Vision II : Deep Learning Methods

Zirui Zang and the F1TENTH Team
Contact: rahulm@seas.upenn.edu

F1 TENTH

Penn Engineering

Vision Module Overview

Lecture I : Classical Methods

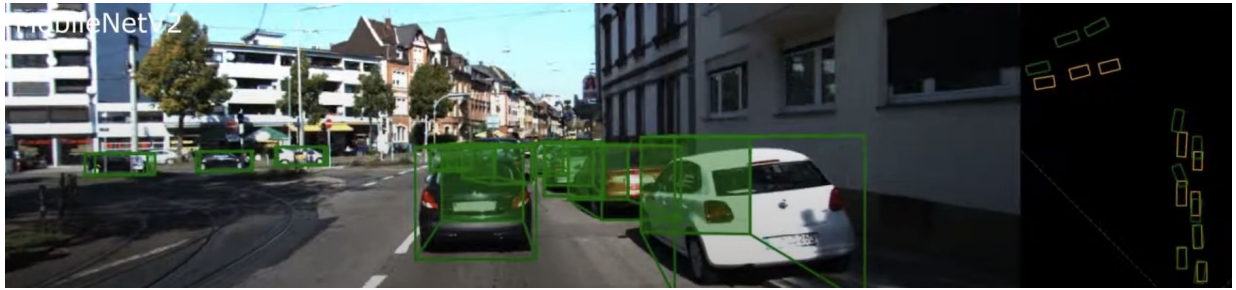
- Vision Hardware
- Accessing Camera on Linux
- Camera Model & Pose Estimation
- Useful OpenCV Functions
- Visual SLAM

Lecture II : DL Methods

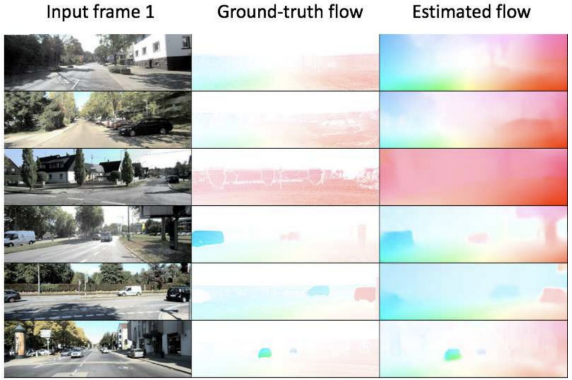
- Deep Learning Basics
- Object Detection w/ Image - YOLO (2015)
- Object Detection w/ Pointcloud - Pointpillars (2018)
- Transformer for Vision (2020)
- Network Deployment

Deep Learning Basics

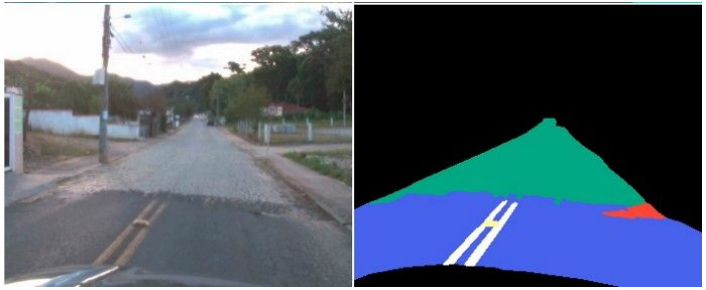
Deep Learning in Autonomous Driving



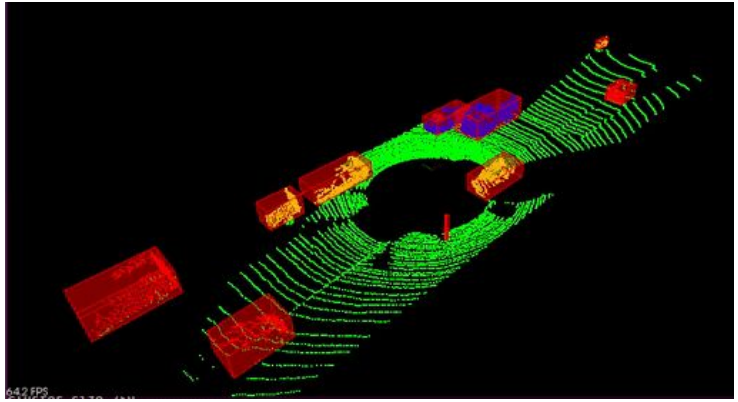
3D Object Detection from Monocular Camera



Optical Flow (Yang Wang et al.)



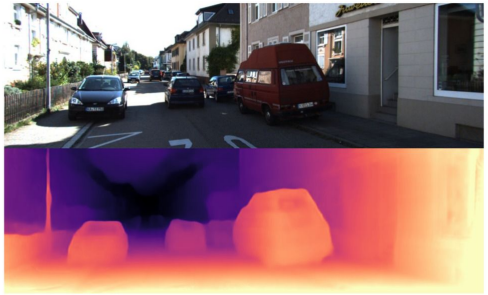
Semantic Segmentation /
Drivable Surface



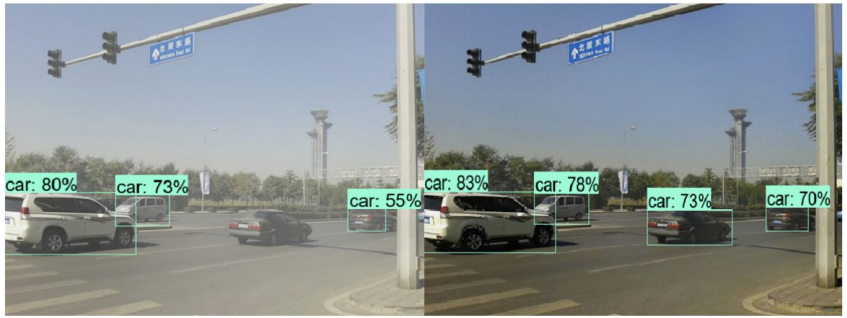
Lidar point cloud detection

Penn Engineering

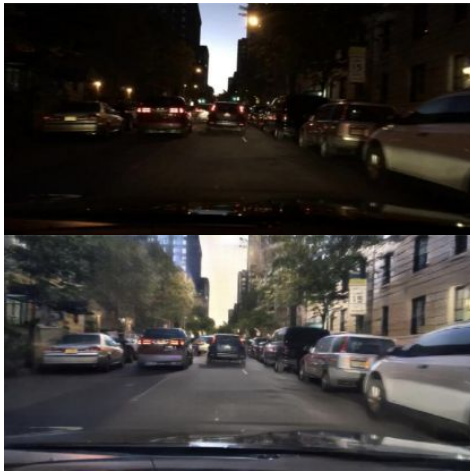
Deep Learning in Autonomous Driving



Depth from Monocular Camera
Monodepth2



Dehaze (Cameron Hodges et al.)



Night to Day (ForkGAN)

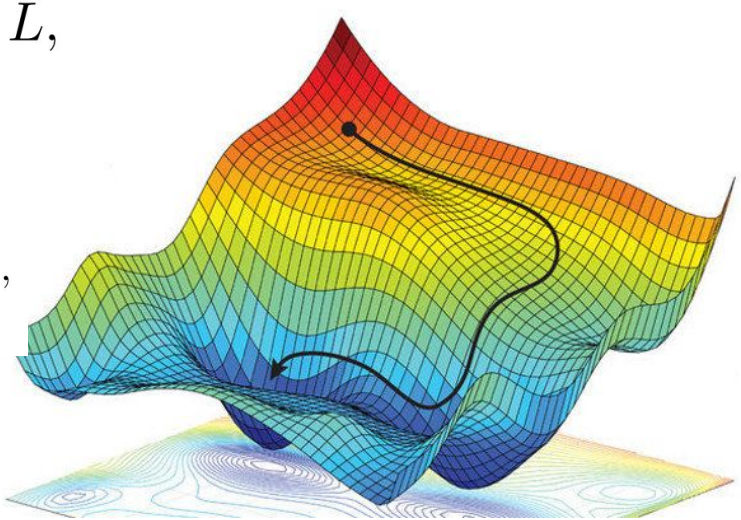
Neural Network Training Pipeline

Model: $f_k = \sigma_k(W_k f_{k-1} + b_k), \quad k = 1, \ldots, L,$

Parameters: $W_k \in \mathbb{R}^{n_k \times n_{k-1}} \qquad b_k \in \mathbb{R}^{n_k}$

Loss Function: $\mathcal{L}(W_1, b_1, \ldots, W_L, b_L) = \sum_{i=1}^{m} \ell(f_L(x_i), y_i),$

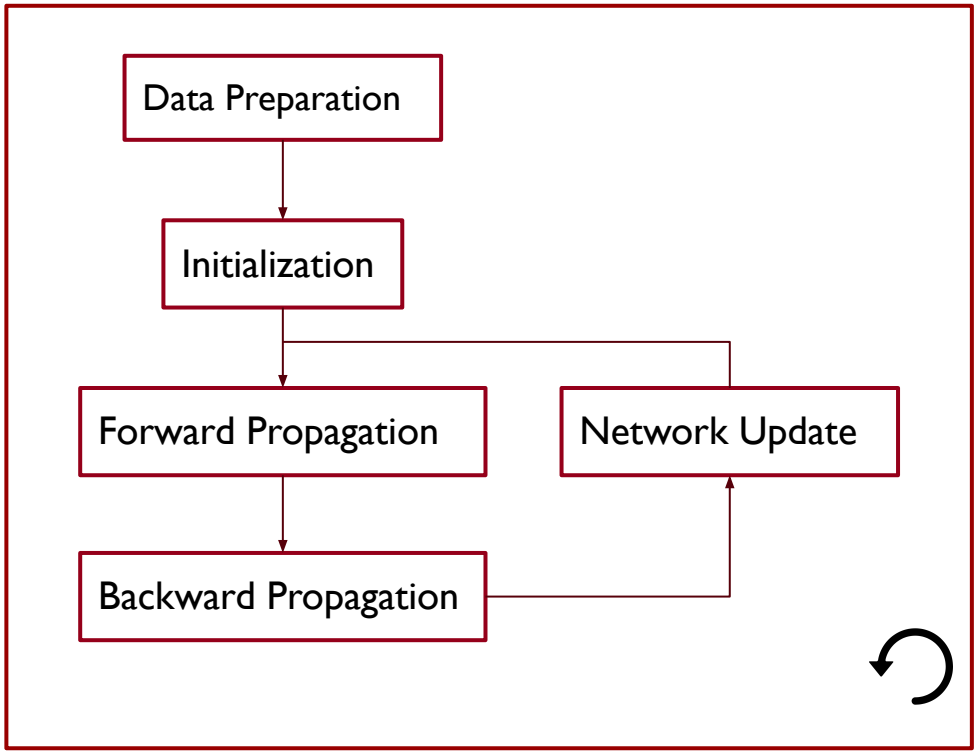
Gradient Descent: $\frac{\partial \mathcal{L}}{\partial W_k} = \frac{\partial \mathcal{L}}{\partial z_k} f_{k-1}^T, \quad \frac{\partial \mathcal{L}}{\partial b_k} = \frac{\partial \mathcal{L}}{\partial z_k}.$



