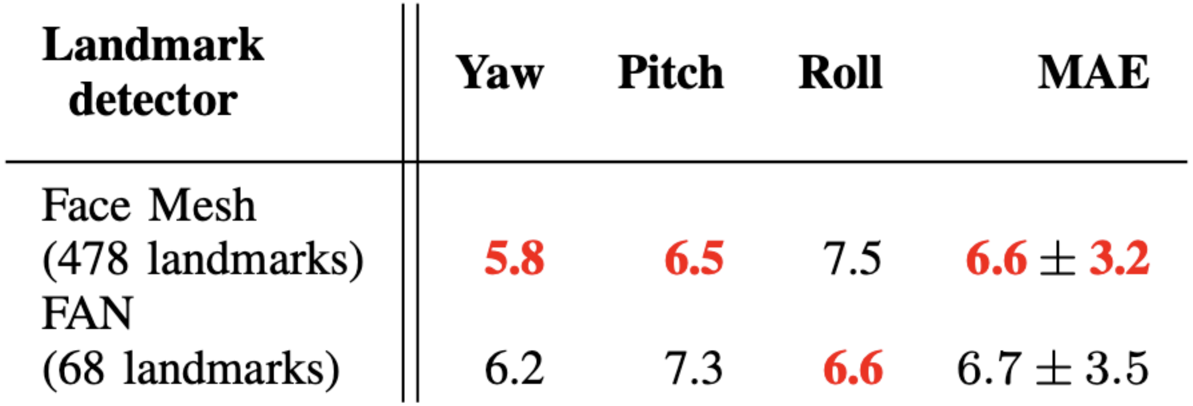
# DS-HPE: Deep Set for Head Pose Estimation



- Utilize DeepSet Concept for landmark-based head pose estimation
- Contributions
  - Comparison between face detectors Dlib and Blazeface and SSD
  - Impact of landmark detectors Face Mesh (478 3D landmarks) and FAN (68 3D landmarks).
  - Compare this model accuracy against SOTA with respect to 4 datasets - Accuracy, latency

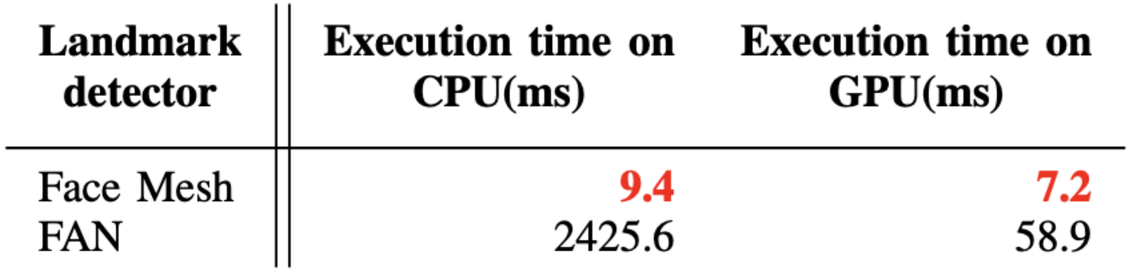
V. Menan, A. Gawesha, P. Samarasinghe and D. Kasthurirathna, "DS-HPE: Deep Set for Head Pose Estimation," *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2023, pp. 1179-1184, doi: 10.1109/CCWC57344.2023.10099159.

# Landmark detectors performance comparison



| Landmark detector | Yaw | Pitch | Roll | MAE |
|---|---|---|---|---|
| Face Mesh (478 landmarks) | **5.8** | **6.5** | 7.5 | **6.6 ± 3.2** |
| FAN (68 landmarks) | 6.2 | 7.3 | **6.6** | 6.7 ± 3.5 |

