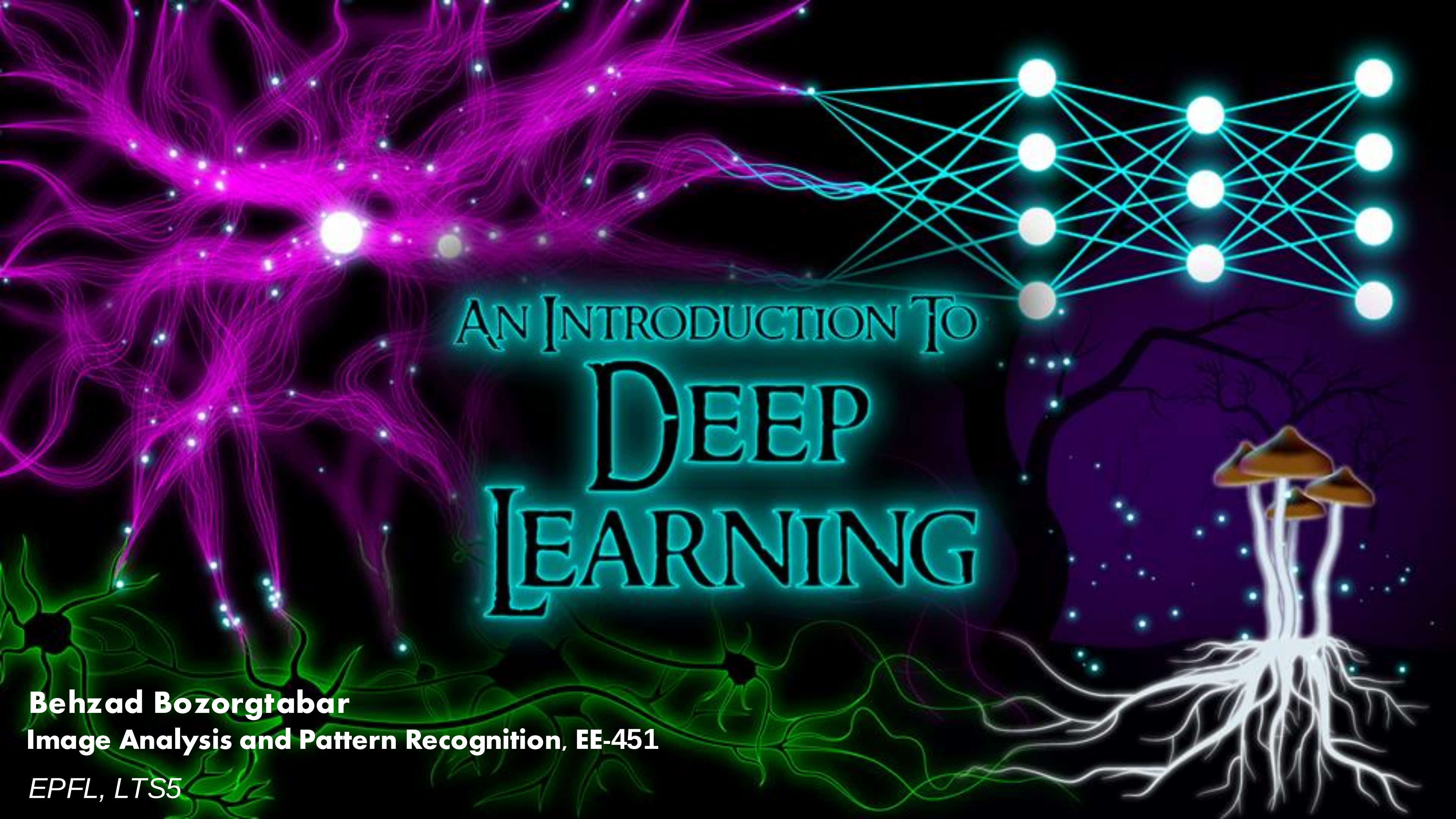
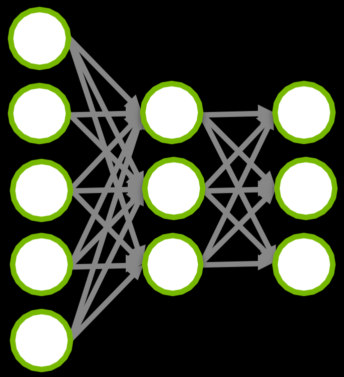
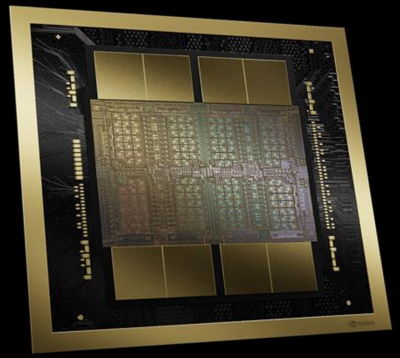


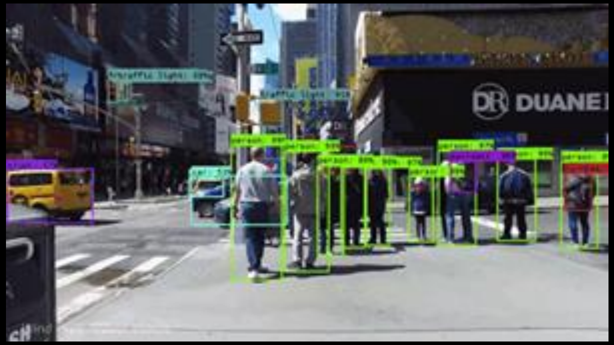
Behzad Bozorgtabar
Image Analysis and Pattern Recognition, EE-451
EPFL, LTS5

Deep Learning for Real World Problems

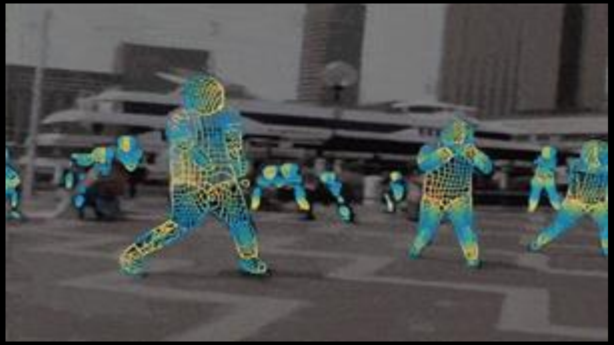




Object Detection



Human Understanding

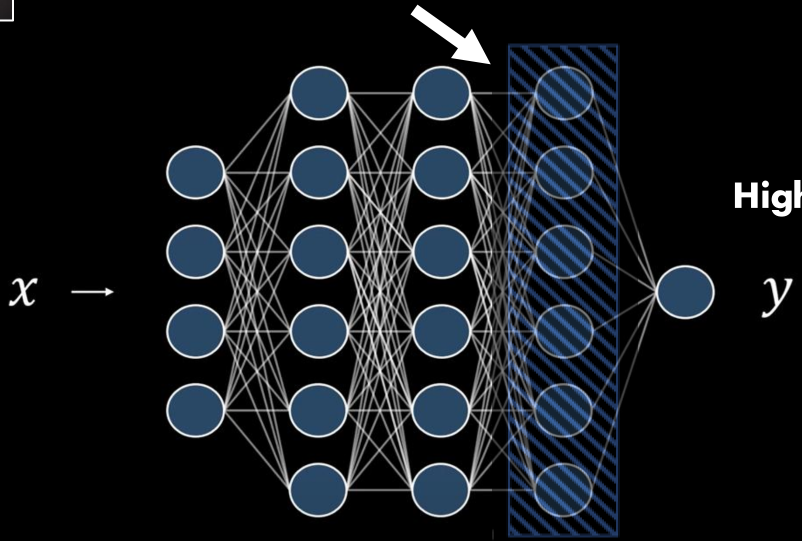


Autonomous Driving

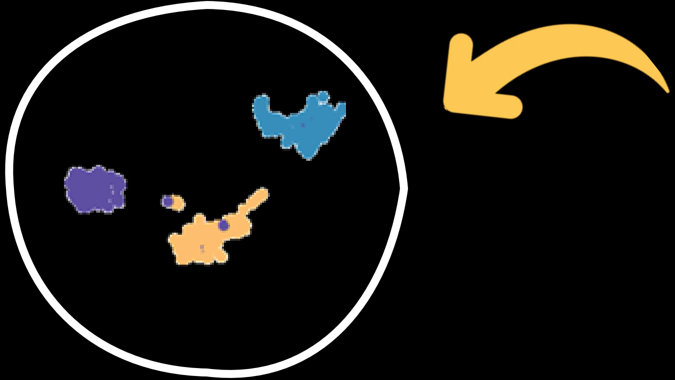
Datapoint 1 — Representation 1

Datapoint 2 — Representation 2

Datapoint 3 — Representation 3



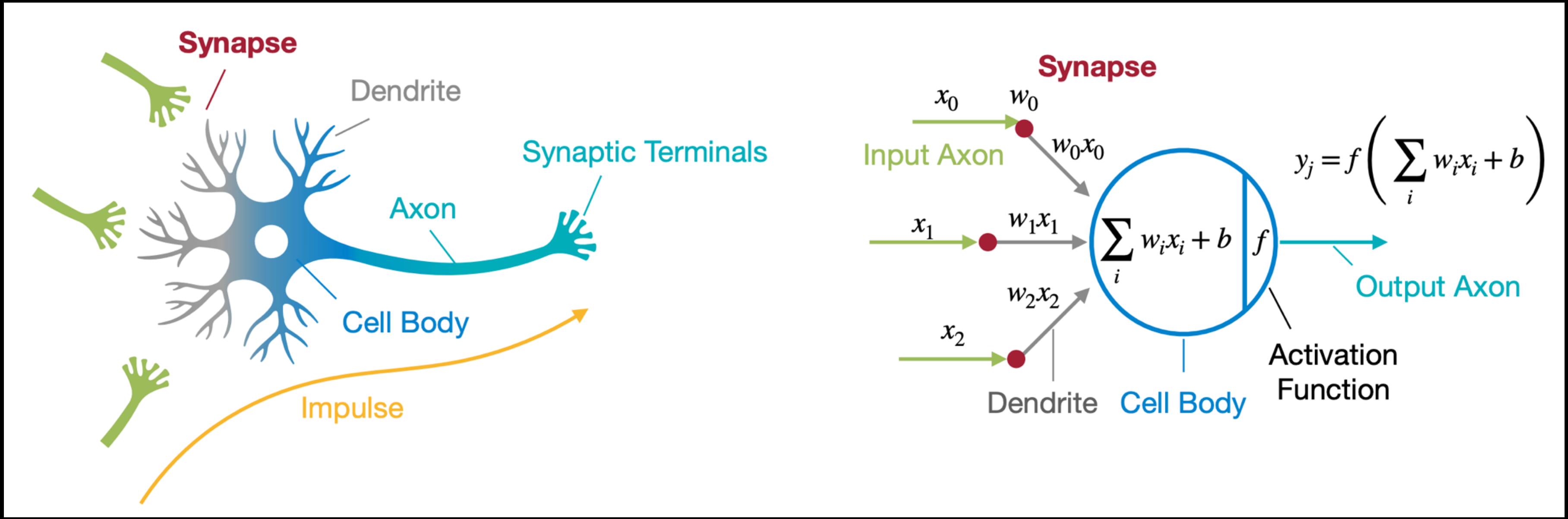
Representation learning



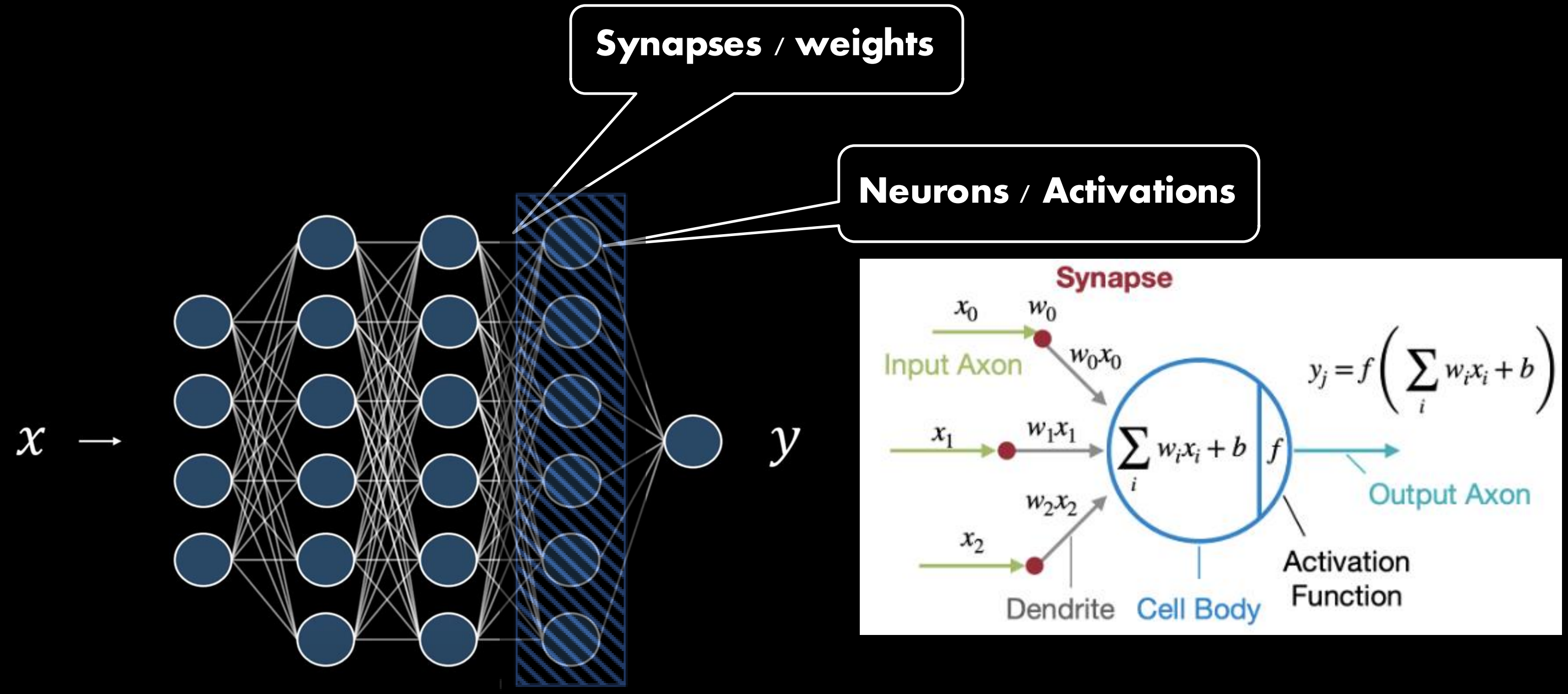
Embedding Space

High-level representations are typically nuisance-invariant

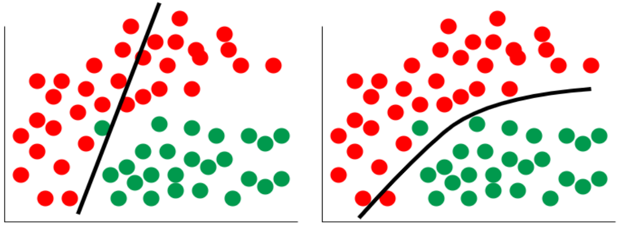
Neuron and Synapse



Neural Network



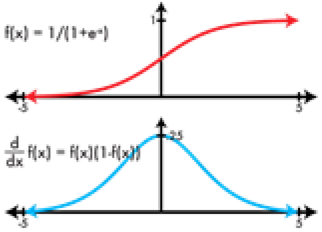
Activation Functions



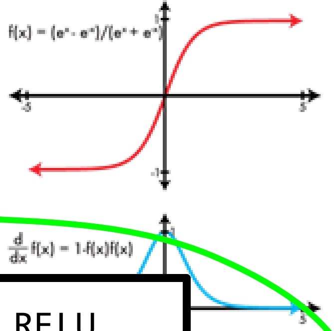
Decision boundary for linear activation functions