Gradient Descent

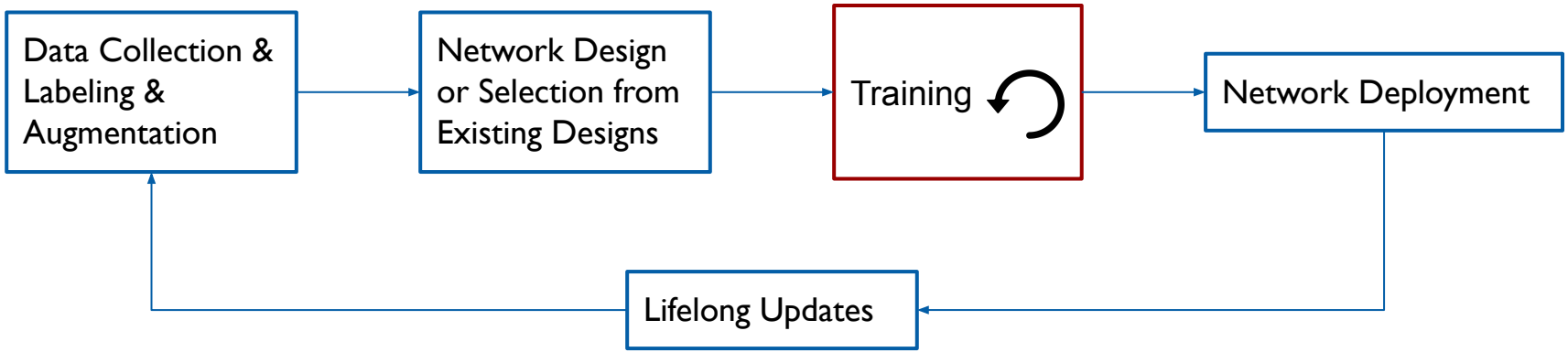
Neural Network Training Pipeline

- Training step often repeated with:
- Network design experiments
  - Add or remove a layer.
  - Change number of channels in a certain layer.
  - Add dropout, pooling layer, skip connections.
- Hyper-parameter tuning
  - learning rate
  - convolution kernel sizes, strides, etc.



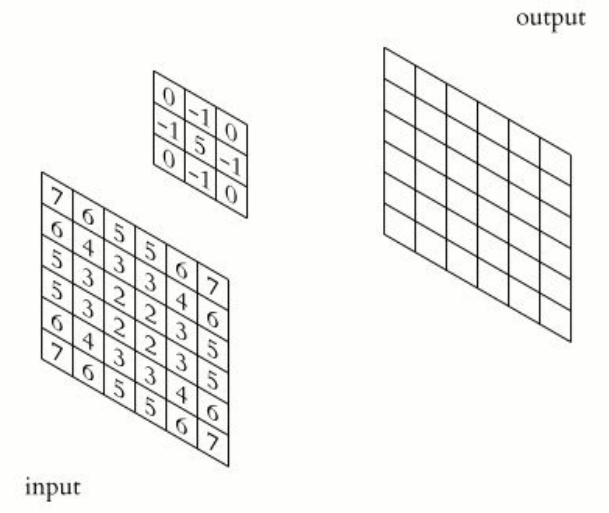
Neural Network Training Pipeline

- Data Collection and Labeling
- Network Design
- Network Training
- Network Deployment
- Lifelong Updates



Object Detection w/ Image

2D Convolution



2D convolution:

$$h = f \otimes g$$

f - the values in a 2D grid that we want to convolve

g - convolutional weights of size MxN

- The weights g is often called a kernel or a filter.
- Kernels with different weights will respond differently to features.

$$g_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad g_2 = \begin{bmatrix} 0.107 & 0.113 & 0.107 \\ 0.113 & 0.119 & 0.113 \\ 0.107 & 0.113 & 0.107 \end{bmatrix} \quad g_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$f =$

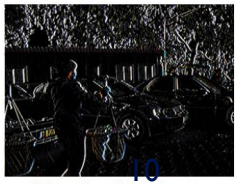
$f \otimes g_1$

Unchanged Image

$f \otimes g_2$

Blurred Image

$f \otimes g_3$

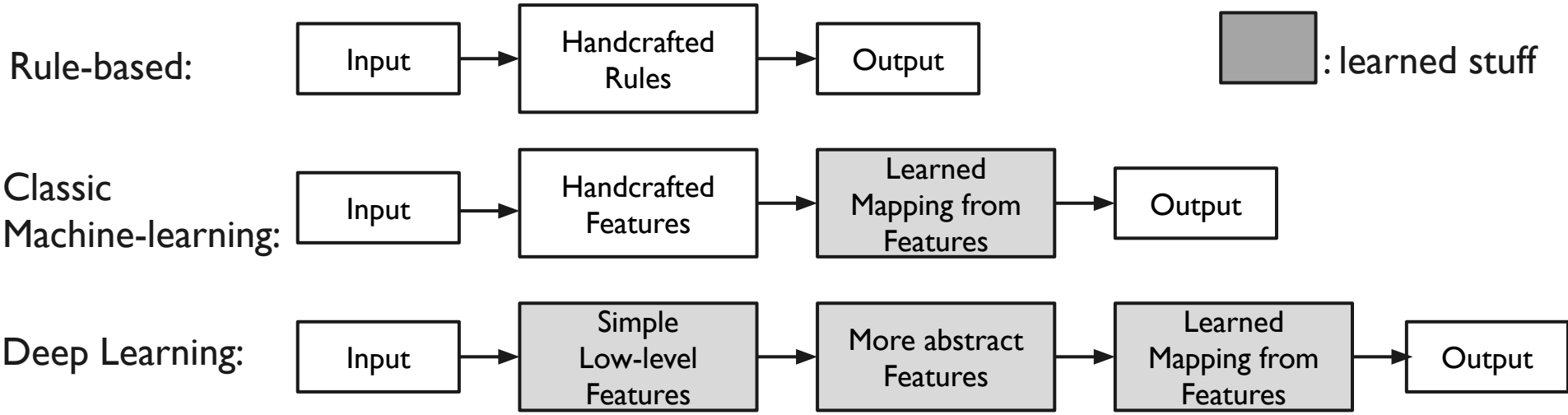


Vertical Edges

Penn Engineering

From Classic to DL

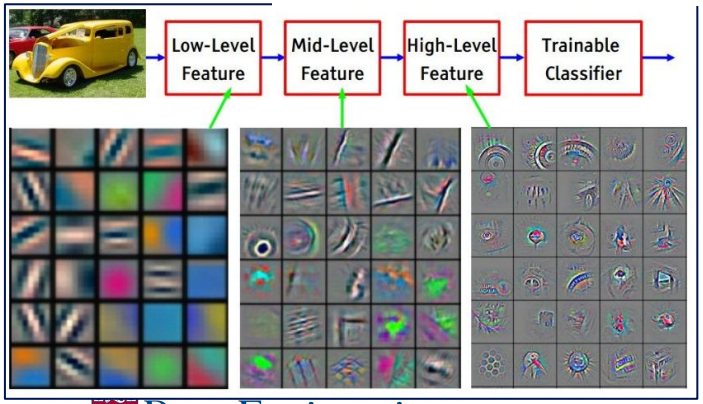
- It all comes down to feature extraction from the data.
- We can use different kernels to detect different features.
- Instead of corners and edges, can we work with higher-level features?
- Do we have to design the kernels with hands?

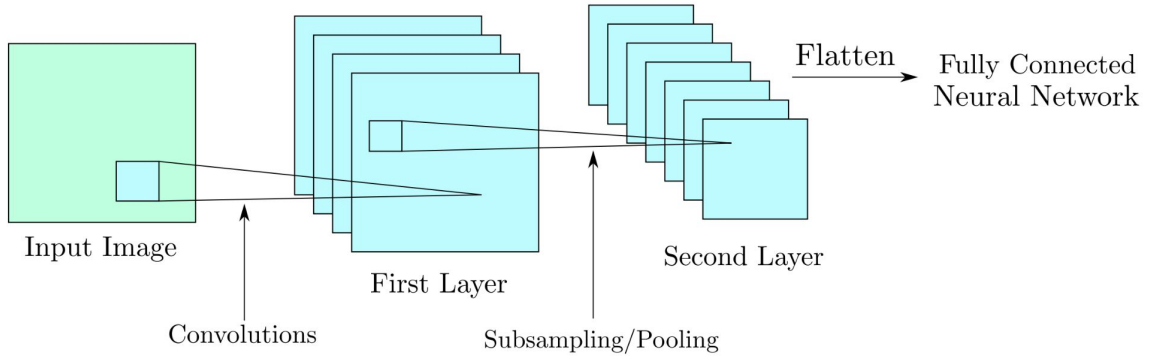


Feature Extraction - CNN

- A hierarchy of convolution kernel to detect feature at different levels.
- Nonlinear activation functions at each layer to catch nonlinear information.
- A classifier to map features to outputs.

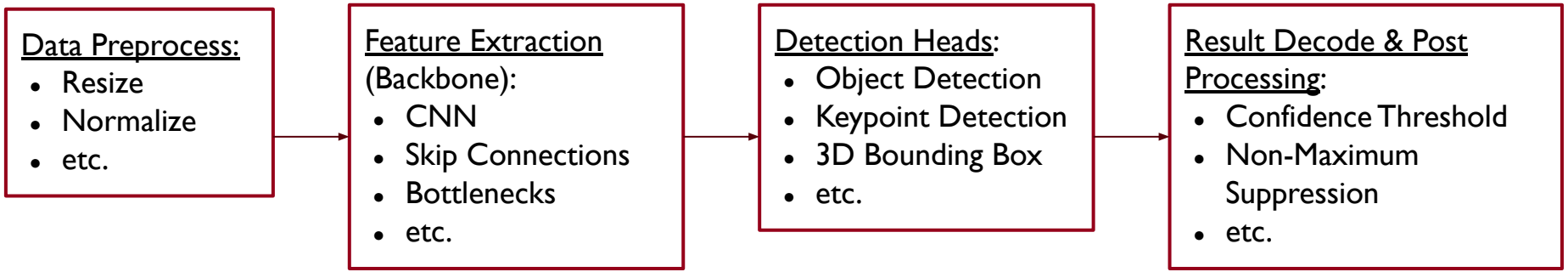
$$(W * I)(i,j) = \sum_{p,q=-N}^{N} W(N+1+p, N+1+q) I(i+p, j+q)$$





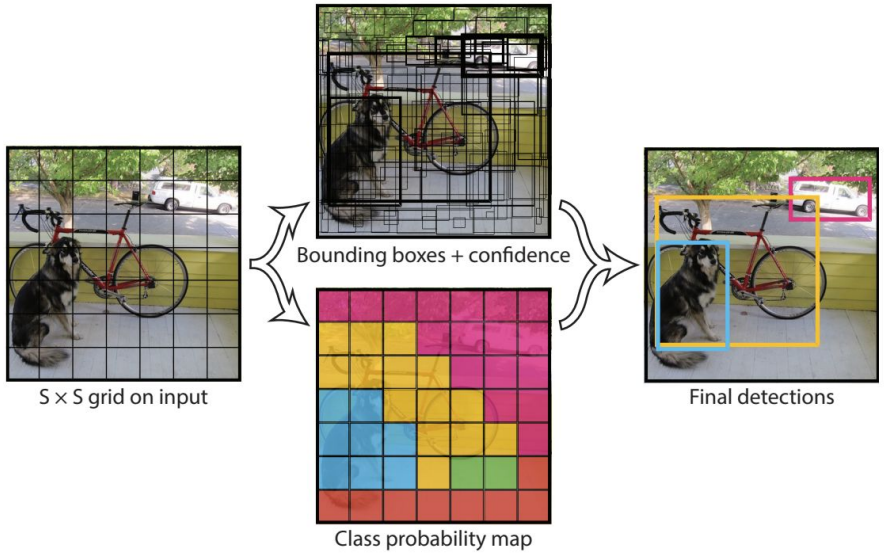
Common Detection Structure

- The structure of the neural network determines it function.
- For object detection and many other tasks, Backbone + Detection Heads is a common structure.