Table 1: MAE variation of DS-HPE with landmark extractors on the EYEDIAP dataset

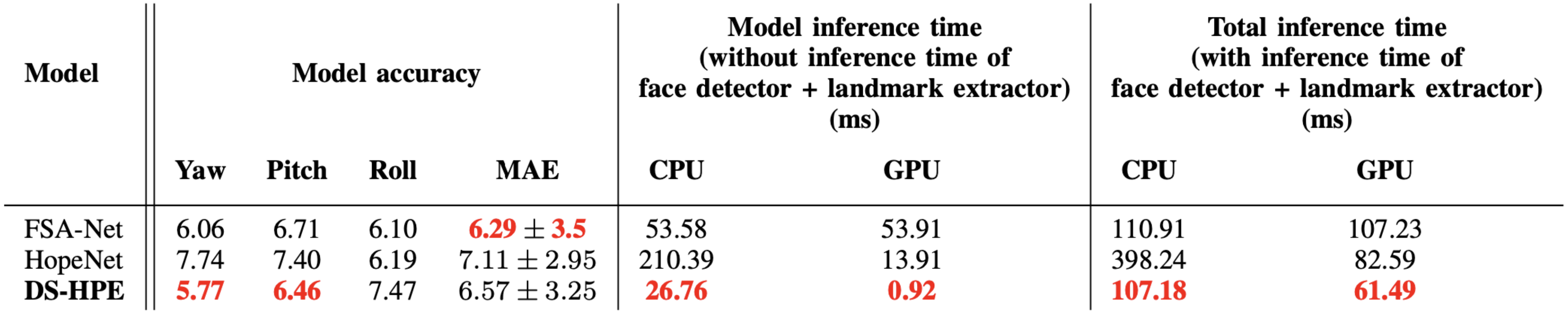


| Landmark detector | Execution time on CPU(ms) | Execution time on GPU(ms) |
|---|---|---|
| Face Mesh | **9.4** | **7.2** |
| FAN | 2425.6 | 58.9 |

Table 2: landmark extractor execution time On CPU vs GPU

- FaceMesh gives 478 landmarks while FAN gives 68 landmarks.
- When both are utlized, DS-HPE can perform in almost same accuracy.
- Face Mesh landmark extractor is 257x faster than FAN on the CPU and 8x faster than FAN on the GPU.
- For resource constrained environments Face Mesh is more suitable