Decision boundary for non-linear activation functions

logistic

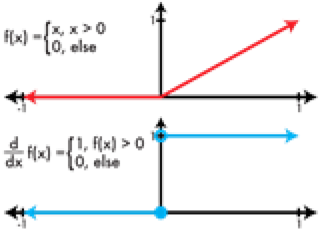


tanh

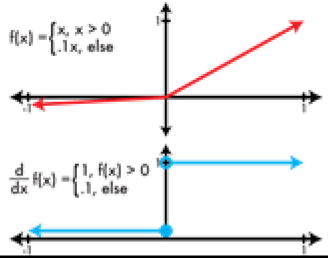


Good ones

REctified Linear Unit (RELU)



Leaky RELU



Fully Connected (Dense) Layer

Each output neuron is connected to all previous layer neurons

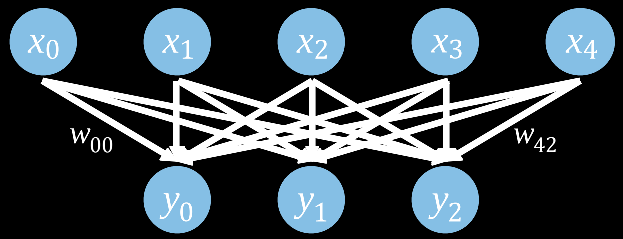
- **Shape of Tensors:**

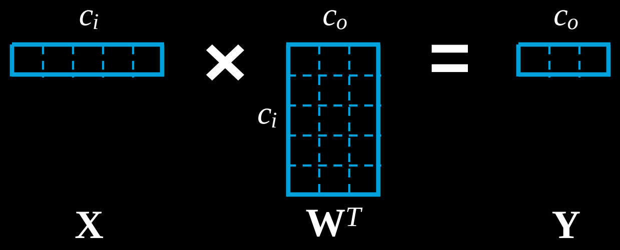
  Input Features X : $(1, c_i)$

  Output Features Y : $(1, c_o)$

  Weights W : (c_o, c_i)

  Bias b : $(c_o,)$

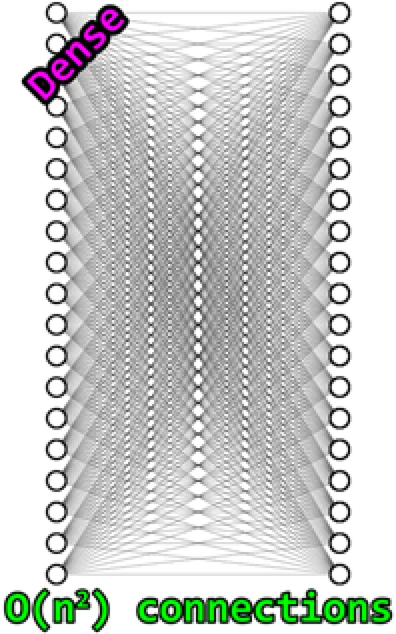




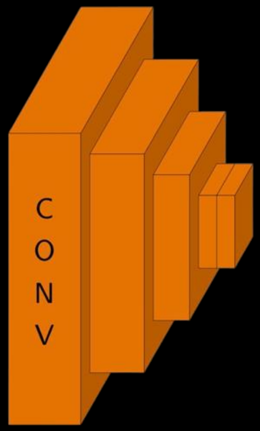
Scaling Issue in Fully Connected Layers

The number of weights grows quadratically with the number of neurons

Complexity of handling image data



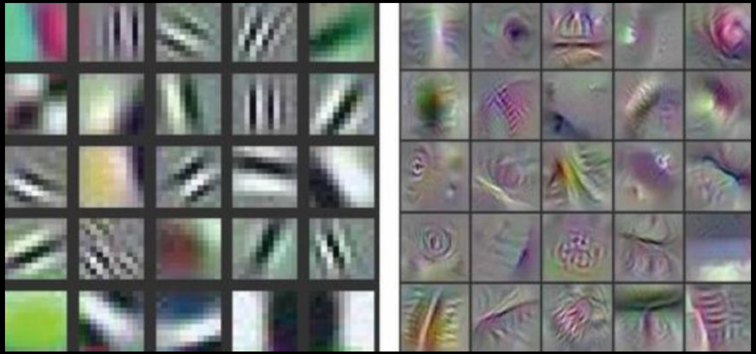
Convolutional Neural Networks and Intuition



Intuition Behind Convolution Layer (1)

Restricting the degrees of freedom

- A structured layer to process a small region with fewer weights (many useful features are local)



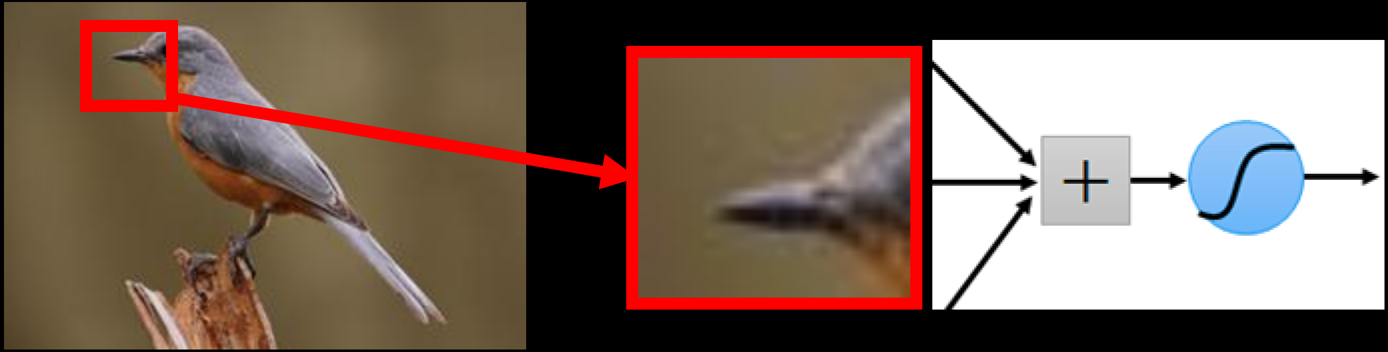
Layer 1:
edge detectors?

Layer 2:
beak? wing?

Intuition Behind Convolution Layer (1)

Restricting the degrees of freedom

- A structured layer to process a small region with fewer weights (many useful features are local)

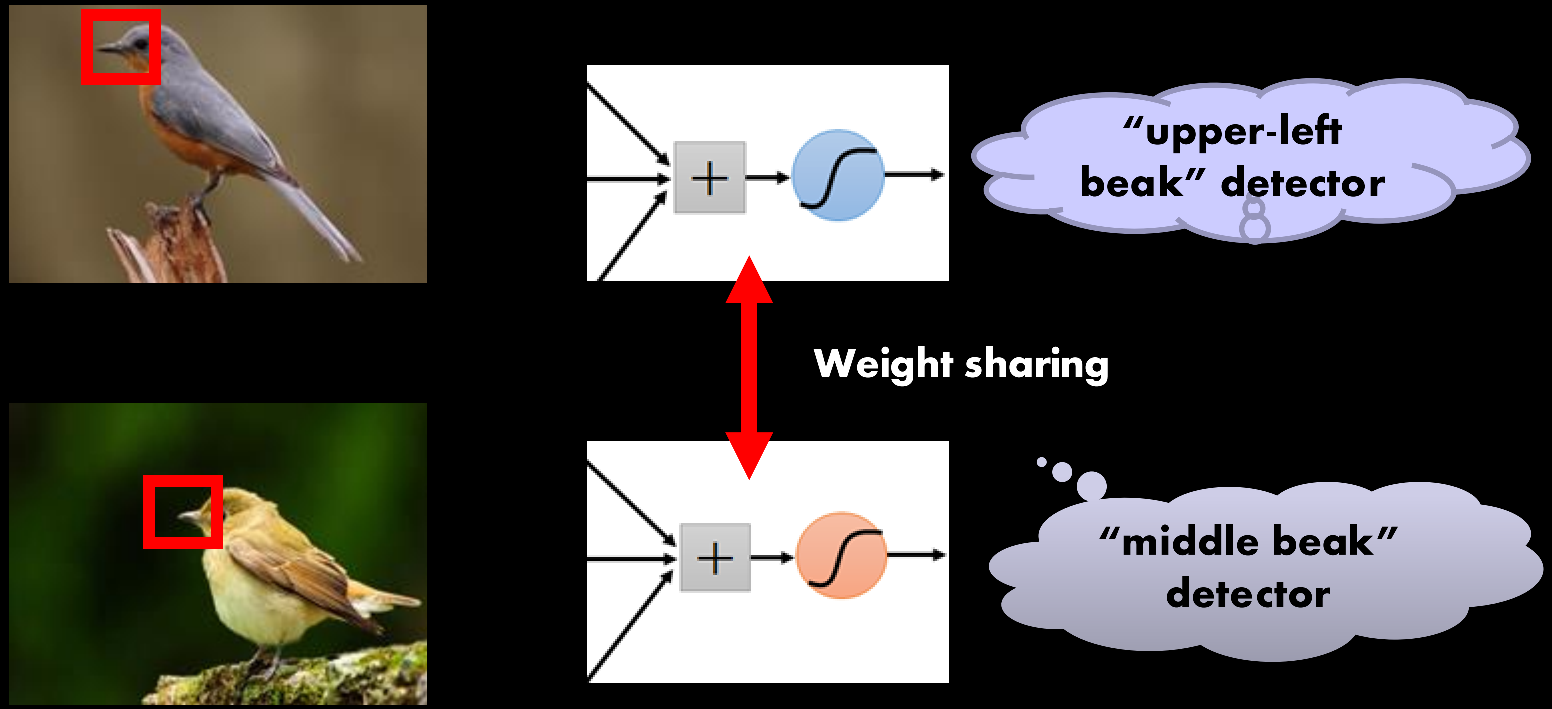


"beak" detector

Intuition Behind Convolution Layer (2)

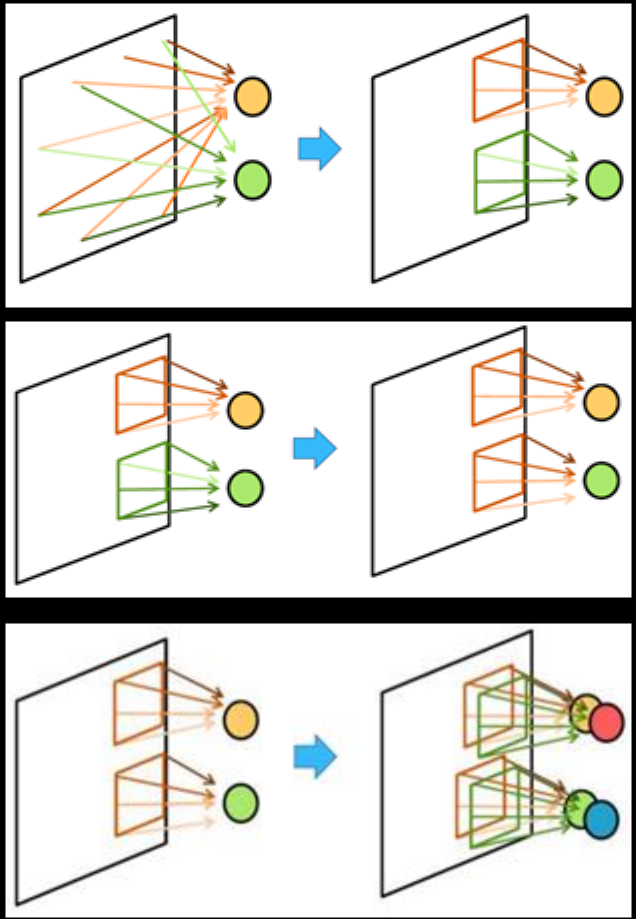
Restricting the degrees of freedom

- Weight sharing: using the same weights for different parts of the image



Transitioning from Fully Connected to Convolution Layer

- **Local Connectivity**
- **Weight Sharing**
- **Multiple Feature Detectors**



Connectivity Pattern: Fully Connected vs. Convolution Layer

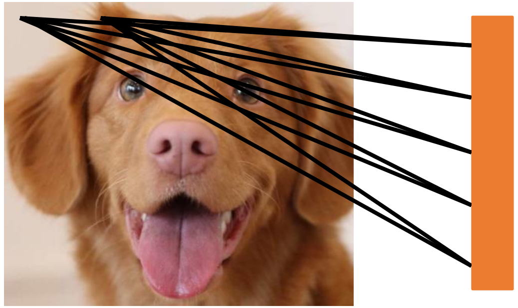
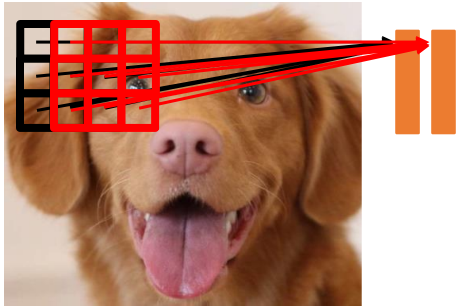