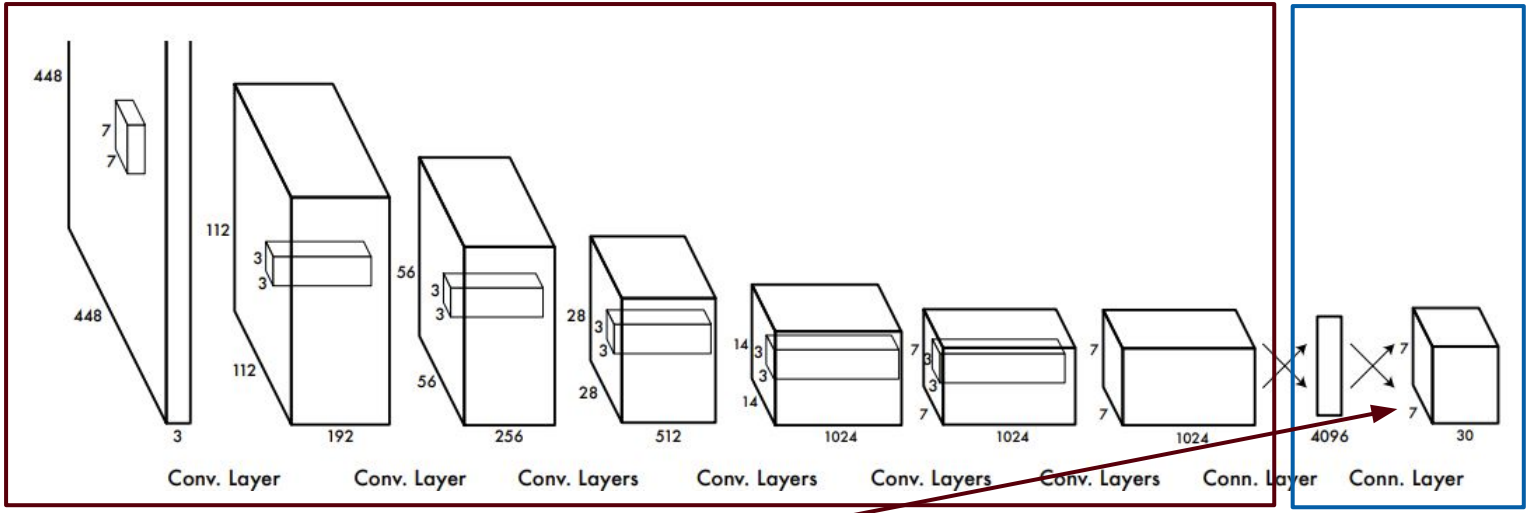


YOLO

- YOLO is a single-stage object detection architecture published in 2015.
- YOLO is fast, fairly accurate, and structurally very simple.
- But it has issues like too many false positives and require heavy post processing.
- It is old but it is a good way to learn deep learning computer vision.



YOLO Structure



7x7x30

Feature Extraction "Backbone"

"Detection Head"

S × S grid on input

7x7 windows, each window has 30 channels.
7x7 is just the output dimension from the convolutions.
Each window proposes 2 objects, each object has (w, h, x, y, confidence).
Then each window has 20 values for object classes.
So 2x5+20 = 30 channels.

Loss Function

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$ ← (x, y) coordinates error

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$ ← (w, h) coordinates error

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$ ← confidence error

confidence error, if there doesn't exist an object in this grid →

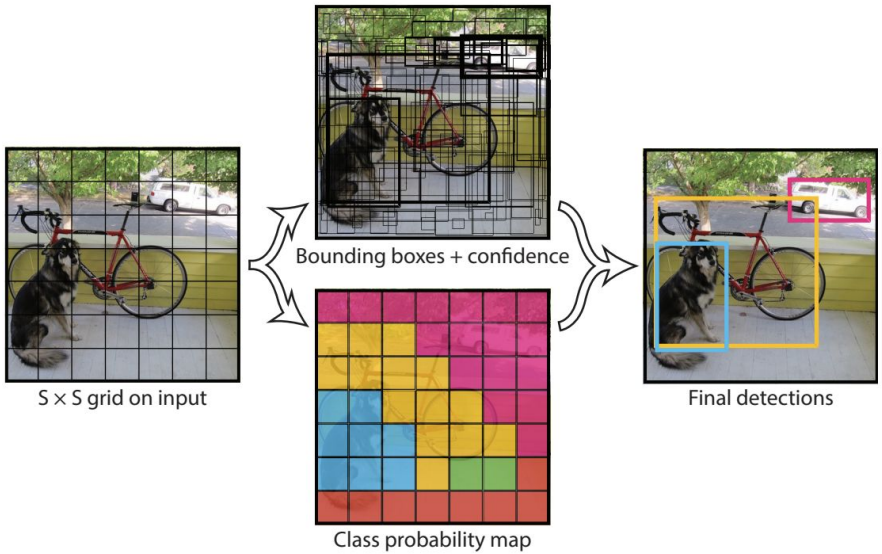
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$ ← Class error, if there exist an object in this class in this grid

YOLO loss function

How YOLO detects.

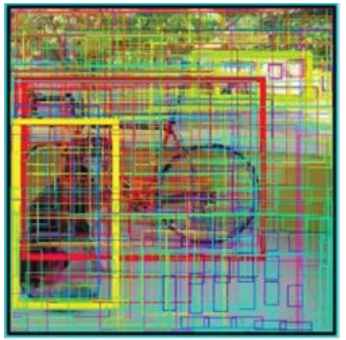
- Total is 7x7x30 tensor output.
  - 7x7 grids from convolutions
  - 2 of the [x, y, w, h, confidence] per grid.
  - 20 classes
- Detection is simple:
  - In each grid, if the confidence > threshold,
  - pick that [x, y, w, h, confidence] and the highest class.

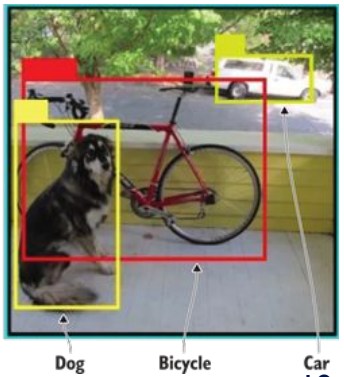


What are the limitations?

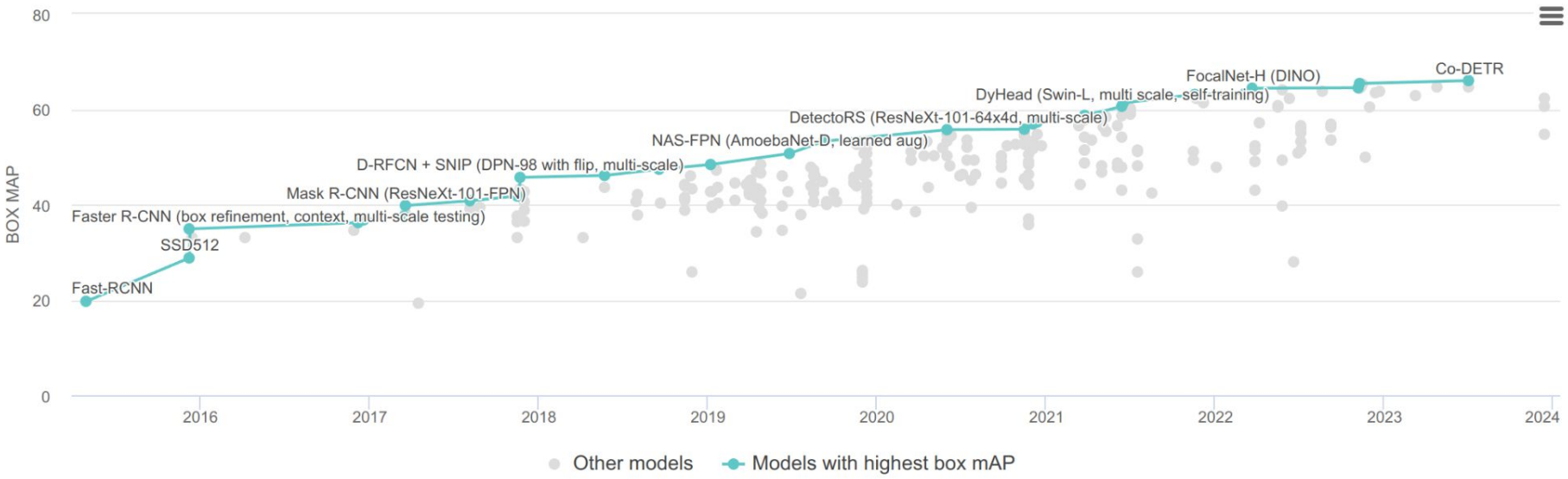
Non-maximum Suppression

- YOLO heavily rely on NMS





Development of Object Detection

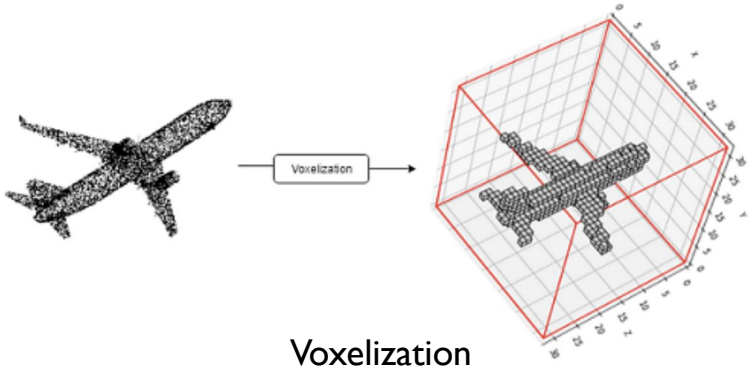


https://paperswithcode.com/sota/object-detection-on-coco

Object Detection w/ Pointcloud

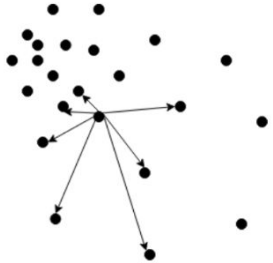
Point Cloud Object Detection

- Lidar point cloud is very sparse.
- Can we just do binning them into 3D voxels and perform 3D convolutions like in YOLO?