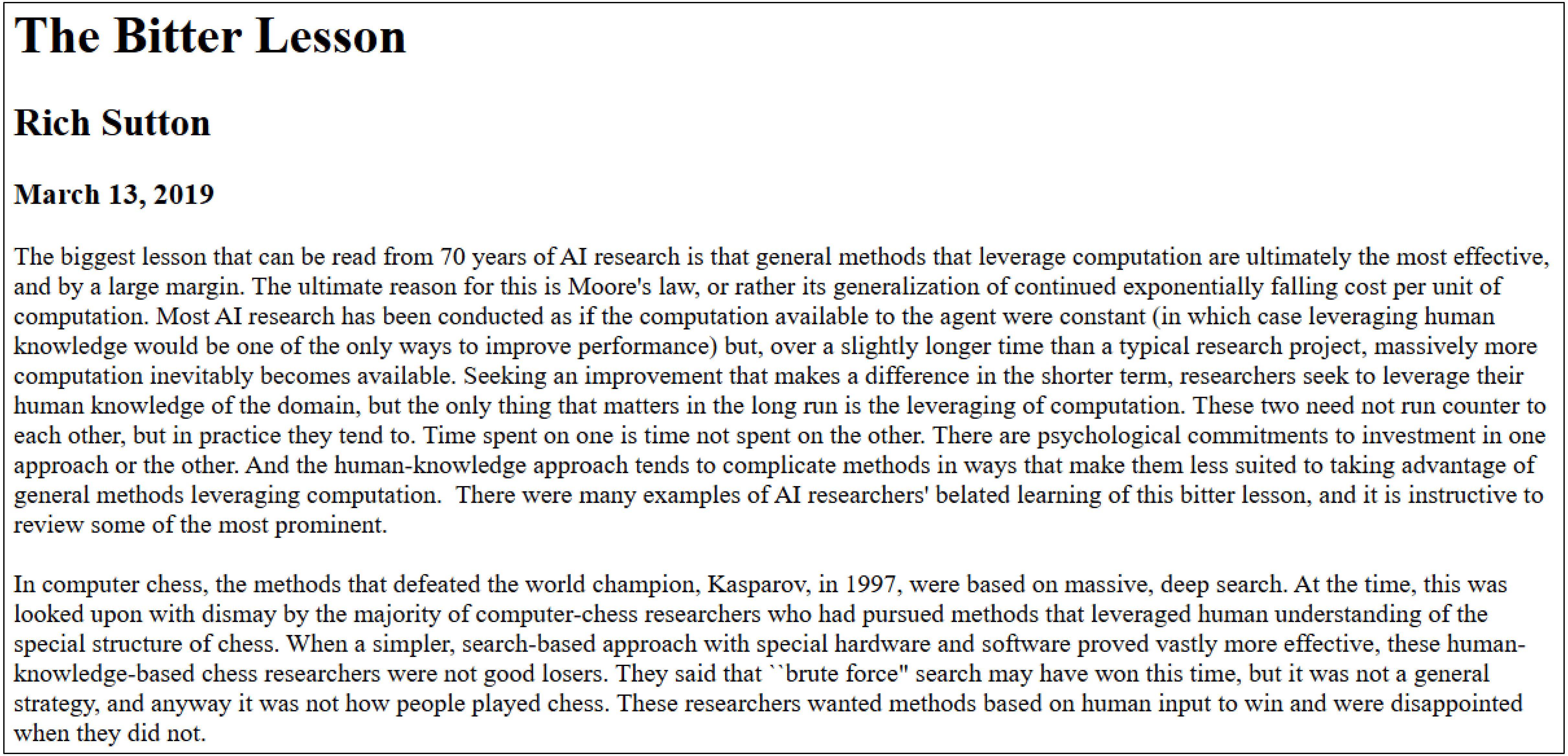
# DS-HPE vs SOA performance



| Model | Model accuracy | | | | Model inference time<br>(without inference time of<br>face detector + landmark extractor)<br>(ms) | | Total inference time<br>(with inference time of<br>face detector + landmark extractor)<br>(ms) | |
|---|---|---|---|---|---|---|---|---|
| | Yaw | Pitch | Roll | MAE | CPU | GPU | CPU | GPU |
| FSA-Net | 6.06 | 6.71 | 6.10 | **6.29 ± 3.5** | 53.58 | 53.91 | 110.91 | 107.23 |
| HopeNet | 7.74 | 7.40 | 6.19 | 7.11 ± 2.95 | 210.39 | 13.91 | 398.24 | 82.59 |
| **DS-HPE** | **5.77** | **6.46** | 7.47 | 6.57 ± 3.25 | **26.76** | **0.92** | **107.18** | **61.49** |

Table 1: DS-HPE vs state-of-the-art models performance on EYEDIAP dataset

- DS-HPE perfroms best on yaw and pitch and performs alsmot same as FSA-Net on overall MAE.
- RGB based ,methods have been the SOA methods but being a landmakr based method our approach
- DS-HPE is 2x faster than the FSA-Net and 7x faster than HopeNet on CPU.
- And 58x faster than FSA-Net and and 15x faster than HopeNet on GPU (without inference time of face detector + landmark extractor)
- DS-HPE pipeline performs the same as the FSA-Net and is 3x faster than HopeNet on the CPU and 1.7x faster than the FSA-Net and 1.5x faster than HopeNet on the GPU. (with face detector and landmark extractor)
- Landmark detector  latency is a huge bottleneck in our proposed pipeline.



# The Bitter Lesson

## Rich Sutton

### March 13, 2019

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation. Most AI research has been conducted as if the computation available to the agent were constant (in which case leveraging human knowledge would be one of the only ways to improve performance) but, over a slightly longer time than a typical research project, massively more computation inevitably becomes available. Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation. These two need not run counter to each other, but in practice they tend to. Time spent on one is time not spent on the other. There are psychological commitments to investment in one approach or the other. And the human-knowledge approach tends to complicate methods in ways that make them less suited to taking advantage of general methods leveraging computation.  There were many examples of AI researchers' belated learning of this bitter lesson, and it is instructive to review some of the most prominent.

In computer chess, the methods that defeated the world champion, Kasparov, in 1997, were based on massive, deep search. At the time, this was looked upon with dismay by the majority of computer-chess researchers who had pursued methods that leveraged human understanding of the special structure of chess. When a simpler, search-based approach with special hardware and software proved vastly more effective, these human-knowledge-based chess researchers were not good losers. They said that ``brute force" search may have won this time, but it was not a general strategy, and anyway it was not how people played chess. These researchers wanted methods based on human input to win and were disappointed when they did not.