Image is $128 \times 128 \times 3 \ = 49,152$

First layer is 64-dim

$64 \times 49,152 \approx 3,000,000$

FC layer



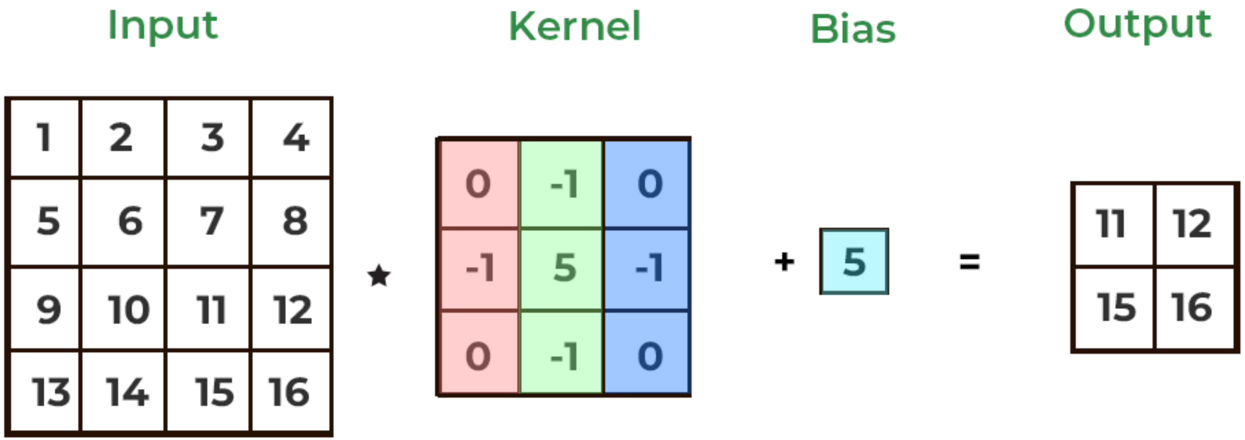
Patch is $3 \times 3 \times 3 = 27$

First layer is 64-dim

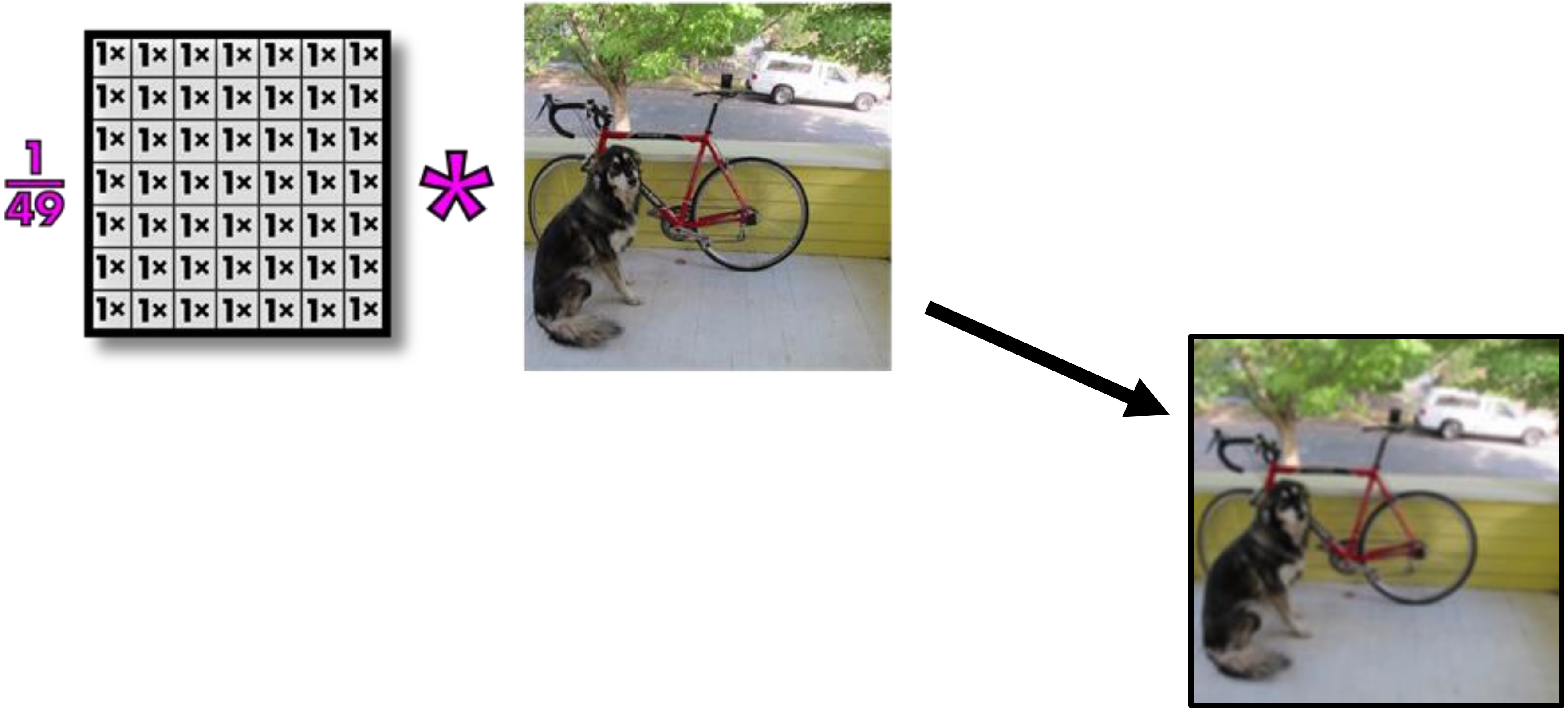
$64 \times 27 = 1728$

Convolution layer

Convolutions?

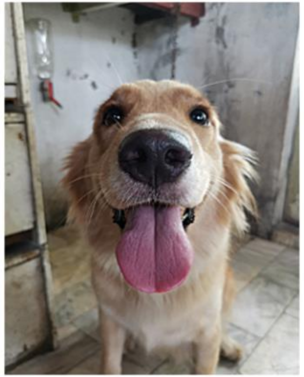


Convolution on Images?



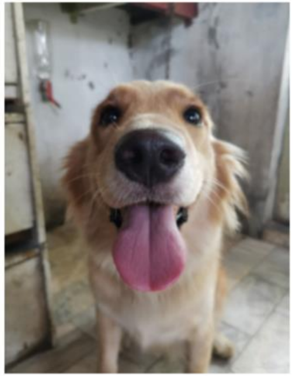
Filter Effects

Input



Edge detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

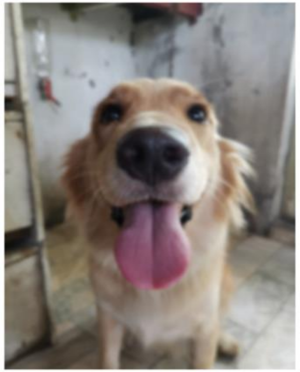


Box mean

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

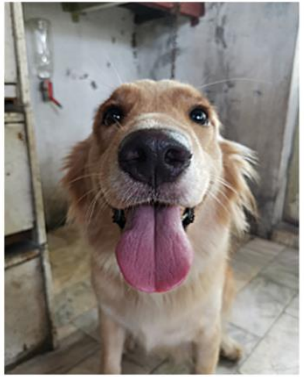


Gaussian blur

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Filter Effects

Input



Edge detection

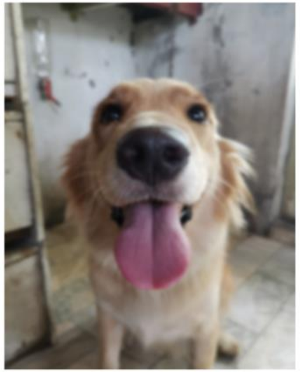
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Box mean

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Gaussian blur

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

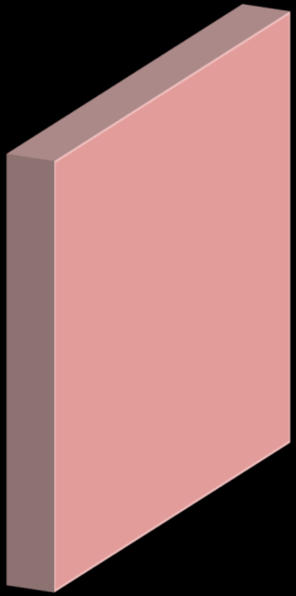
LET'S LEARN THESE FILTERS!

Convolutions on Volumetric Images

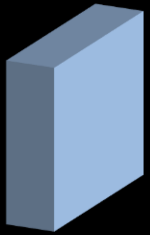
width height depth

image $32 \times 32 \times 3$

Depth dimension *must* match; i.e., filter extends the full depth of the input



filter $5 \times 5 \times 3$



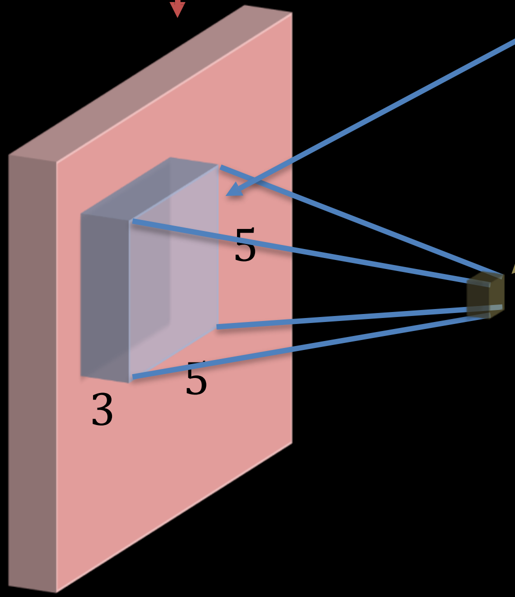
Convolve filter with image
i.e., 'slide' over it and:
– Apply filter at each location
– Compute dot product

Images have depth: e.g., RGB → 3 channels

Convolutions on Volumetric Images

32×32×3 image (X)

filter (weight tensor w)



1 number at time:

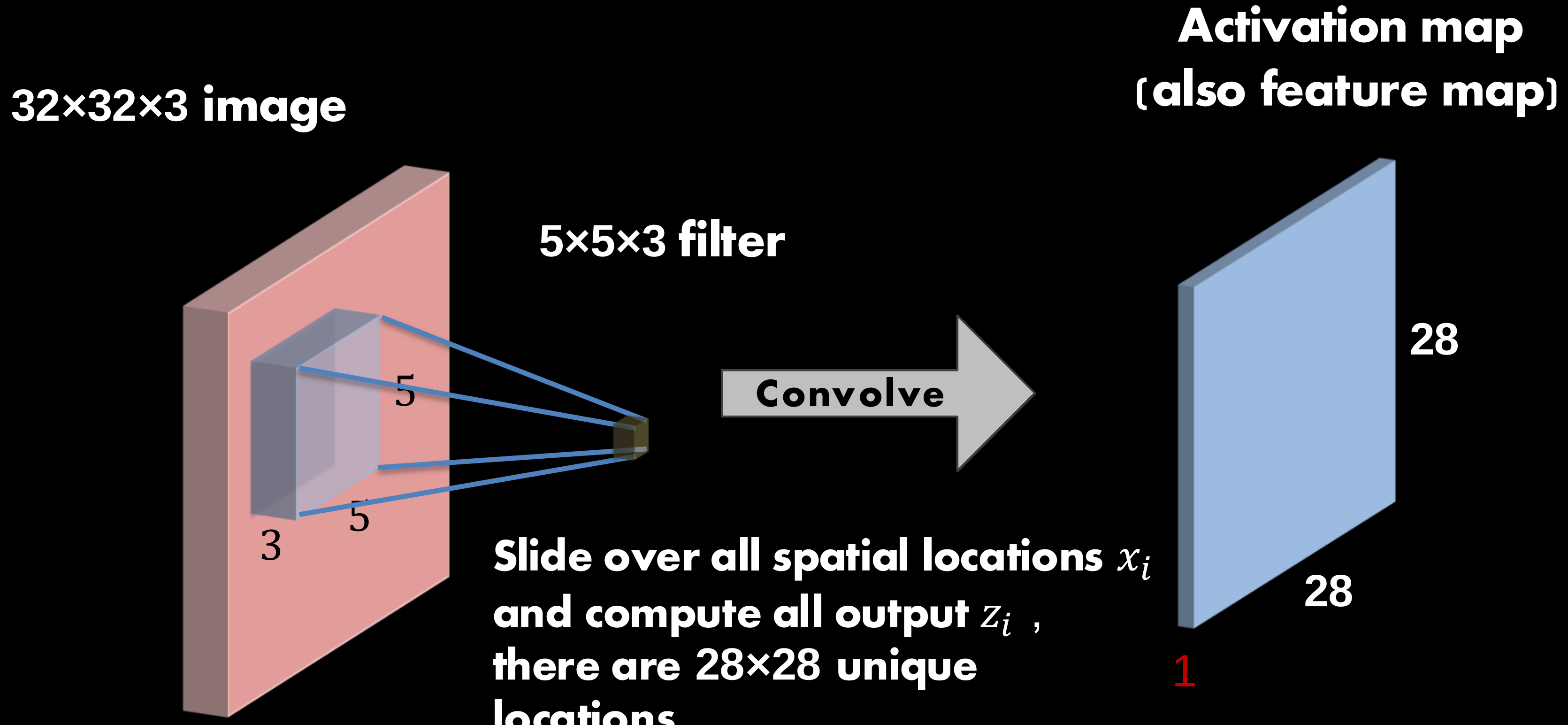
equals to dot product between filter weights w and $x_i - th$ chunk of the image. Here: 5.5.3=75-dim +bias