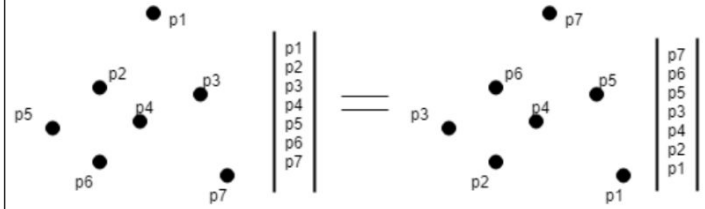


Voxelization

(a) Irregular. Sparse and dense regions



(b) Unstructured. No grid, each point is independent and distance between neighboring points is not fixed

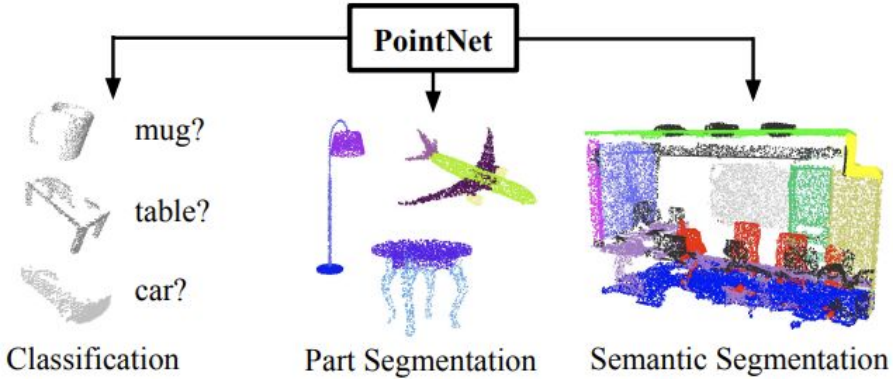


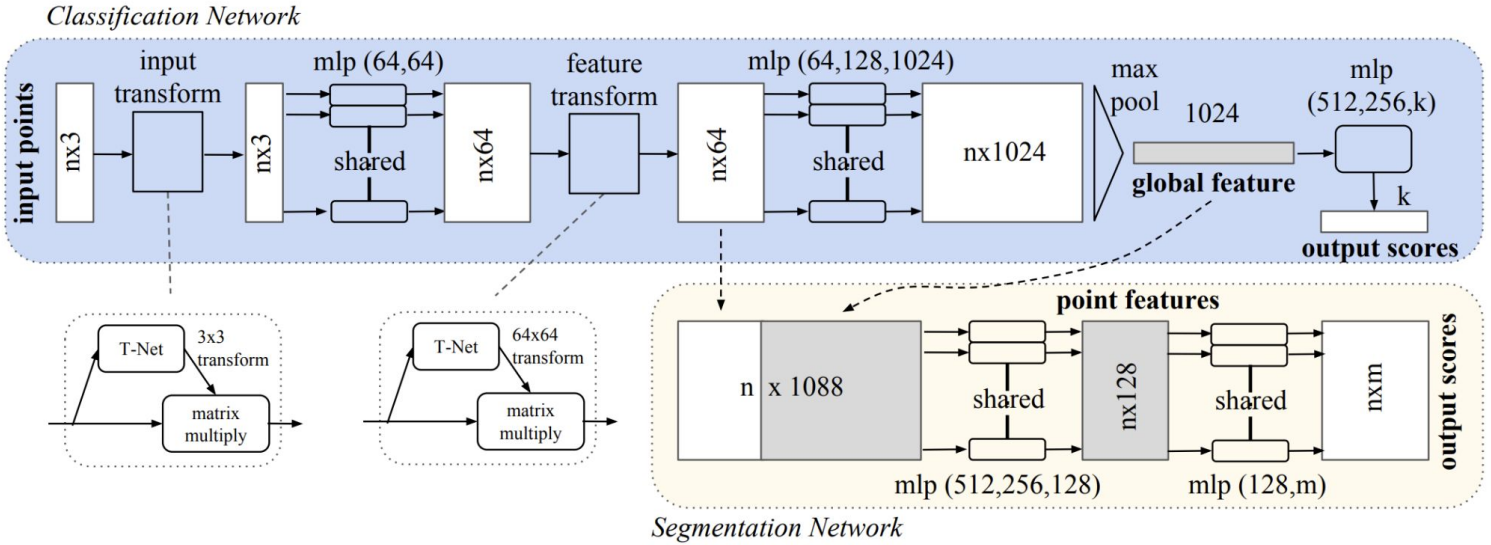
(c) Unordered. As a set, point cloud is invariant to permutation

Challenges

PointNet

- Converting point clouds into a dense tensor.





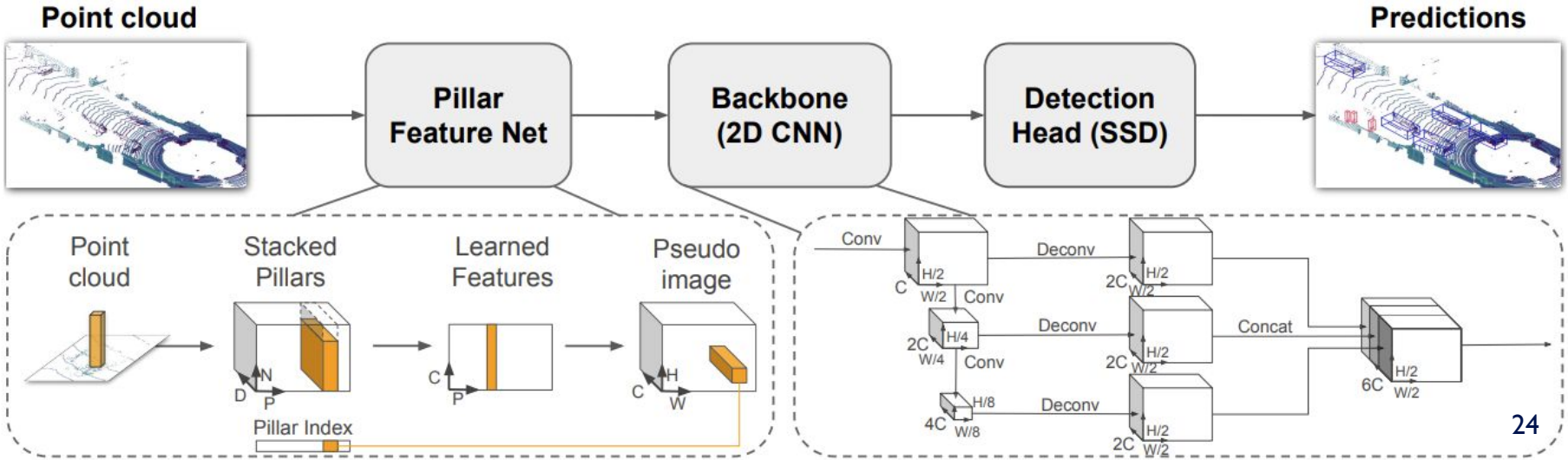
Pointpillars

- Object detection on point cloud data.
- Point cloud data is a list of unsorted (x, y, z) coordinates.

PointPillars: Fast Encoders for Object Detection from Point Clouds

Pointpillars Structure

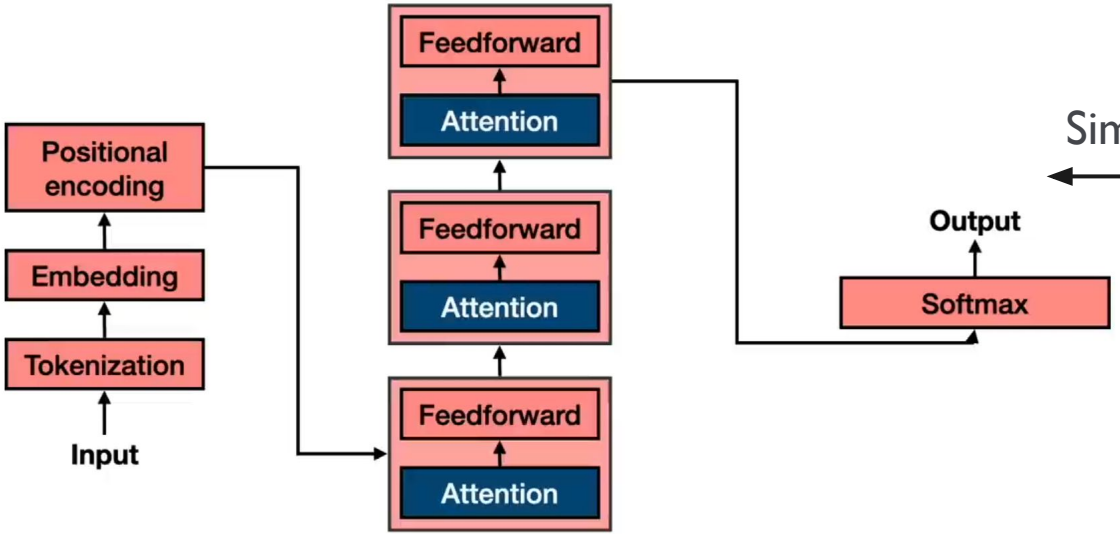
- Combine sparse point cloud into pillars
- Align the pillars into pseudo image.
- Still using 2D CNN to extract feature and a detection head to learn the detections.



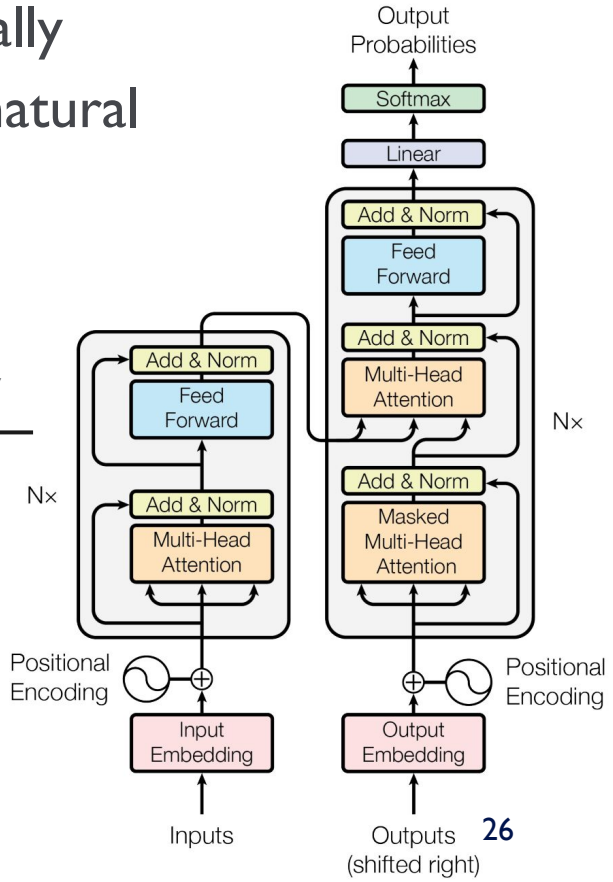
Transformer for Vision

Transformers

- Compared with CNNs, Transformers were originally designed to deal with time-series inputs, such as natural languages.