$$z_i = w^T x_i + b$$

(5×5×3)×1 (5×5×3)×1 1

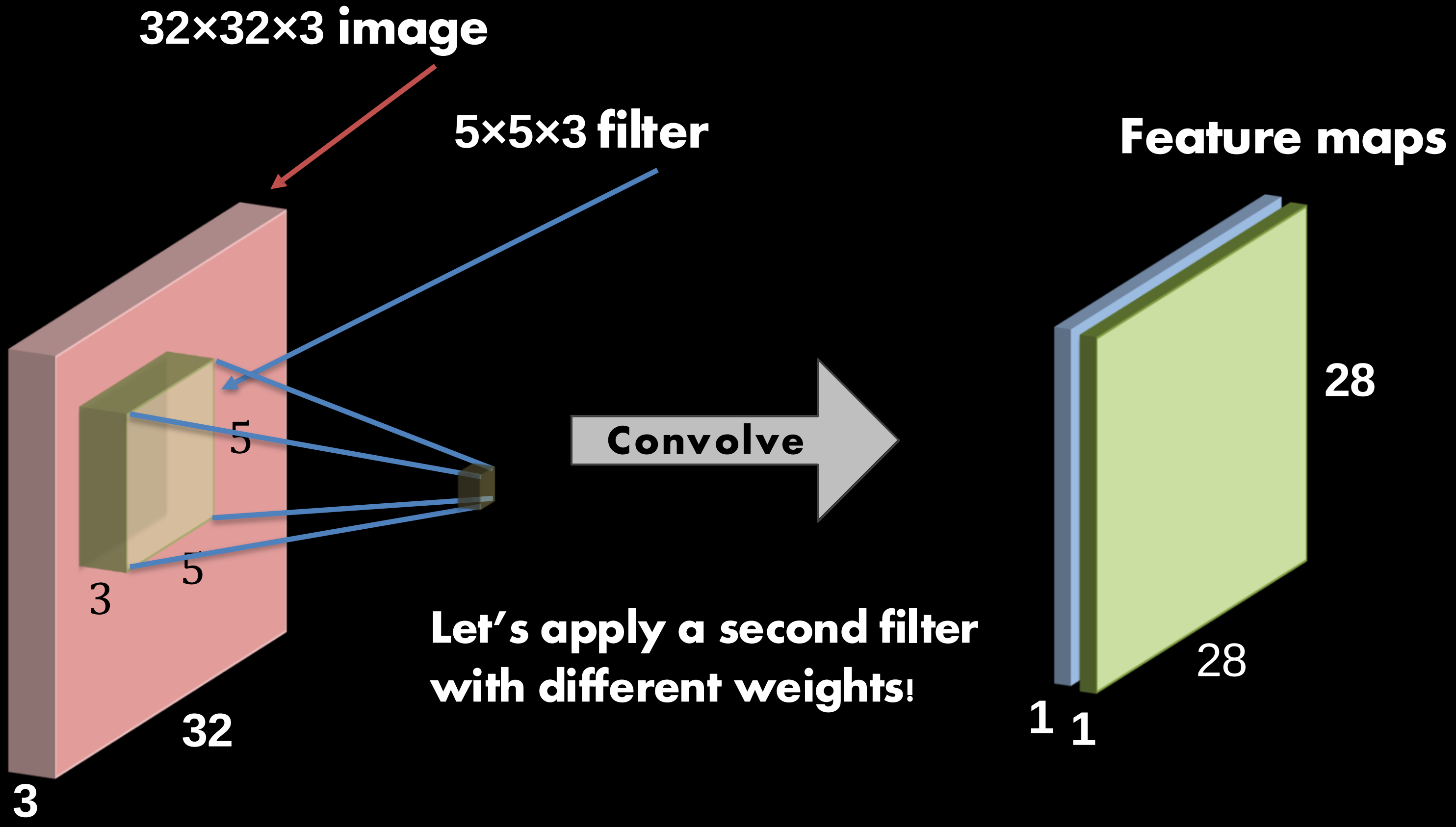
Slide from I2DL: Prof. Cremers

Convolutions on Volumetric Images



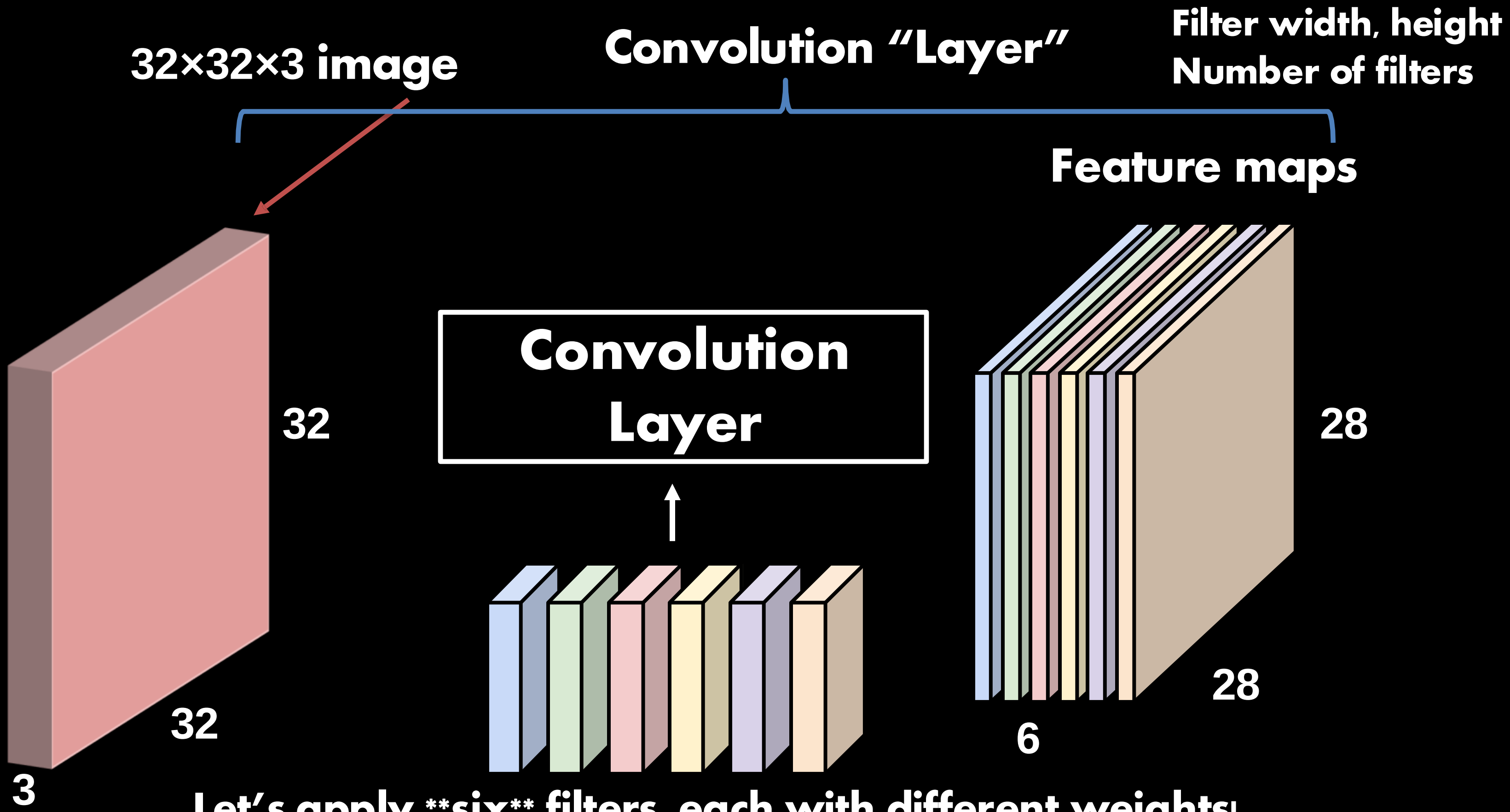
Slide over all spatial locations x_i and compute all output z_i , there are 28×28 unique locations

Convolutions on Volumetric Images



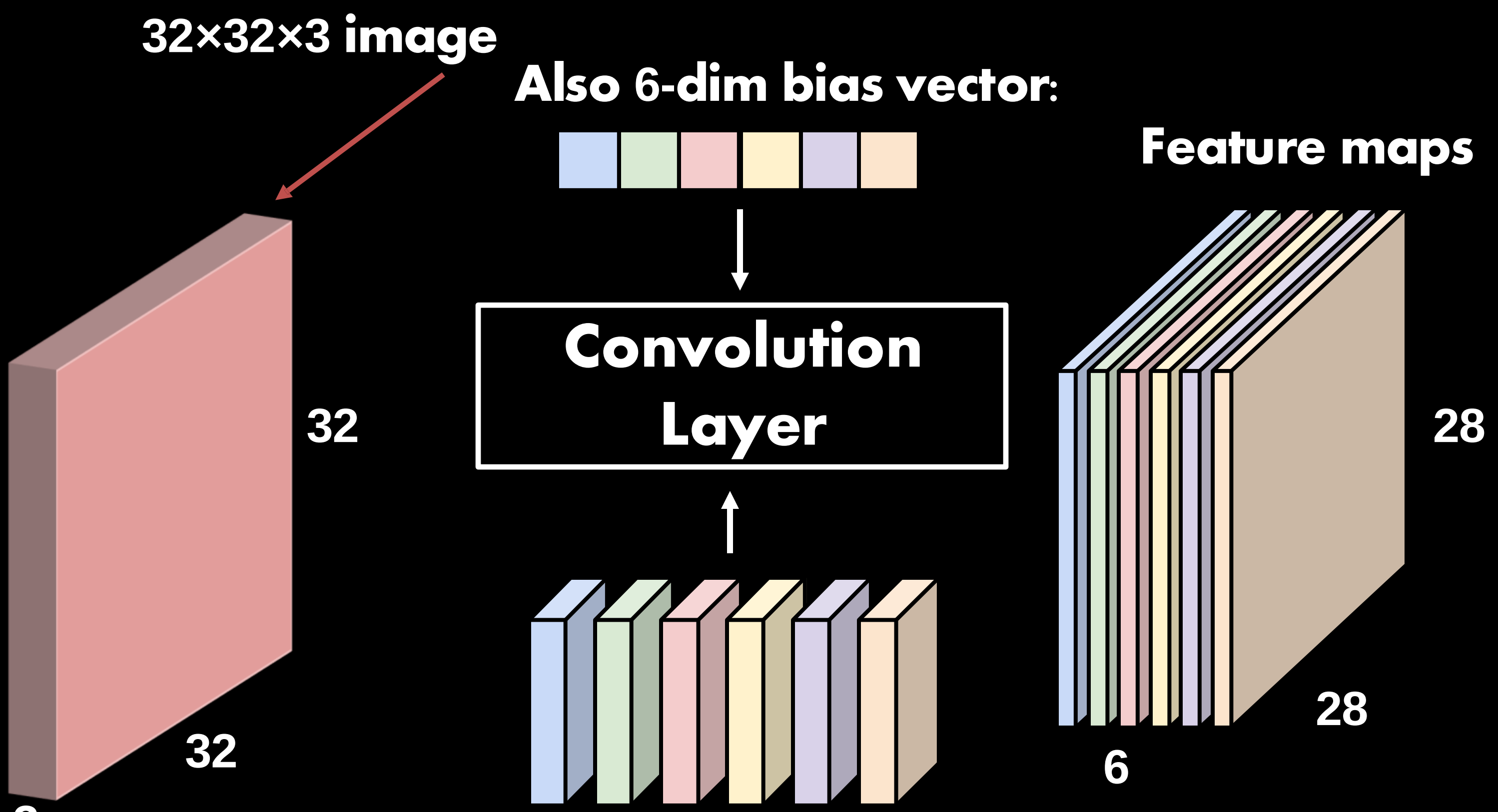
Let's apply a second filter with different weights!

Convolution Layer



Let’s apply **six** filters, each with different weights!

Convolution Layer

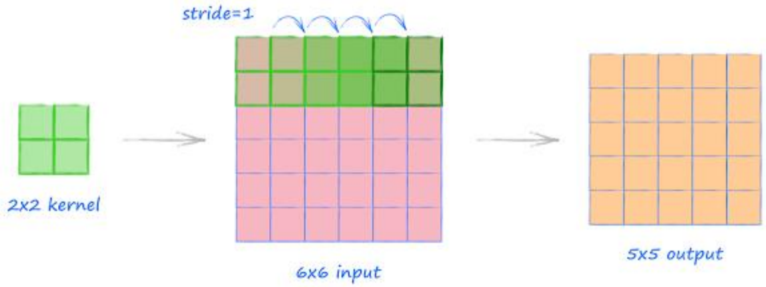


Let's apply **six** filters, each with different weights!

Stride

How far to move filter (kernel) between applications

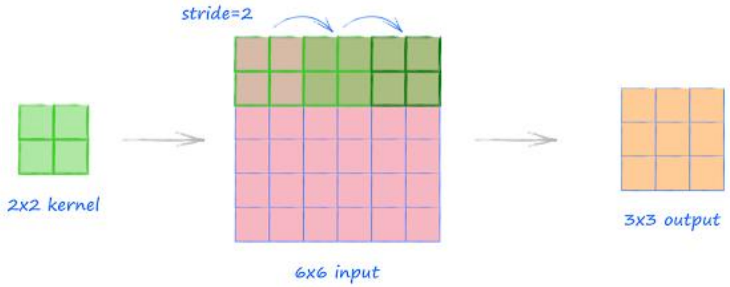
Increasing stride downsamples the image



Stride

How far to move filter (kernel) between applications

Increasing stride downsamples the image



Input: $N \times N$

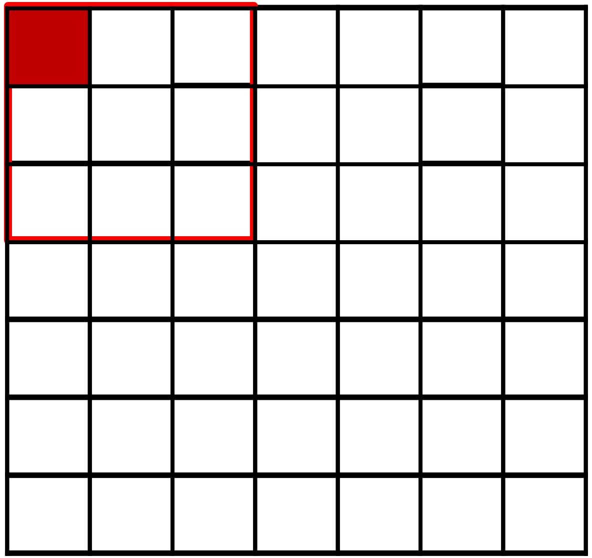
Filter: $F \times F$

Stride: S

Output: $\left(\frac{N-F}{S}+1\right)\left(\frac{N-F}{S}+1\right)$

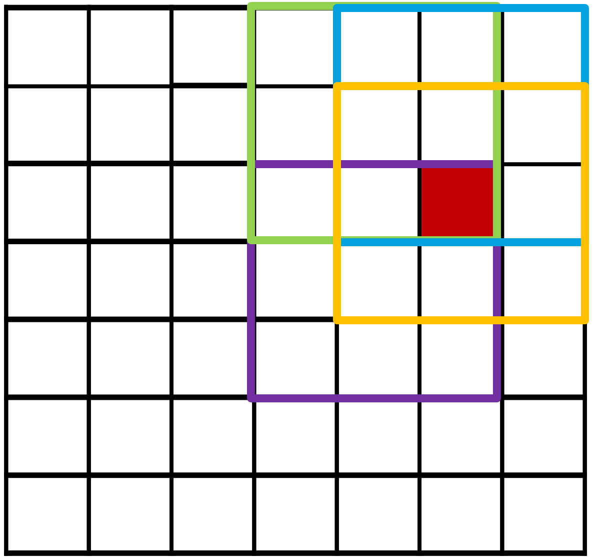
Padding

Convolutions have problems on edges



Padding

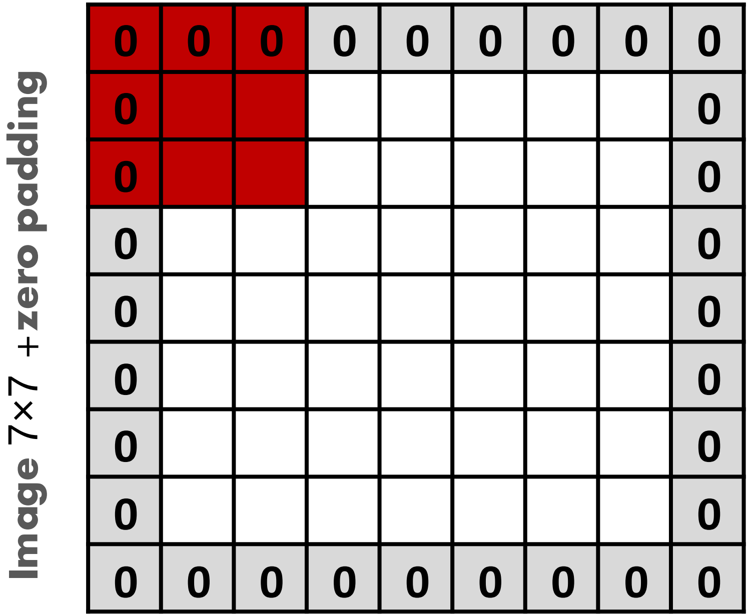
Convolutions have problems on edges



Padding

Convolutions have problems on edges

Pad: add extra pixels on images

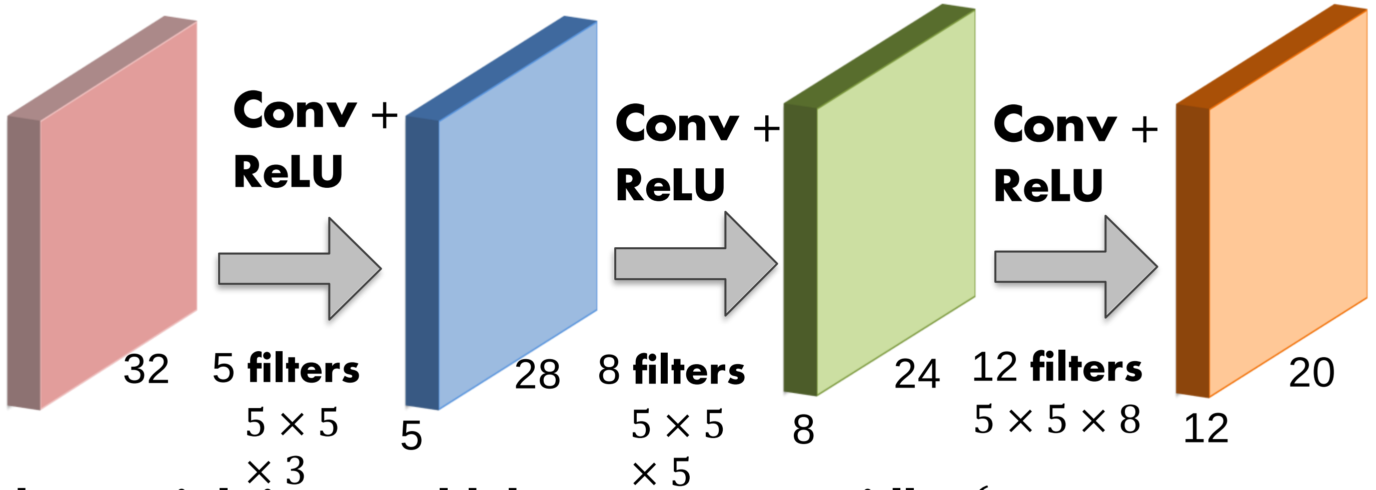


Padding

Convolutions have problems on edges

Pad: add extra pixels on images

Padding maintains feature map dimensions after convolution



The spatial size would decrease too rapidly $(32 \rightarrow 28 \rightarrow 24 \rightarrow 20)$

Pooling Layer

Processing: pool values over a region of the feature map

Output: a reduced version of the feature map by a factor of the stride

Pooling types: Max, Average

Most common: 2×2 maxpooling, stride of 2

Input feature map (single slice)

7	3	5	2
8	7	1	6
4	9	3	9
0	8	4	5

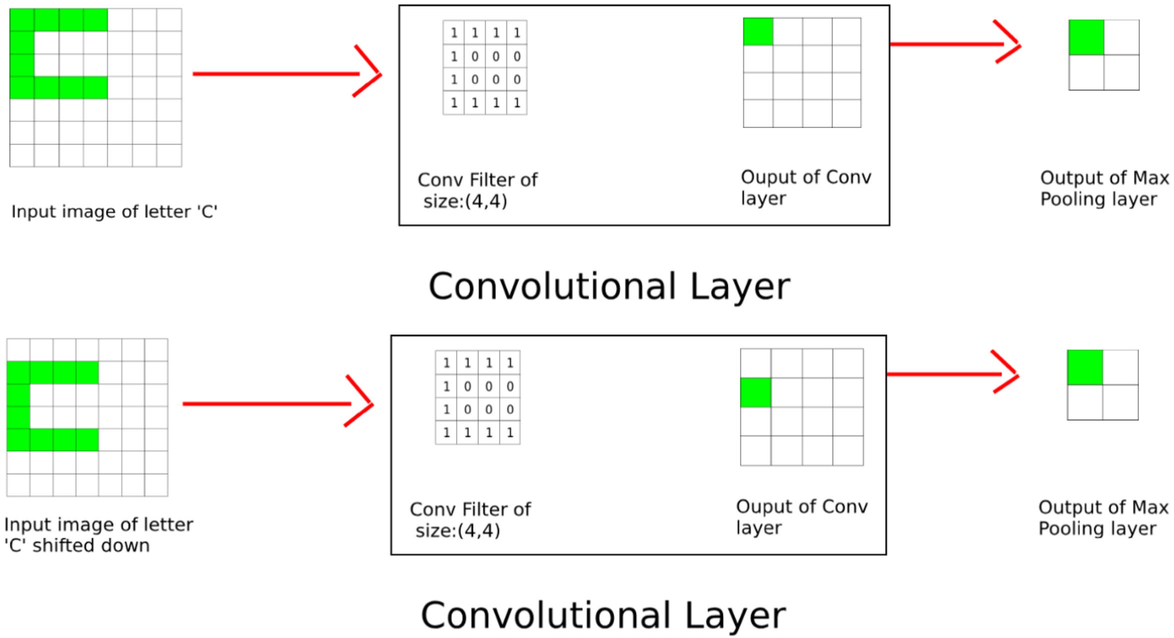
2×2 maxpooling and stride of 2

'Pooled' output

8	6
9	9

Pooling Layer

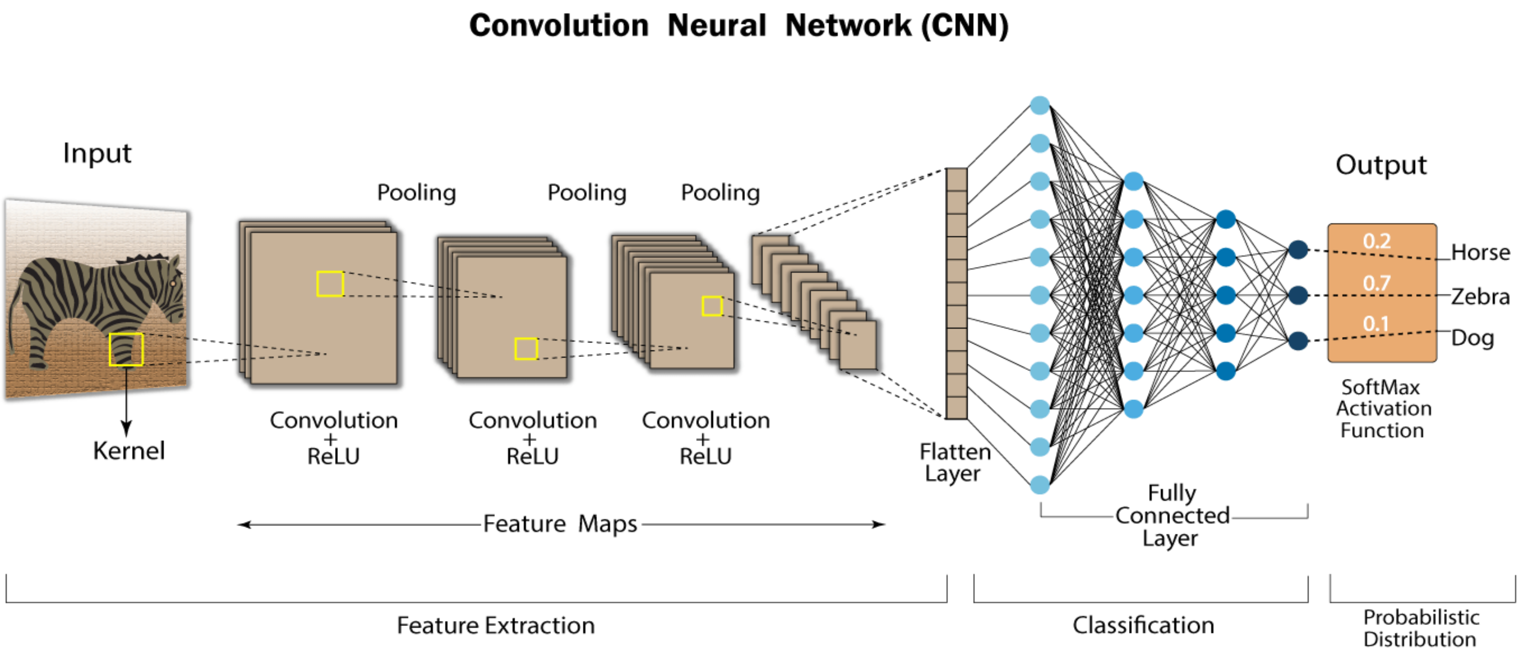
Introduces (small) translation invariance



CNN Prototype for Image Classification

Feature Extractor : Convolution+ ReLU activations+ Pooling (repeated)

Classification Head : Flattening→ FC Layers→Output Classification

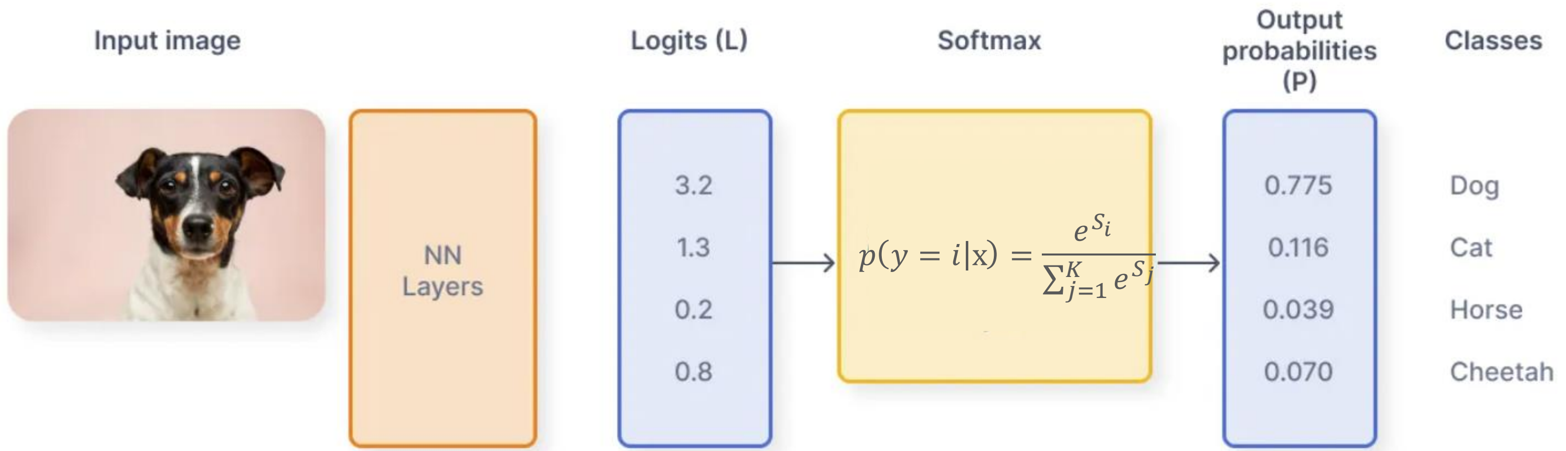


Softmax: Multi-Class Classification

Softmax: the normalized exponential function of all scores (logits)

- x represents the input features (final layer)
- S_i is unnormalized score of class i (final layer)