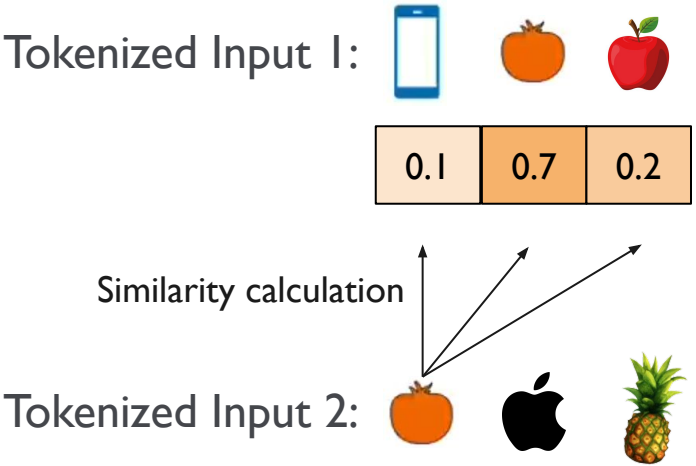


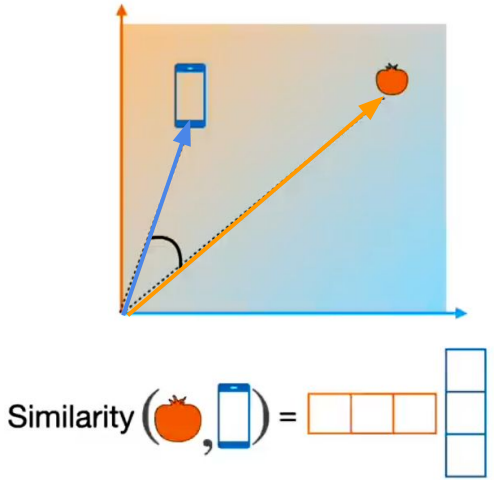
Simplify



Attention Mechanism

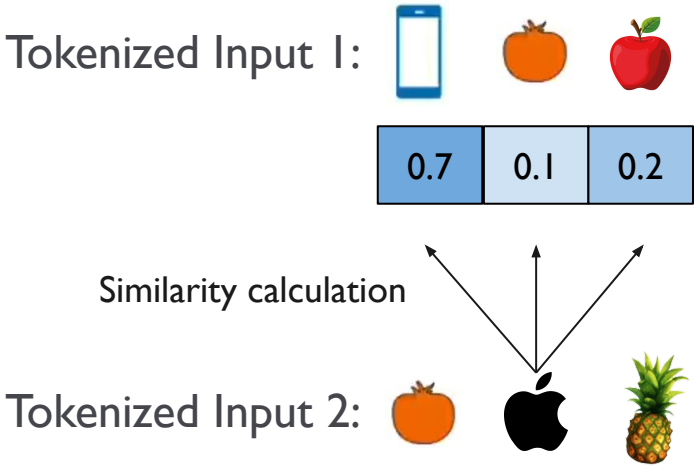
- An attention layer can be thought of as learning a particular kind of **weighing function** in our regression estimate.
- **Dot-product** can be an easy way to calculate the similarity.





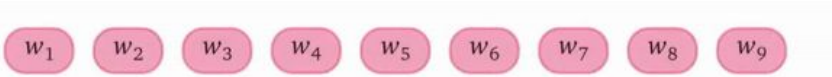
Attention Mechanism

- An attention layer can be thought of as learning a particular kind of **weighing function** in our regression estimate.
- **Dot-product** can be an easy way to calculate the similarity.



Attention Calculation (Repeat for every patch)

Self-attention

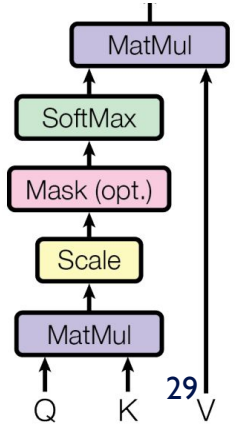


- Query matrix $Q^{(l)} = x^{(l)} W_Q^{(l)}$
- Key matrix $K^{(l)} = x^{(l)} W_K^{(l)}$
- Value matrix $V^{(l)} = x^{(l)} W_V^{(l)}$

$$SA(Q^{(l)}, K^{(l)}, V^{(l)}) = A^{(l)} = \text{softmax}\left(\frac{Q^{(l)} K^{(l)^T}}{\sqrt{d_k}}\right) V^{(l)} = \text{softmax}\left(\frac{x W_Q {W_K}^T x^T}{\sqrt{d_k}}\right) x W_V$$

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

For stable training



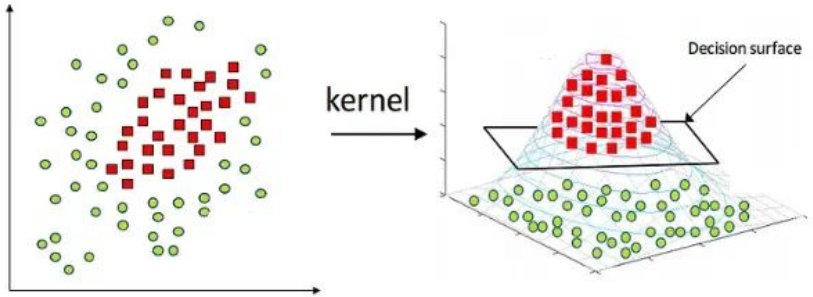
- Why do they use different W to transform input into different spaces first?

Intuition - Analogy to Kernel regression

- Given a dataset $(\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$
- $\mathbf{x}_i$ is input, $y_i = f(\mathbf{x}_i)$ is label.
- Kernel regression constructs an estimator:

$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^n K(\mathbf{x},\mathbf{x}_i) y_i}{\sum_{i=1}^n K(\mathbf{x},\mathbf{x}_i)}$$

- Kernel matrix K maps the input into another space where **similarity** between the input x_i and data can be measured.



Intuition - Analogy to Kernel regression

Kernel Regression:

$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i) y_i}{\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i)}$$

Attention:

$$SA(Q^{(l)}, K^{(l)}, V^{(l)}) = A^{(l)} = \text{softmax}\left(\frac{Q^{(l)} K^{(l)^T}}{\sqrt{d_k}}\right) V^{(l)}$$

for Kernel Regression		for Attention
train data x_i	$\equiv$	k_i keys
test data x	$\equiv$	q query
targets y_i	$\equiv$	v_i values
kernel $k(x_i, x)$	$\equiv$	$e^{k_i^\top q}$ exp-dot-product
non-linear function → $\sigma(k_i^\top q)$	$\equiv$	$\frac{e^{k_i^\top q}}{\sum_j e^{k_j^\top q}}$. ← softmax

Q & K projection

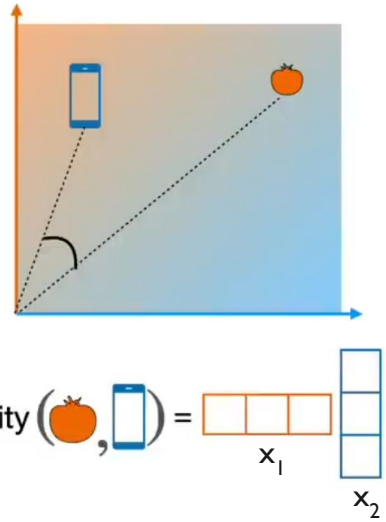
Kernel Regression:

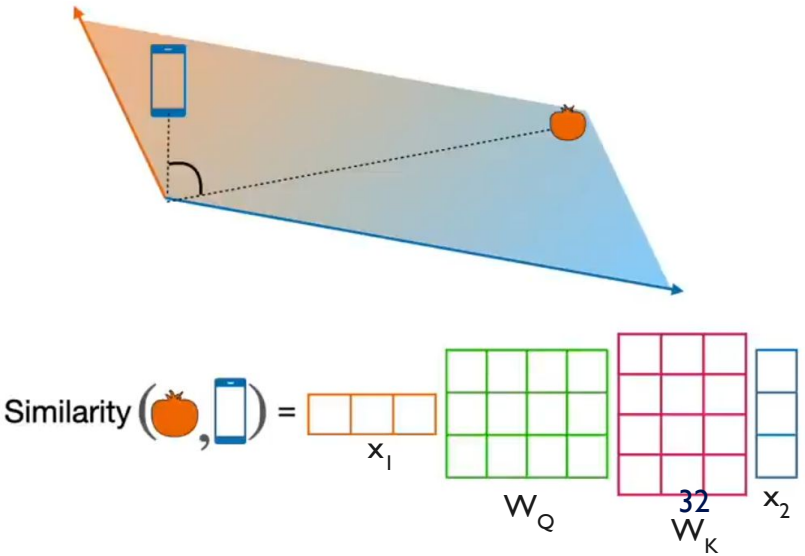
$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i) y_i}{\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i)}$$

Attention:

$$SA(Q^{(l)}, K^{(l)}, V^{(l)}) = A^{(l)} = \text{softmax}\left(\frac{Q^{(l)} K^{(l)^T}}{\sqrt{d_k}}\right) V^{(l)}$$

- The Q and K matrix functions as a learnable kernel.



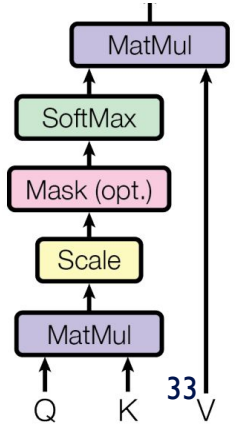


Self-attention

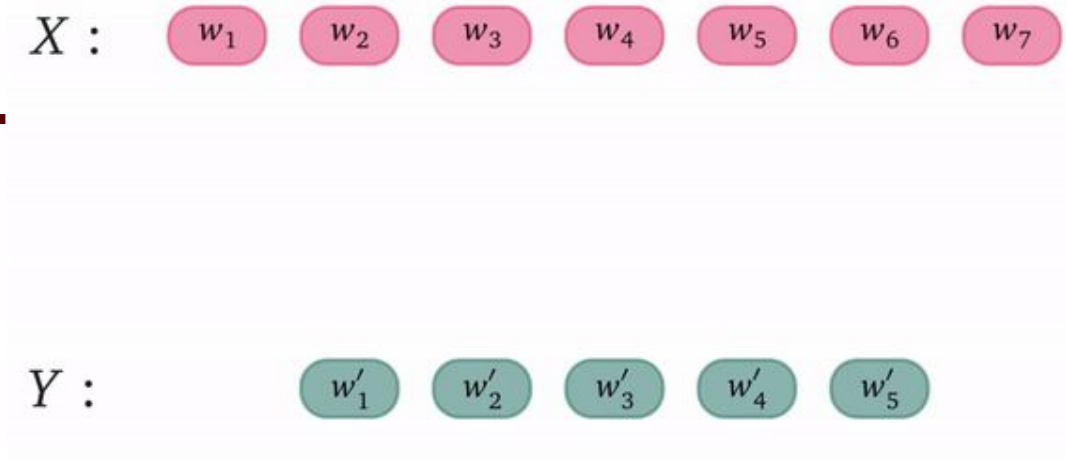
- Query matrix $Q^{(l)} = x^{(l)} W_Q^{(l)}$
- Key matrix $K^{(l)} = x^{(l)} W_K^{(l)}$
- Value matrix $V^{(l)} = x^{(l)} W_V^{(l)}$

$$SA(Q^{(l)}, K^{(l)}, V^{(l)}) = A^{(l)} = \text{softmax}\left(\frac{Q^{(l)} {K^{(l)}}^T}{\sqrt{d_k}}\right) V^{(l)} = \text{softmax}\left(\frac{x W_Q {W_K}^T x^T}{\sqrt{d_k}}\right) x W_V$$

- For self-attention, QK^T functions as **similarity measurements** between tokens, softmax is for normalization, and V functions as **output space**.