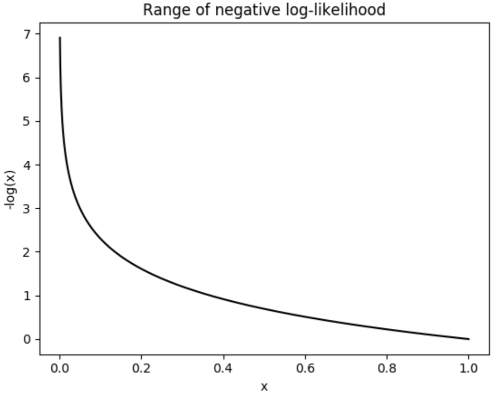
$$p(y = i|\mathrm{x}) = \frac{e^{S_i}}{\sum_{j=1}^{K} e^{S_j}}$$

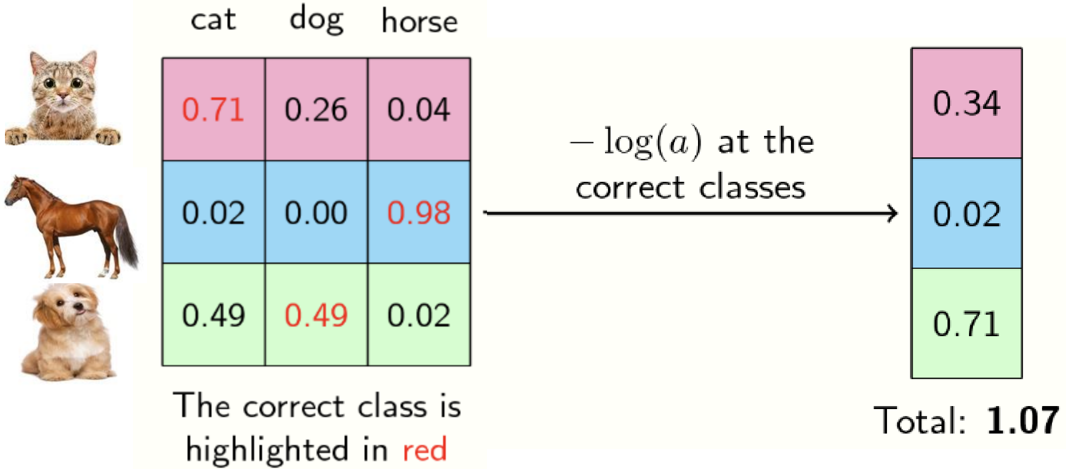


Cross-Entropy Loss for Multi-Class Classification

- y_i is the one-hot encoded label for class i
- p_i is the predicted probability of class i

$$\mathcal{L} = -\sum_{i=1}^{K} y_i \log(p_i)$$





Cross-Entropy Loss for Multi-Class Classification

- y_i **is the one-hot encoded label for class** i
- p_i **is the predicted probability of class** i

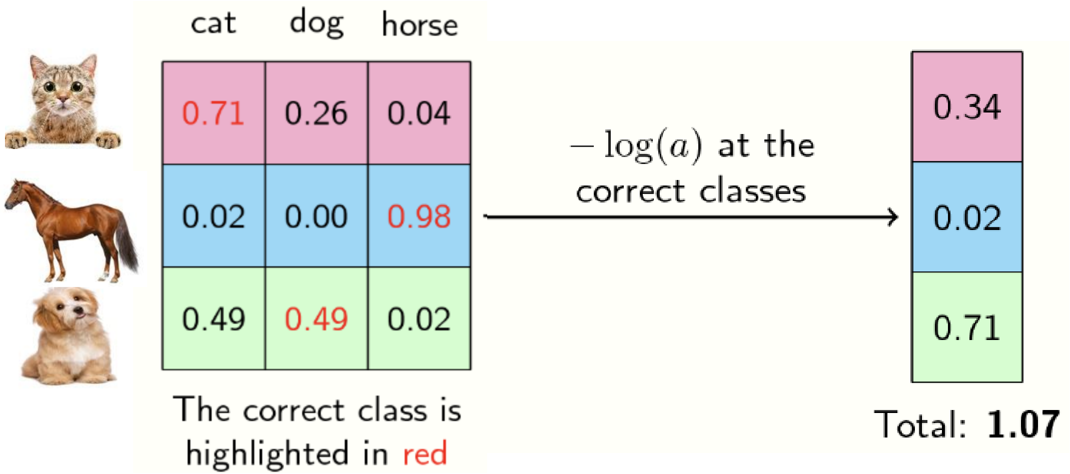
$$\mathcal{L} = -\sum_{i=1}^{K} y_i \log(p_i)$$

The gradient of loss w.r.t. logit

$$\frac{\partial \mathcal{L}}{\partial s_i} = p_i - y_i$$

For the correct class j $(y_j = 1)$

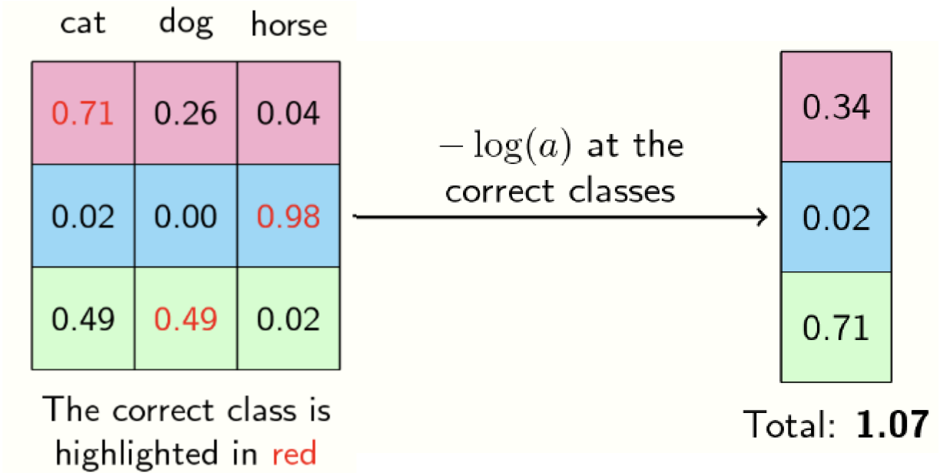
$$\frac{\partial \mathcal{L}}{\partial s_j} = p_j - 1$$



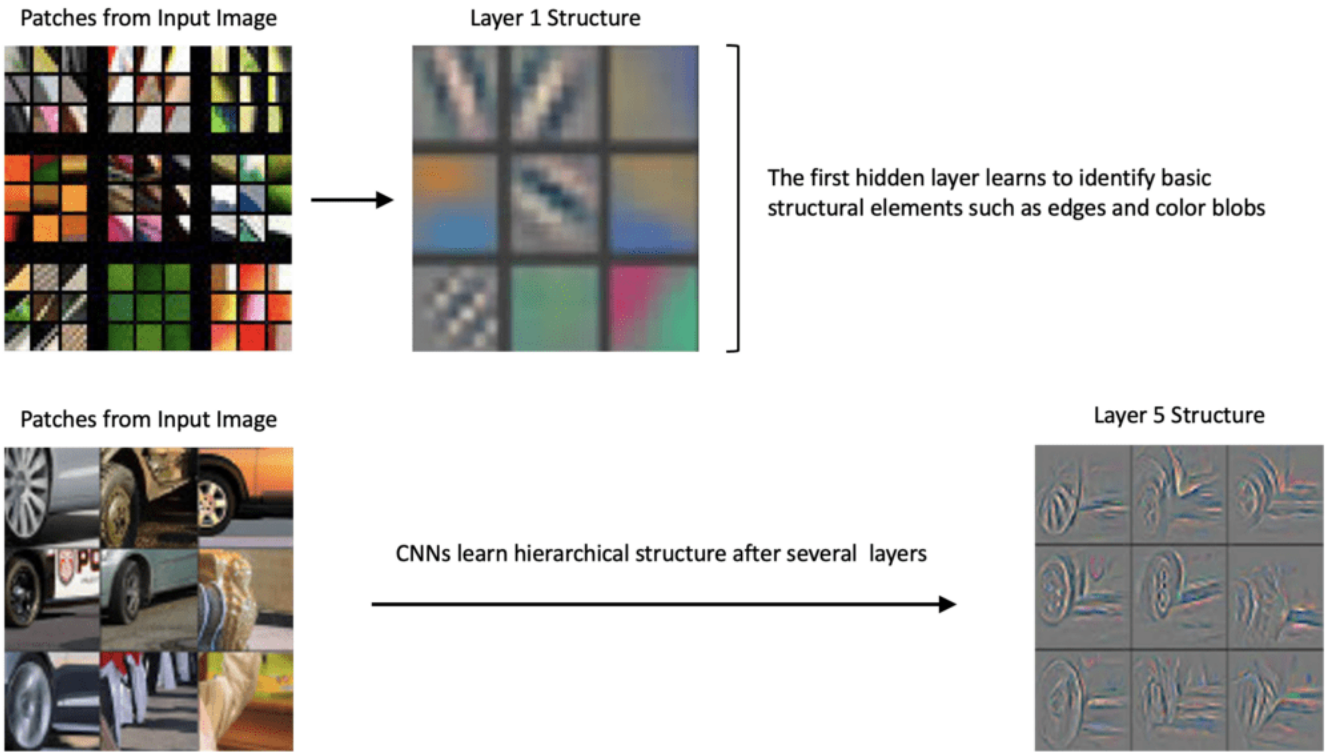
Cross-Entropy Loss for Multi-Class Classification

- $y_{n,i}$ is the actual label for the n-th sample for class i
- $p_{n,i}$ is the predicted probability for the n-th sample of class i
- N is the number of samples in a batch

$$\mathcal{L}_{total} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{K} y_{n,i}\log(p_{n,i})$$



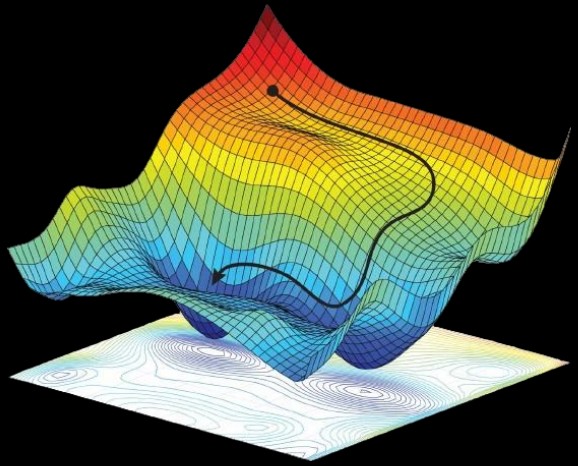
CNN Learns Hierarchical Features



2014

Visualizing and Understanding Convolutional Networks

Optimizing Neural Networks



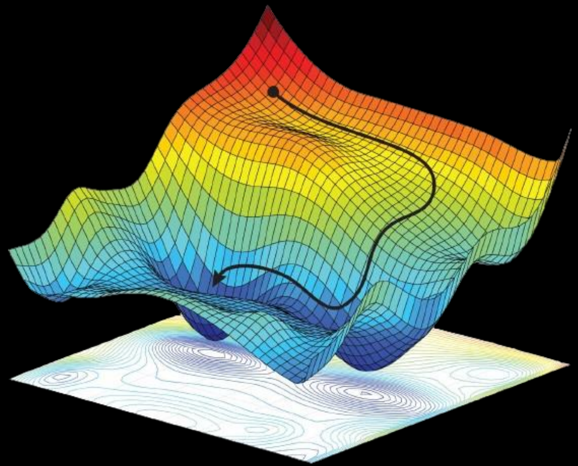
Loss Optimization

Finding network's parameters (weights) that achieve the lowest loss

$$W^* = \underset{W}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

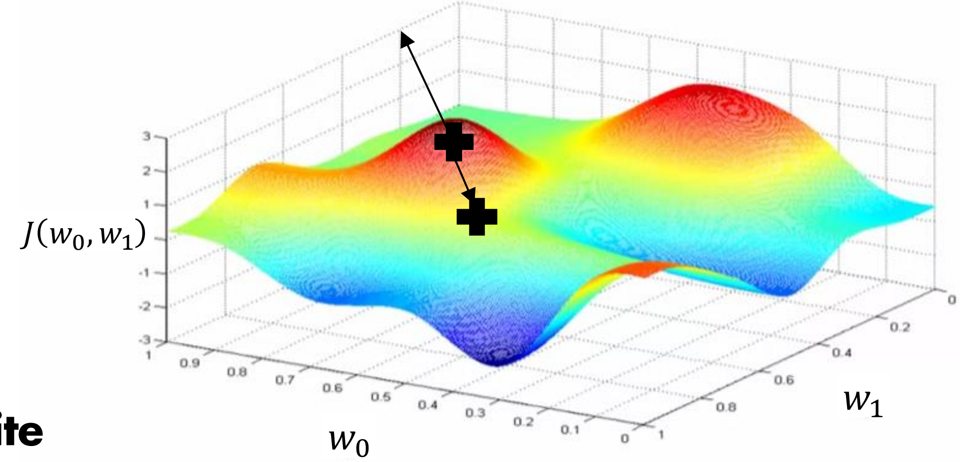
$$W^* = \underset{W}{\arg\min}\, J(W)$$

$$W = \left\{W^{(0)}, W^{(1)}, \dots\right\}$$



Loss Optimization

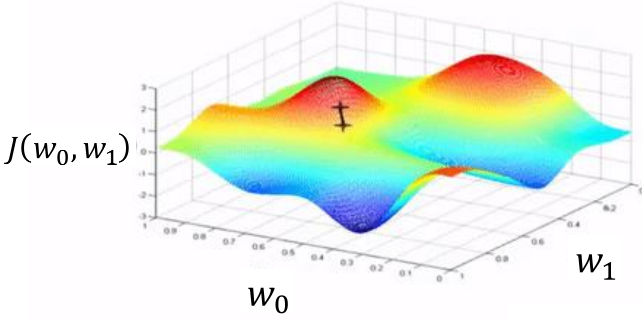
- Randomly pick a point (w_0, w_1)
- Compute gradient, $\frac{\partial J(W)}{\partial W}$
- Take a small step in the opposite direction of the gradient
- Repeat this process until convergence



Mini-Batch Gradient Descent

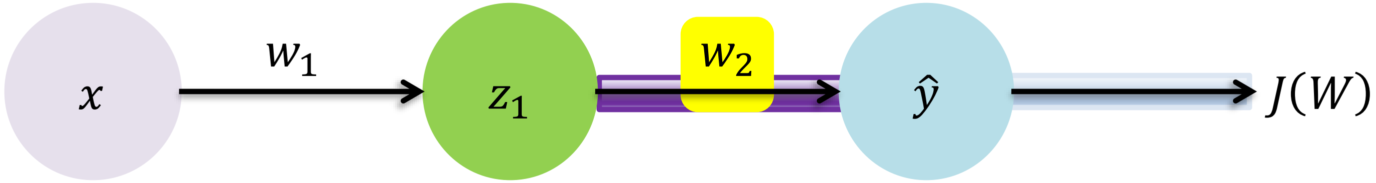
- **Algorithm:**
  - **Initialize weights randomly** $\sim\mathcal{N}(0,\sigma^2)$
  - **Loop until convergence:**
  - **Pick a mini-batch of** B **data samples**
  - **Compute gradient,** $\frac{\partial J(W)}{\partial W} = \frac{1}{B}\sum_{k=1}^{B}\frac{\partial J_k(W)}{\partial W}$
  - **Update weights,** $W \leftarrow W - \alpha\frac{\partial J(W)}{\partial W}$
  - **Return weights**

Use a suitable method (e.g., Xavier or He initialization) to ensure stable variance of activations and gradients.