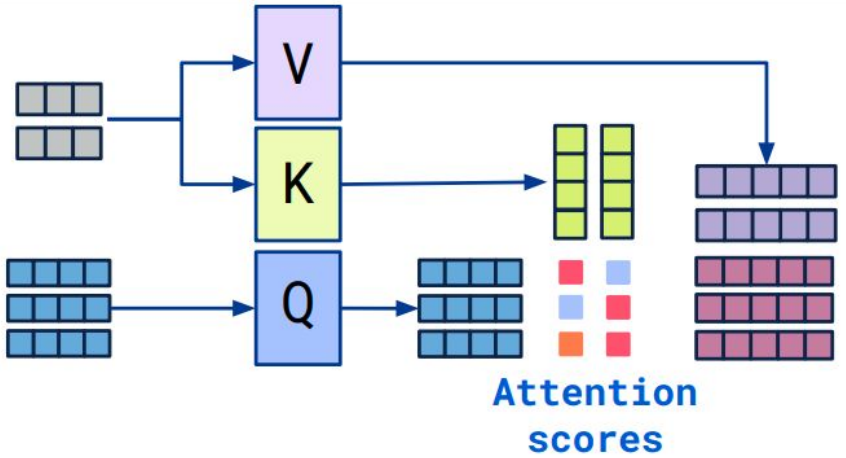


Cross-attention



- Query matrix $Q = x_1 W_Q$
- Key matrix $K = x_2 W_K$
- Value matrix $V = x_2 W_V$

- For cross-attention, K and V are from **different** inputs, providing more information communication.
- The K and V are usually from **context information** or conditional inputs.



Cross-attention in stable diffusion

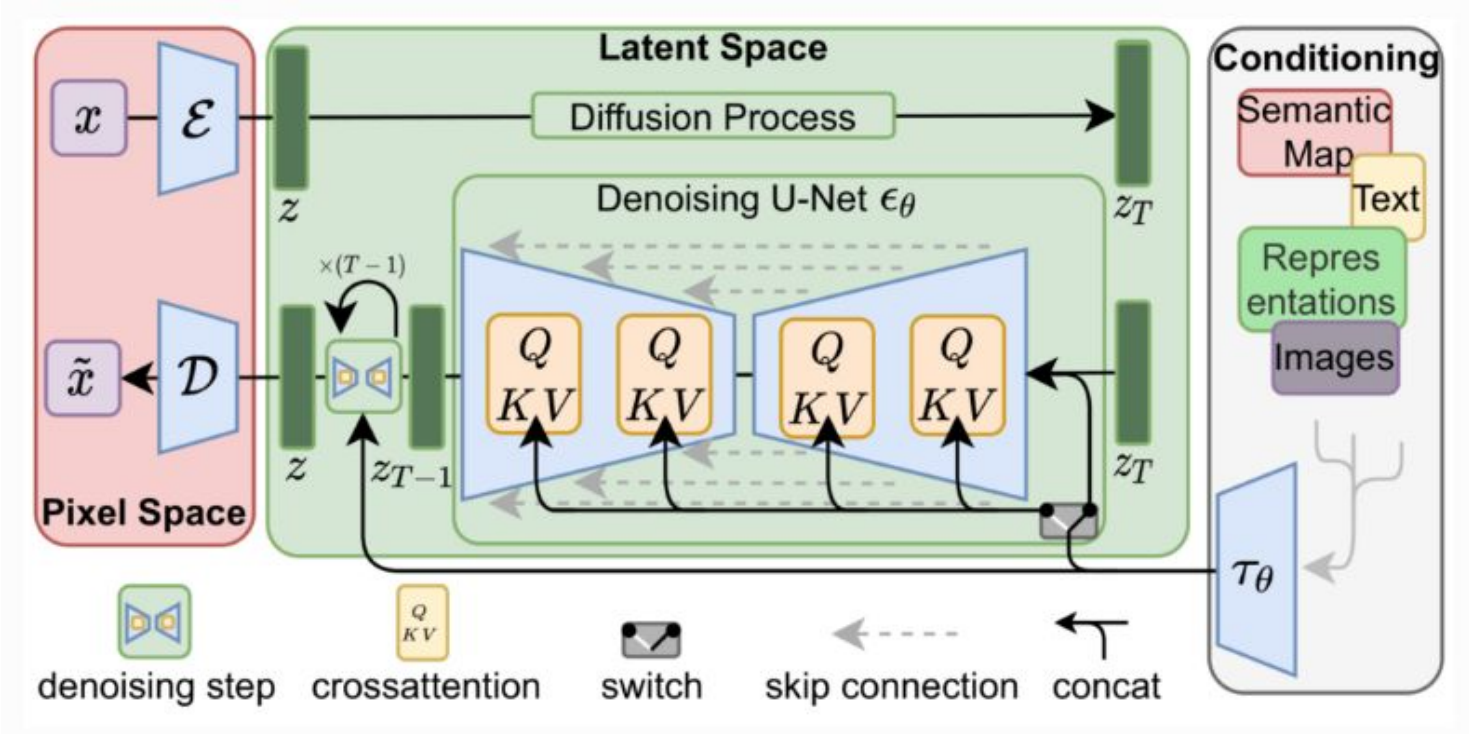
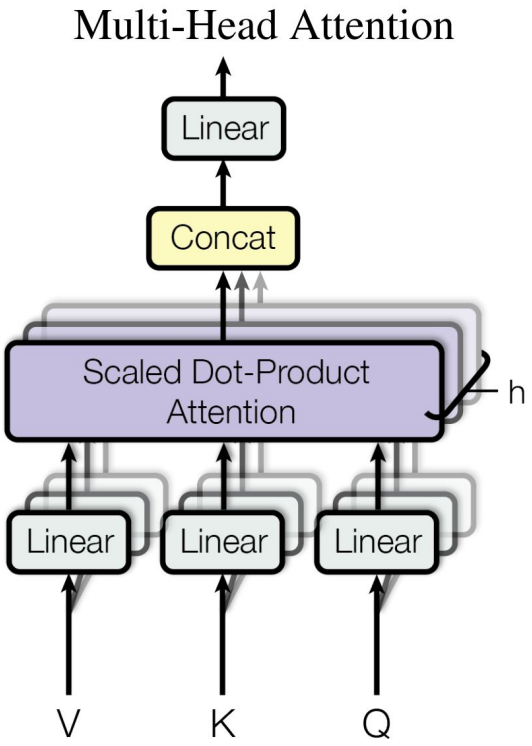


Figure 3. We condition LDMs either via concatenation or by a more general cross-attention mechanism.

Multi-head Attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Multiple heads are like channels in CNN, functioning like ensembling the attention mechanism.

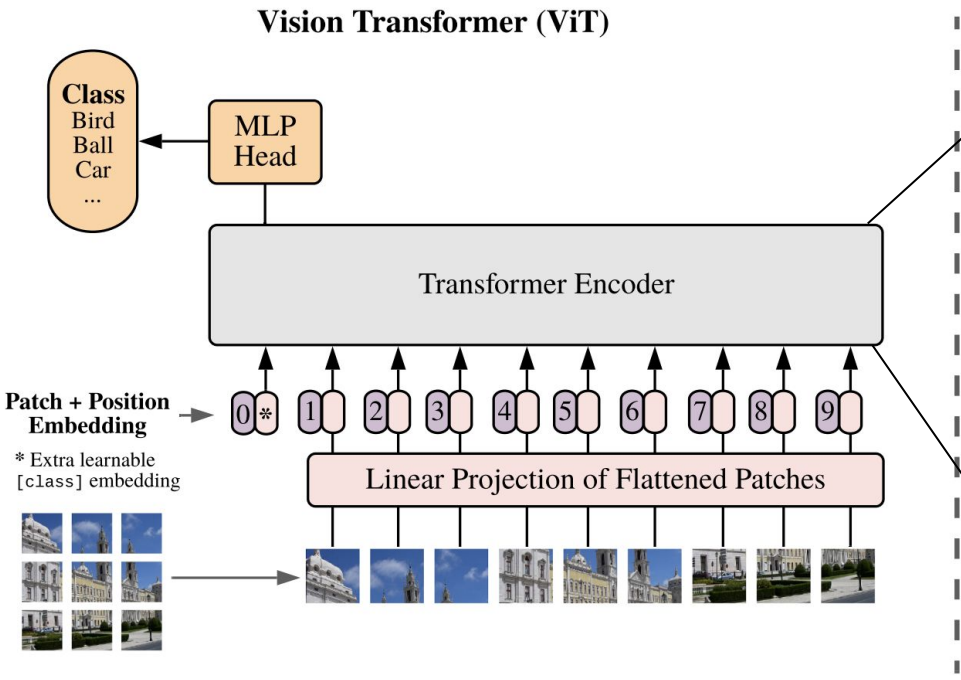


Vision Transformers

- Examples of self-attention

Attention Calculation (Repeat for every patch)

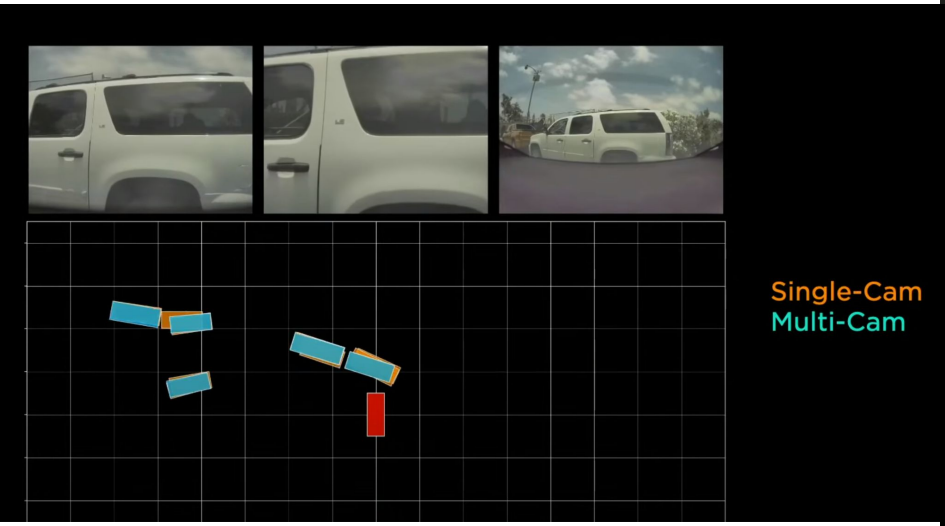
DETR - End to end object detection with transformers (ECCV2020)