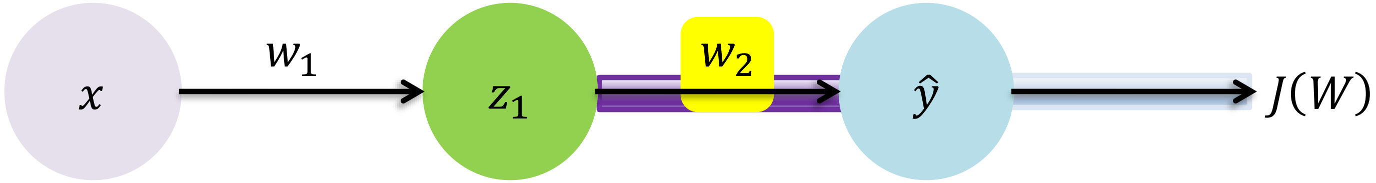
Better estimation of true gradient and fast to compute, smoother convergence

Backpropagation: Chain Rule in Action



$$\frac{\partial J(W)}{\partial w_2} =$$

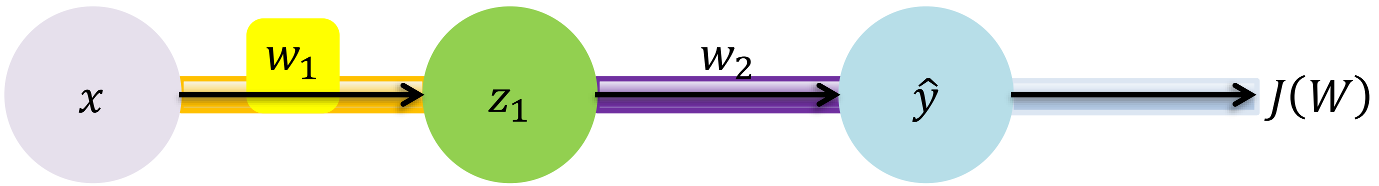
Backpropagation: Chain Rule in Action



$$\frac{\partial J(W)}{\partial \boxed{w_2}} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}$$

Let's apply chain rule !

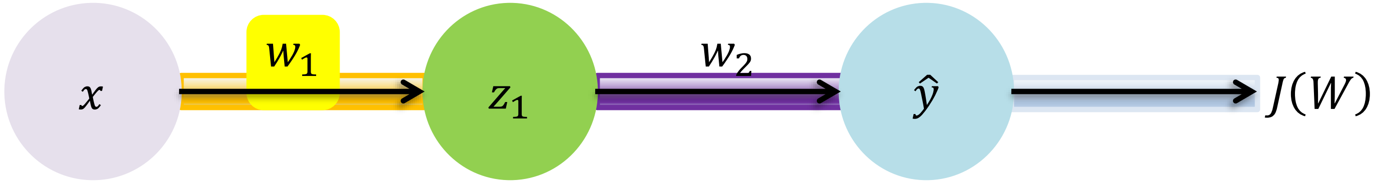
Backpropagation: Chain Rule in Action



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule

Backpropagation: Chain Rule in Action



$$\frac{\partial J(W)}{\partial w_1} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}} * \underbrace{\frac{\partial \hat{y}}{\partial z_1}} * \underbrace{\frac{\partial z_1}{\partial w_1}}$$

Gradient Dynamics in Deep Network Training

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial h_n} * \frac{\partial h_n}{\partial h_{n-1}} * \frac{\partial h_{n-1}}{\partial h_{n-2}} * \cdots * \frac{\partial h_2}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

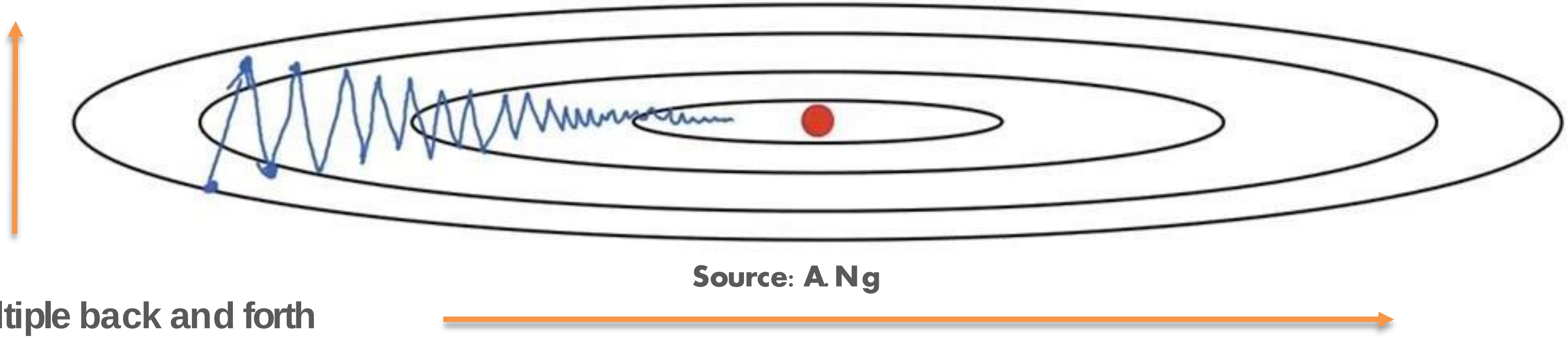
In most cases, there are two possible solutions:

- **We get zero if** $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| < 1 \ \rightarrow \ \prod_{i=2}^{n} \frac{\partial h_i}{\partial h_{i-1}} \ \cdots \ \text{Vanish!}$
- **We get infinity if** $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| > 1 \ \rightarrow \ \prod_{i=2}^{n} \frac{\partial h_i}{\partial h_{i-1}} \ \cdots \ \text{Explode!}$
- **We only get a reasonable answer if the numbers are all close to one**

Limitations of Gradient Descent + Alternatives

Challenges with Vanilla Gradient Descent:

- **Oscillations due to anisotropic curvature of the loss surface**
- **Slow convergence**



Source: A. Ng

We take multiple back and forth steps in this direction.

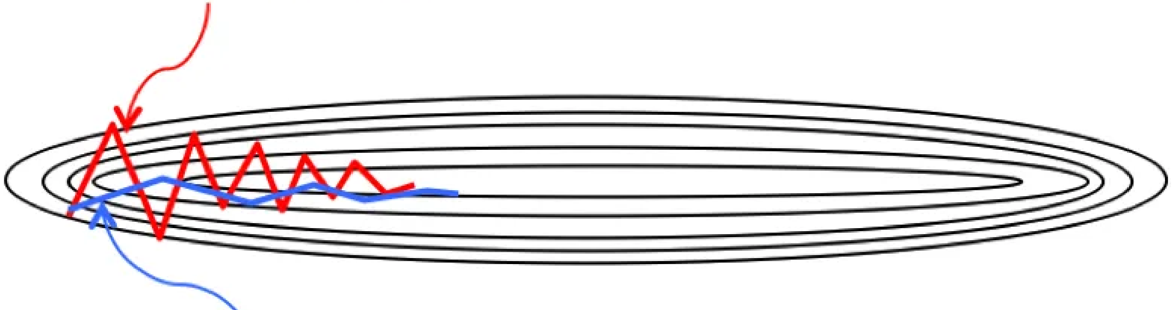
We'd ideally like to move faster in this direction

Limitations of Gradient Descent + Alternatives

Gradient Descent with **Momentum**:

- **Smoother updates, dampens oscillations**
- **Speeds up convergence**

$$v_t = \gamma v_{t-1} + \alpha \nabla J(W_t)$$
$$W_{t+1} = W_t - v_t$$



[Sutskever et al., ICML'13] On the importance of initialization and momentum in deep learning

Limitations of Gradient Descent + Alternatives

Adam (Adaptive Moment Estimation) Optimizer:

- **Combines momentum (first moment of gradients) with adaptive learning rates based on the second moment (squared gradients)**
- **Popular in deep learning due to its robustness**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla J(W_t) \qquad \text{(First moment: gradient mean)}$$

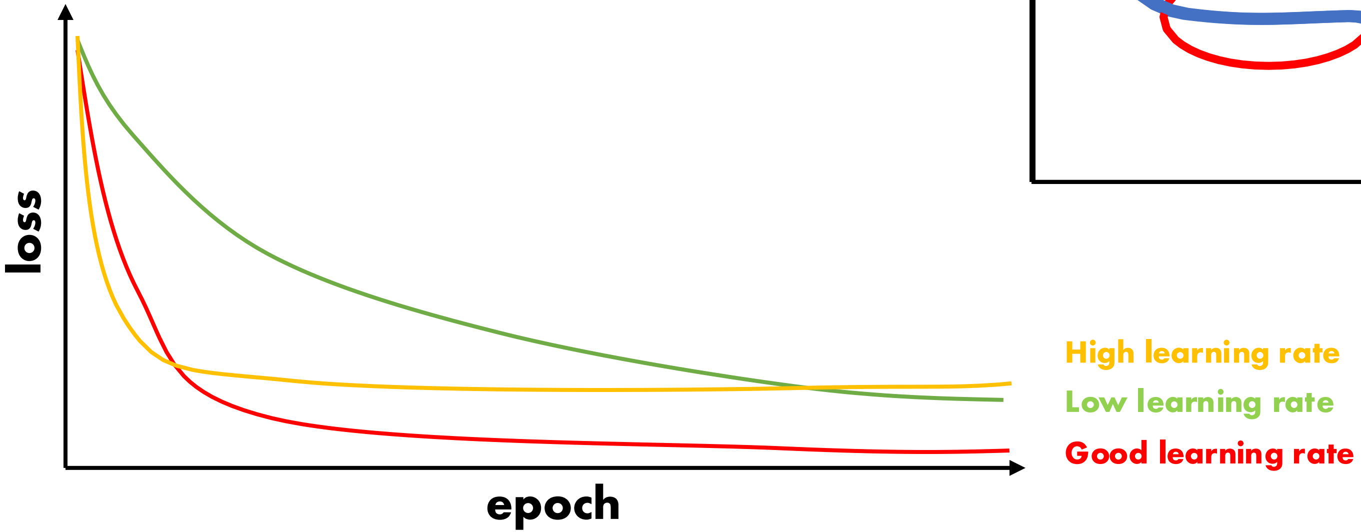
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla J(W_t))^2 \qquad \text{(Second moment: gradient variance)}$$

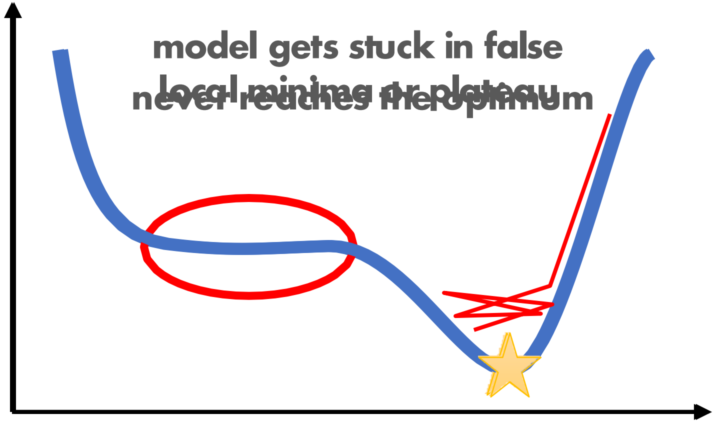
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \qquad \text{(Bias-corrected)}$$

$$W_{t+1} = W_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \qquad \beta_1 \approx 0.9, \beta_2 \approx 0.999, \epsilon \approx 10^{-8}.$$