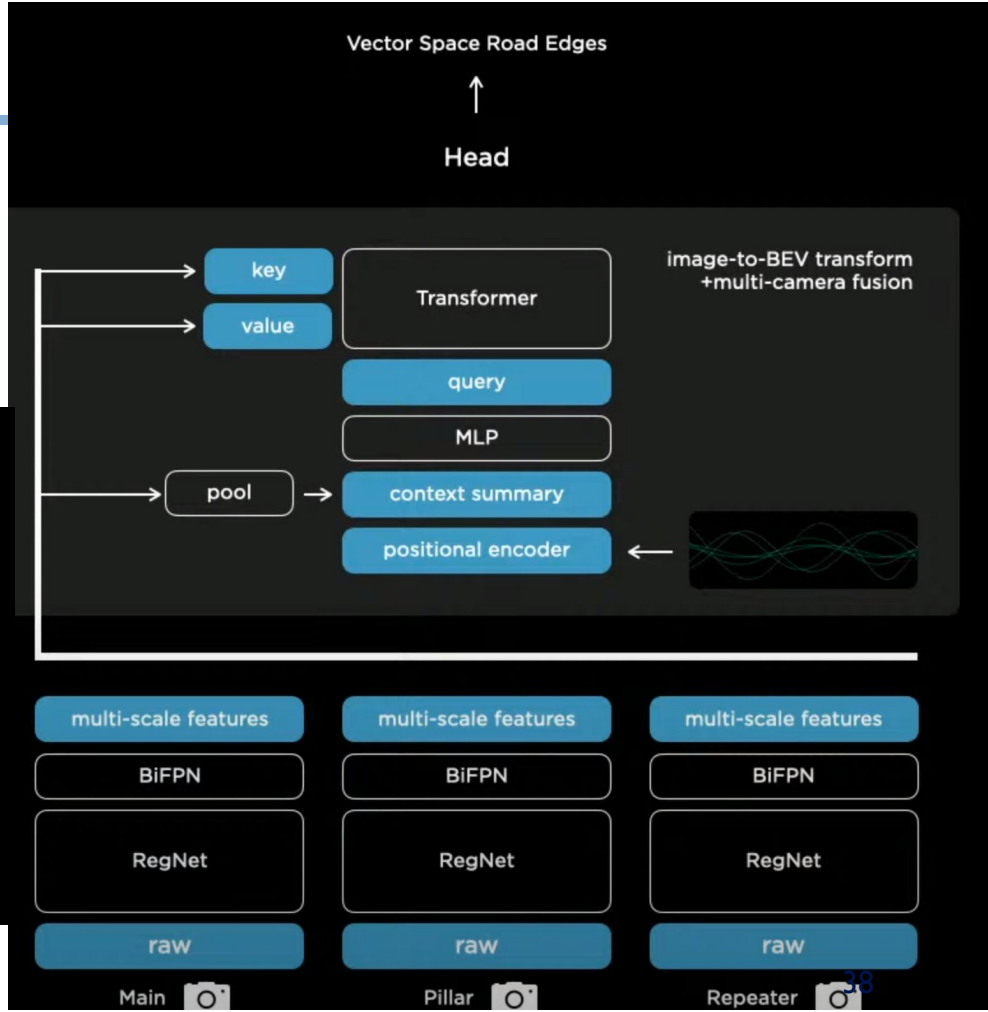


An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (ICLR 2021)

Tesla AI Day 2021

- CNN for feature extraction
- Transformer for context learning.
- An example of cross-attention.

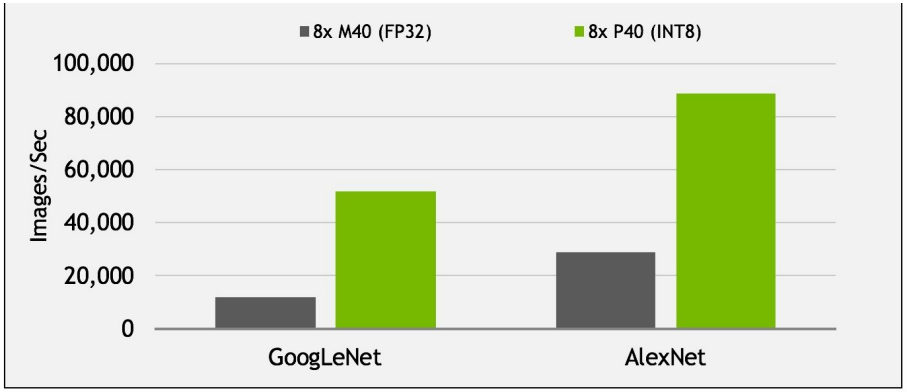




Network Deployment

Network Deployment

- When deploying the network, we want to optimize its size and speed.
- Common techniques includes:
  - Pruning
  - Quantization
  - Platform Optimizations
- However, robustness and safety may be concerns.



GoogLeNet, AlexNet, batch size = 128, CPU: Dual Socket Intel E5-2697v4

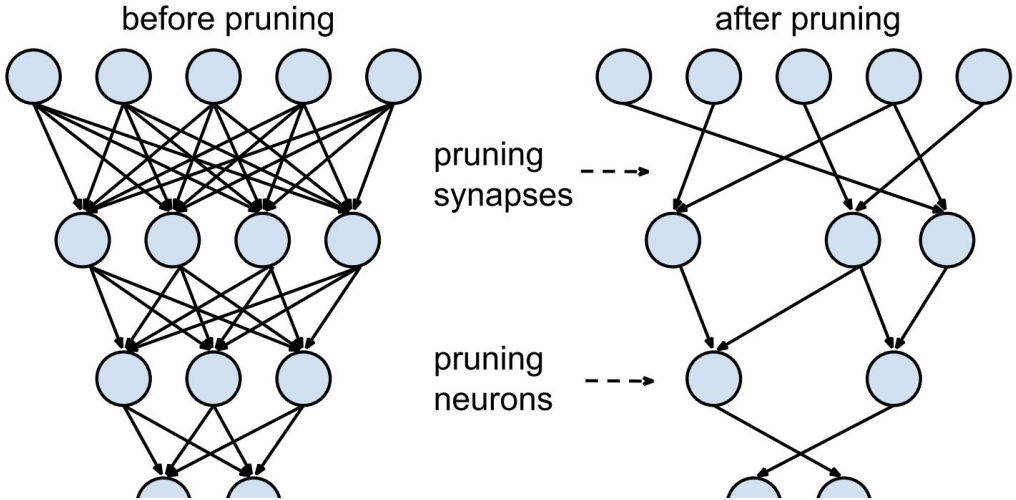
FP32 vs. INT8

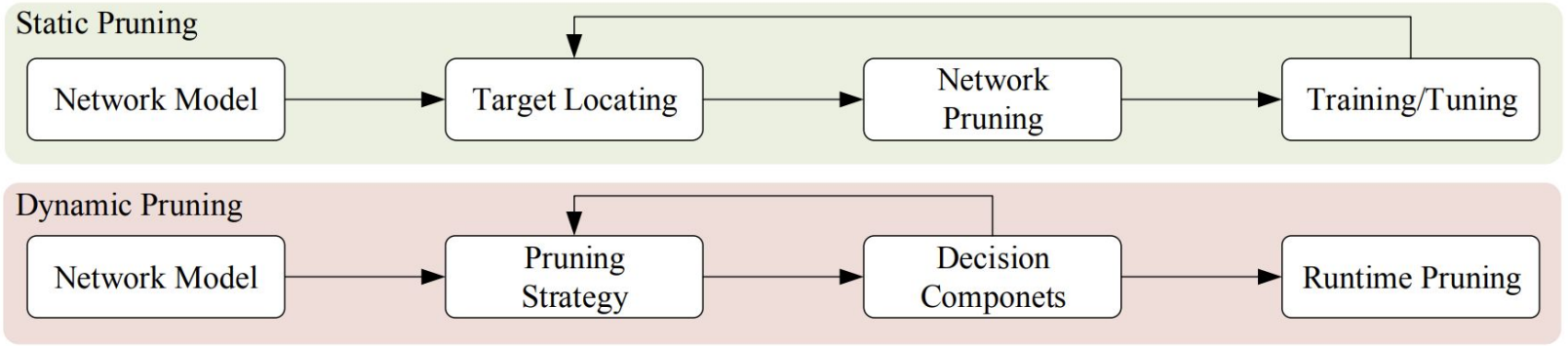
Network	Original Size		Compressed Size	Compression Ratio	Original Accuracy		Compressed Accuracy
Inception-V3	91MB	→	4.2MB	**22x**	93.56%	→	93.67%
ResNet-50	97MB	→	5.8MB	**17x**	92.87%	→	93.04%

Pruning not necessarily loses accuracy

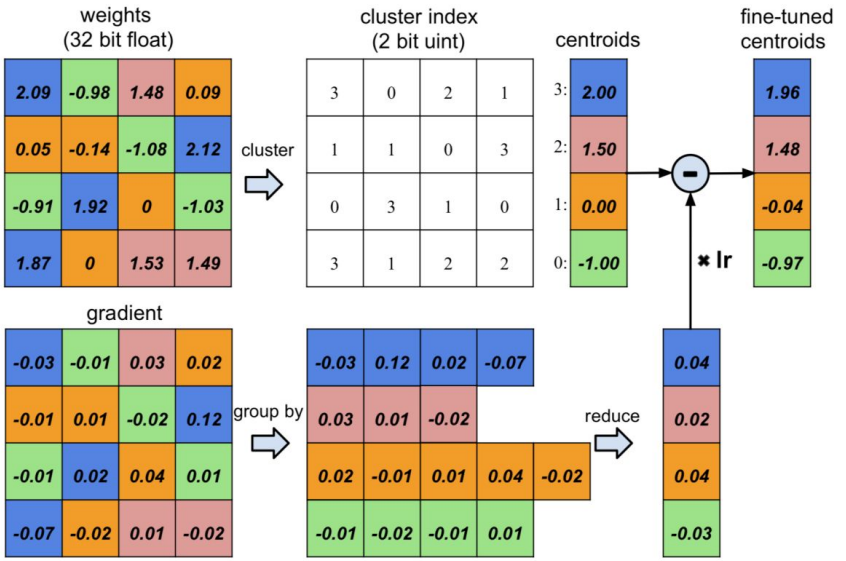
Network Pruning

- Pruning is removing redundant neurons.
- Static pruning: performed offline prior to inference.
- Dynamic pruning: performed at runtime

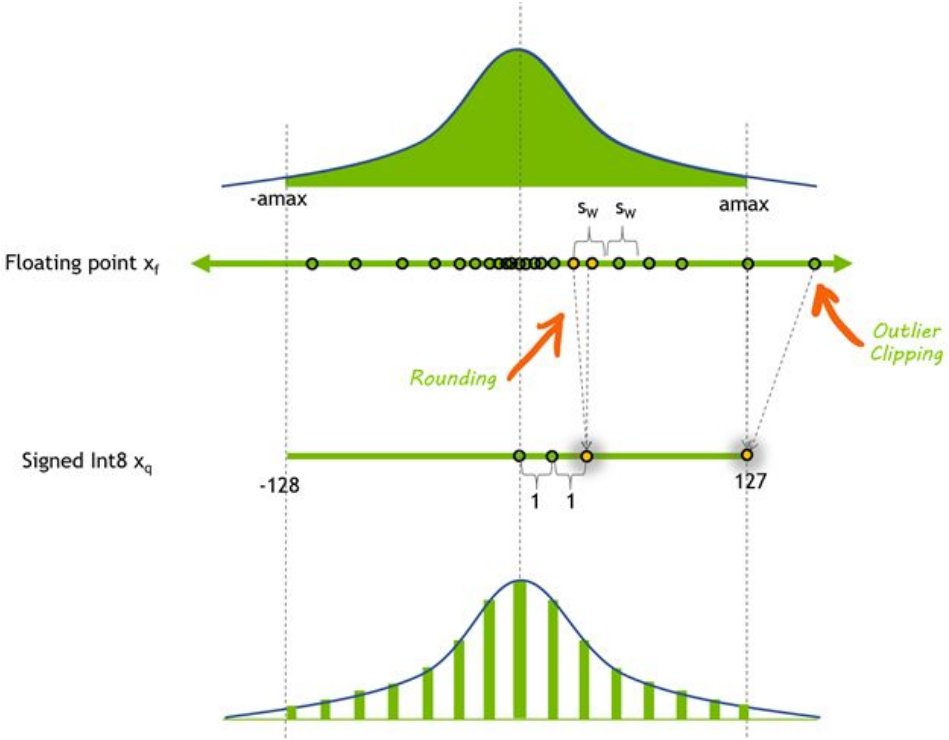




Network Quantization



Quantization



8-bit signed integer quantization of a floating-point tensor

Deployment Platforms

CPUs

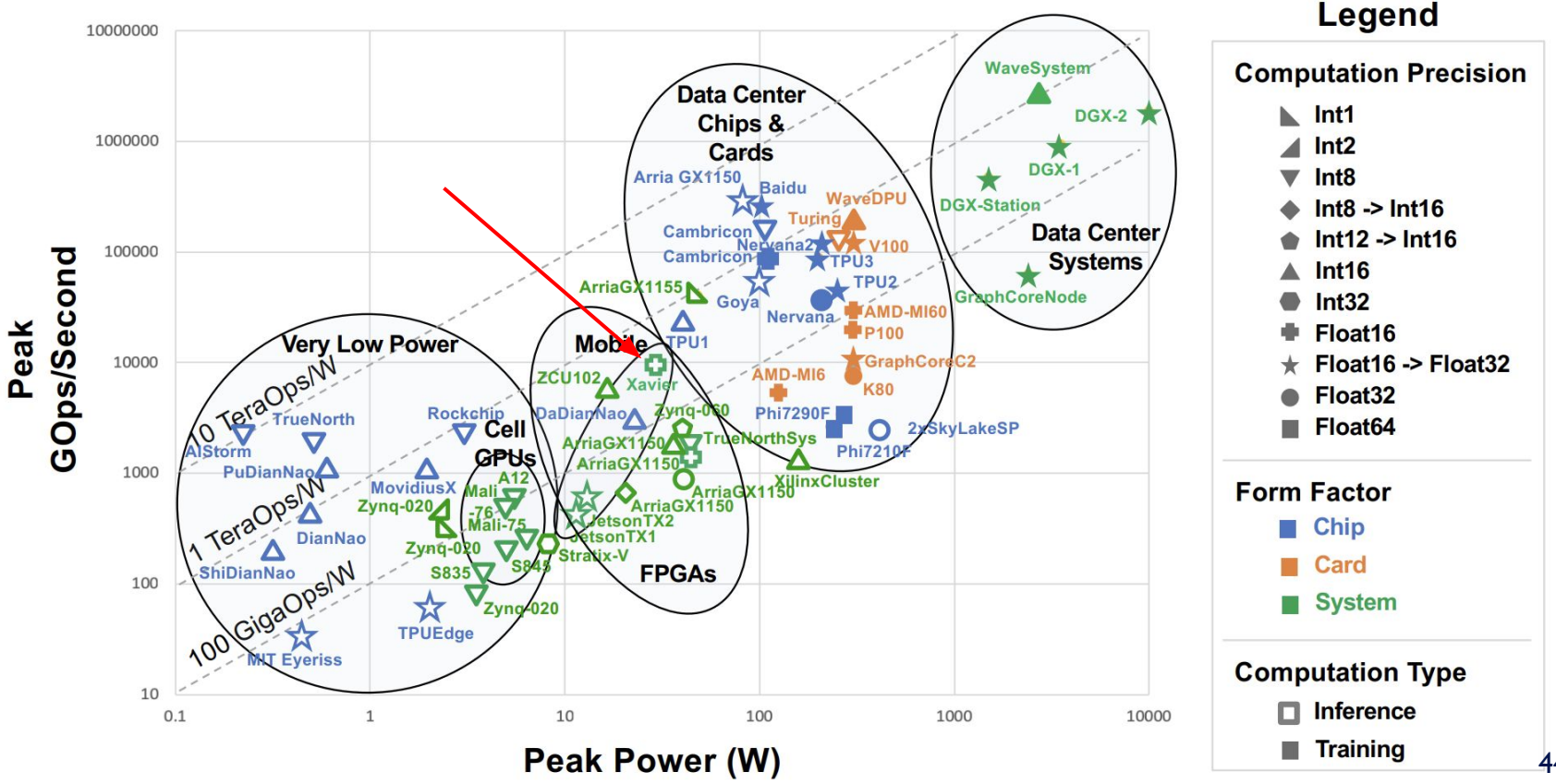
GPUs

Field Programmable Gate Arrays (FPGA)

Mobile SoCs

or other ASICs

Deployment Platforms

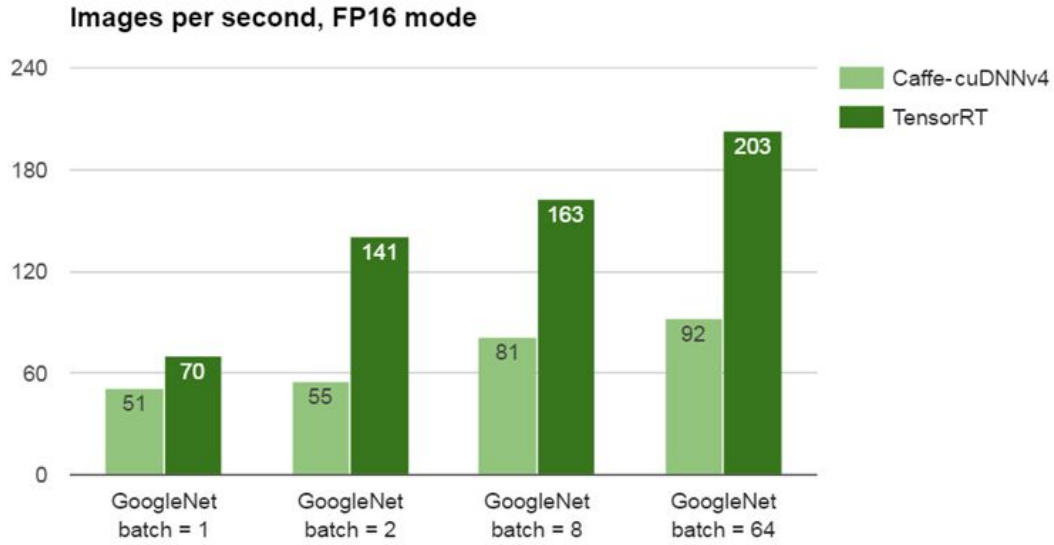


Platform Optimizations

- Lookup Table
- Memory Optimization
  - Some chips have very limited memory bandwidth and can be a bottleneck in performance.
- Special Hardware Optimization Libraries (e.g. cuDNN, OpenVINO)
  - These libraries may use special instructions in the chip so it can be much faster.
- Open Neural Network Exchange (ONNX) is an open-source tool to parse AI models written for a variety diverse frameworks.

TensorRT

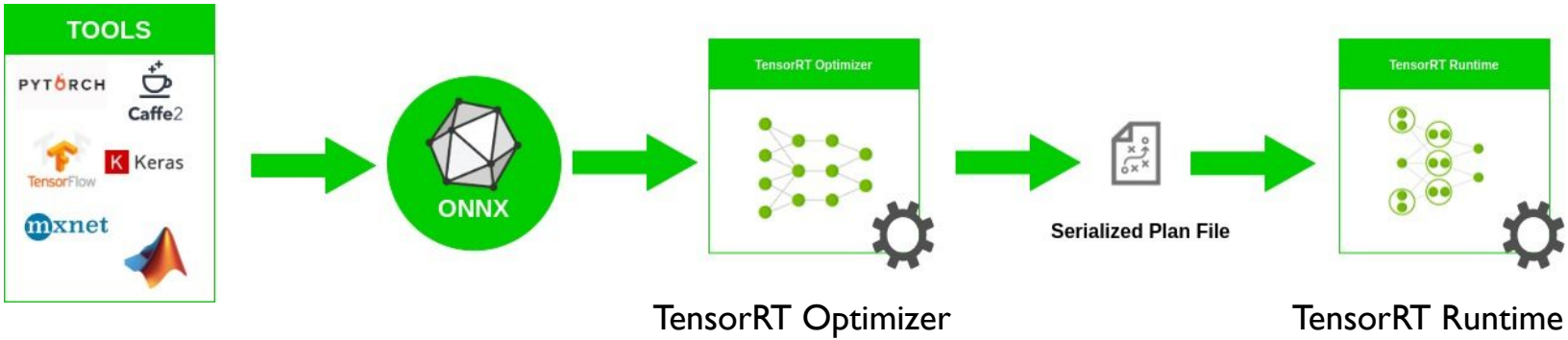
- NVIDIA TensorRT is an SDK for high-performance deep learning inference. It includes a deep learning inference optimizer and runtime.
- Black box optimizer but works great.
- Easy to use.
- Only on NVIDIA platforms.



TensorRT on Jetson TX1

TensorRT Engine Generation

- The serialized engine is platform-specific. You can't use the generated engine on a different hardware.
- Support fp16, int8 quantization.
- Support runtime change of the input dimension, called dynamic shape.



References

- https://www.cs.princeton.edu/courses/archive/spring16/cos495/slides/DL_lecture1_feedforward.pdf
- https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/
- https://spectrum.ieee.org/driver-assistance-package
- https://www.youtube.com/watch?v=IIReDnbLQAE
- https://www.youtube.com/watch?v=9s_FpMpdYW8&t=22s
- https://arxiv.org/pdf/2001.06280.pdf
- https://arxiv.org/pdf/1812.05784.pdf
- https://jacobgil.github.io/deeplearning/pruning-deep-learning
- https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/
- https://www.youtube.com/watch?v=UPtG_38Oq8o&t=1473s
- https://www.youtube.com/watch?v=mmzRYGCfTzc&t=531s
- https://www.youtube.com/watch?v=XfpMkf4rD6E
- https://pratikac.github.io/pub/23_ese546.pdf