[Kingma et al., ICLR'15] Adam: A method for stochastic optimization

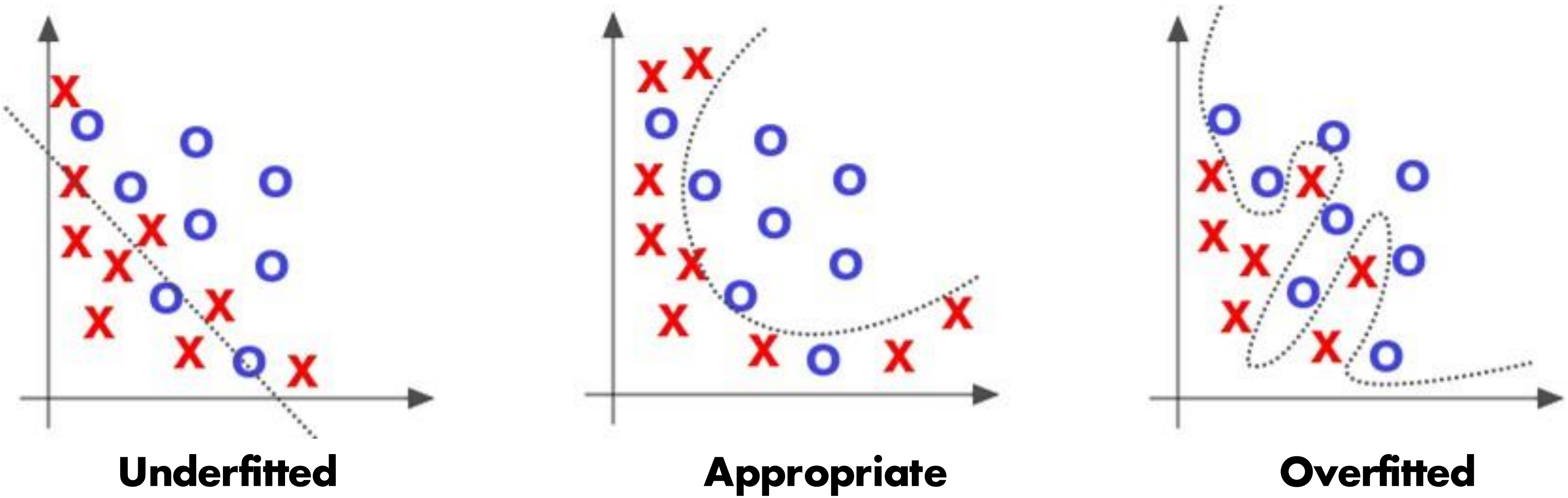
Learning Rate Tuning





Regularization &
Data Augmentation

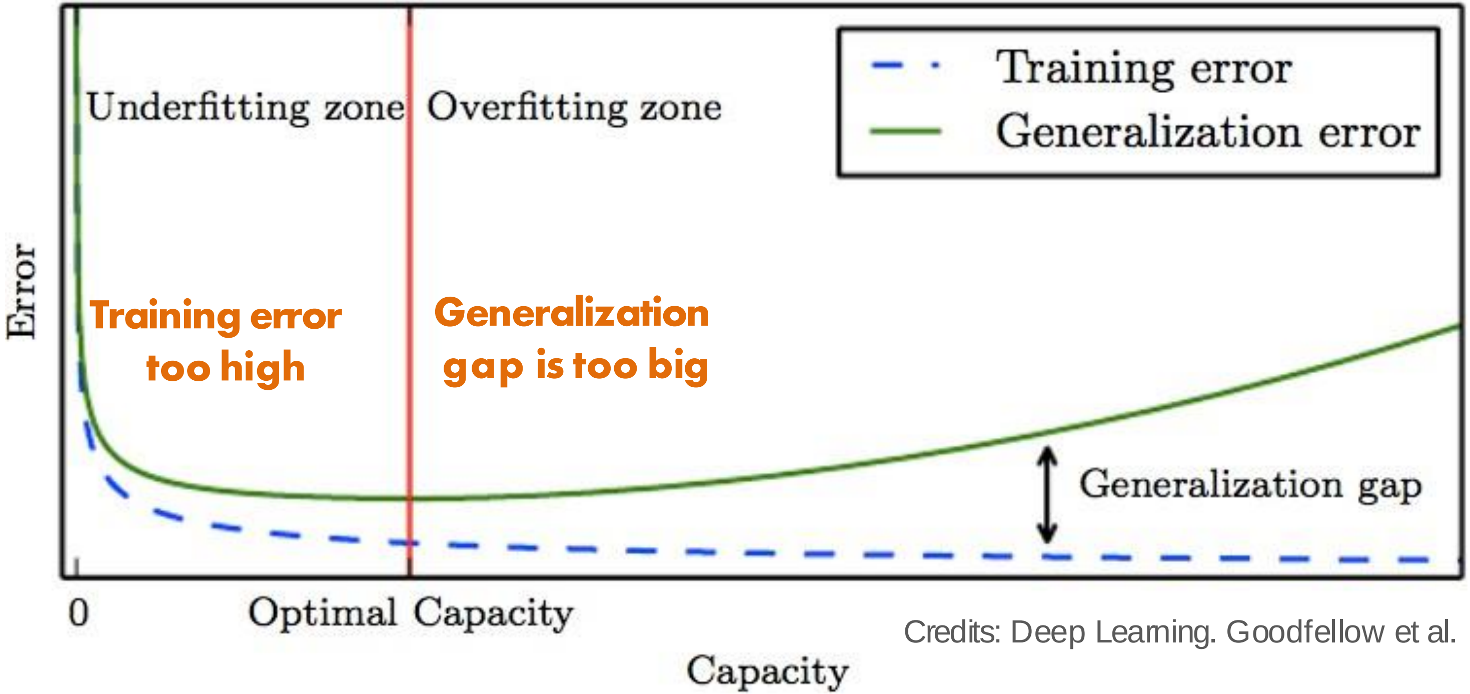
Over-and Underfitting



Source: Deep Learning by Adam Gibson, Josh Patterson, O'Reily Media Inc., 2017

Over-and Underfitting

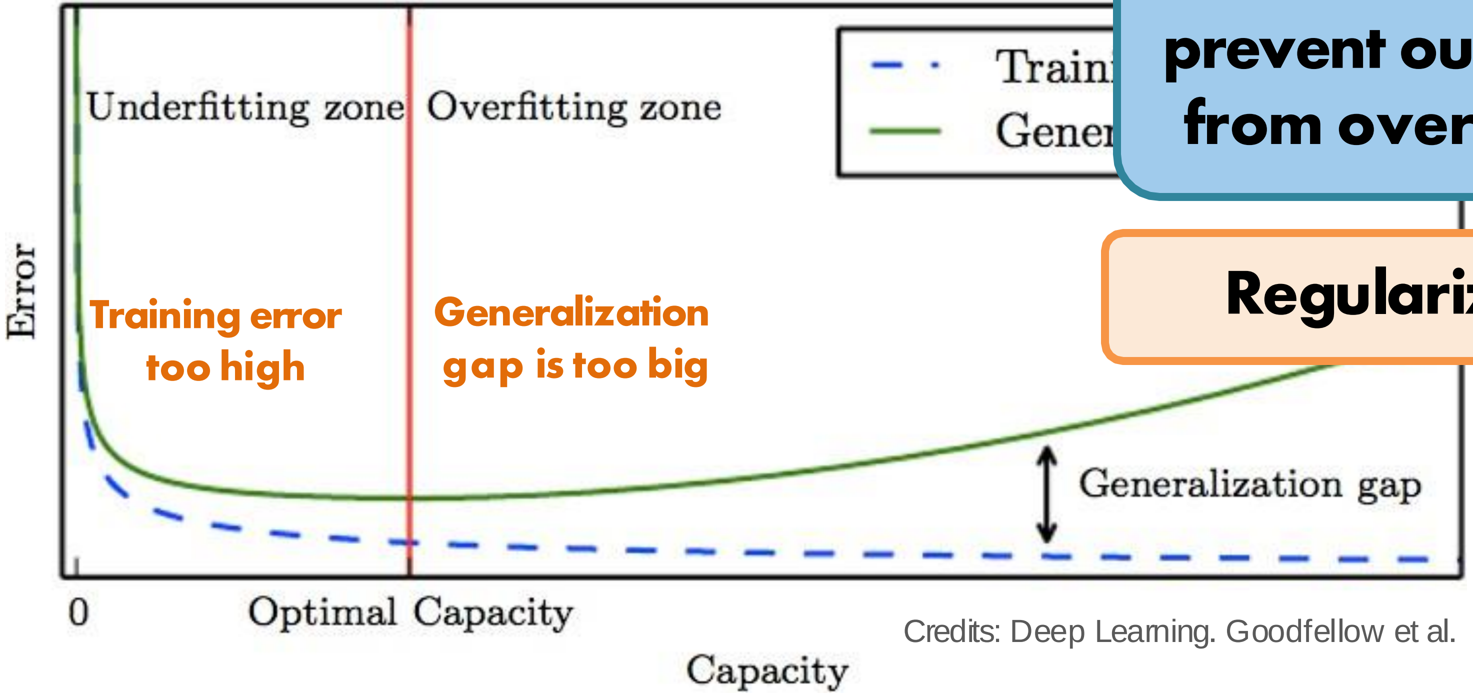
Training/ Validation curve



Credits: Deep Learning. Goodfellow et al.

Over-and Underfitting

Training/ Validation curve



How can we prevent our model from overfitting?

Regularization

Credits: Deep Learning. Goodfellow et al.

Regularization

- **Loss function** $\mathcal{L}(y, \hat{y}, w) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 + \boxed{\lambda R(w)}$

- **Regularization techniques**
  - $\left.\begin{array}{l}\textbf{L}_2\textbf{ regularization}\\ \textbf{L}_1\textbf{ regularization}\end{array}\right\}$ **Add regularization term to loss function**
  - **Dropout**
  - **Early stopping**
  - …

Regularization Example

- **Input** : 3 **features** $x = [1, 2, 1]$
- **Two linear classifiers that give the same result:**

- $w_1 = [0, 0.9, 0]$ → **Ignores 2 features**
- $w_2 = [0.15, 0.75, 0.15]$ → **Use all features**

Regularization Example (L_2)

- **Loss function** $\mathcal{L}(y, \hat{y}, w) = \sum_{i=1}^{n} \left(x_i w_{ji} - y_i\right)^2 + \boxed{\lambda R(w)}$
- **L_2 regularization** $R(w) = \|w\|_2^2 = \sum_{i=1}^{n} w_i^2$

$$R(w_1) = 0 + 0.9^2 + 0 = 0.81$$

$$R(w_2) = 0.15^2 + 0.75^2 + 0.15^2 = \boxed{0.6075}$$

Minimization Promotes weight uniformity

$$x = [1, 2, 1], w_1 = [0, 0.9, 0], w_2 = [0.15, 0.75, 0.15]$$

Regularization Example (L1)

- **Loss function** $\mathcal{L}(y, \hat{y}, w) = \sum_{i=1}^{n} \left(x_i w_{ji} - y_i\right)^2 + \boxed{\lambda R(w)}$

- **L1 regularization** $R(w) = \|w\|_1 = \sum_{i=1}^{n} |w_i|$

$$R(w_1) = 0 + 0.9 + 0 = \boxed{0.9}$$

Minimization enforces sparsity

$$R(w_2) = 0.15 + 0.75 + 0.15 = 1.05$$

$$x = [1, 2, 1], w_1 = [0, 0.9, 0], w_2 = [0.15, 0.75, 0.15]$$

Regularization: Effect (L_1)

- Dog classifier takes different inputs

Furry

Has two eyes

Has a tail

Has paws

Has two ears

L_1 regularization encourages the model to rely on only a few key features

Regularization: Effect (L_2)

- **Dog classifier takes different inputs**

Furry

Has two eyes

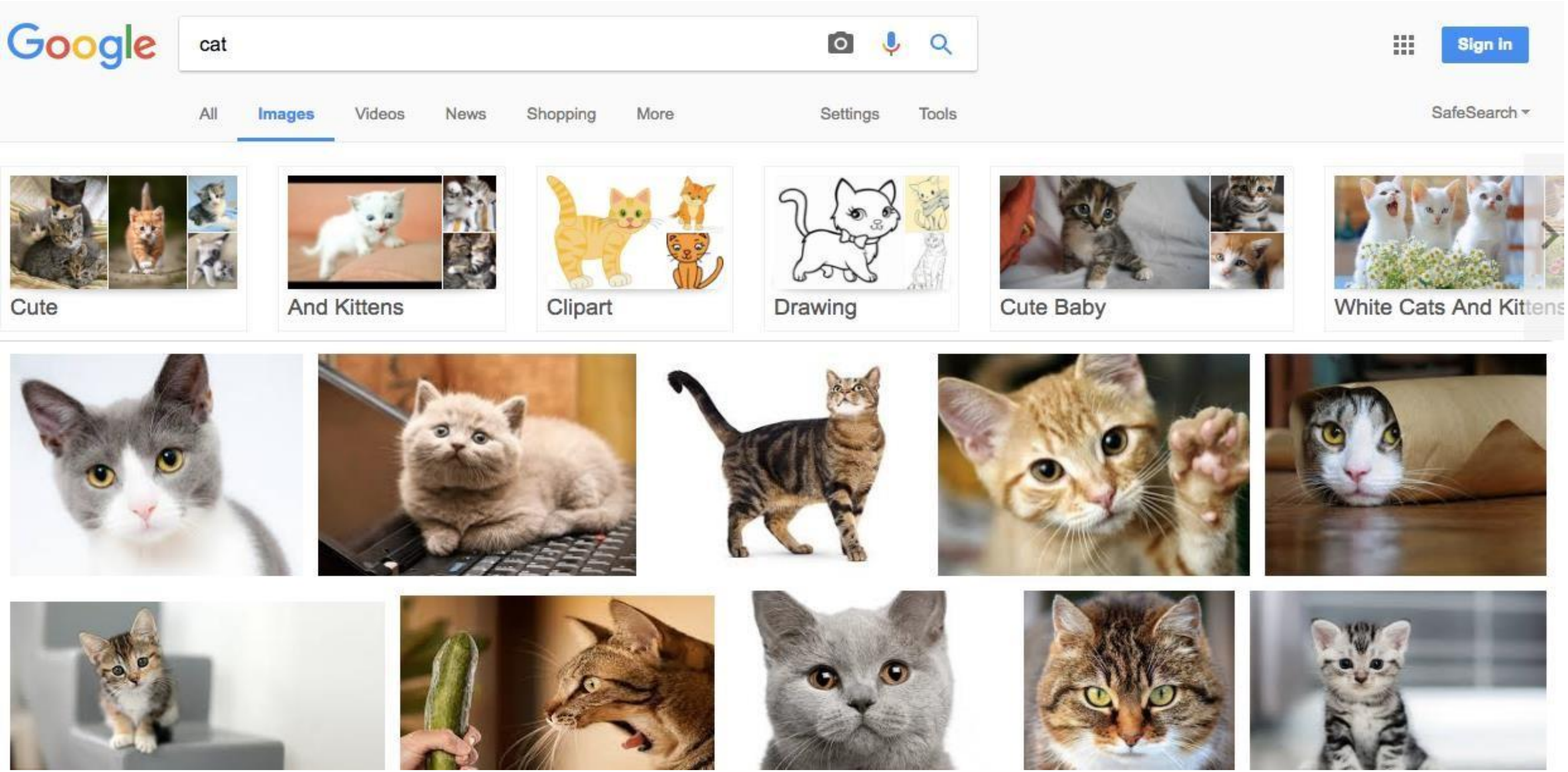
Has a tail

Has paws

Has two ears

L_2 regularization leverages all information to influence model learning

Data Augmentation: Motivation

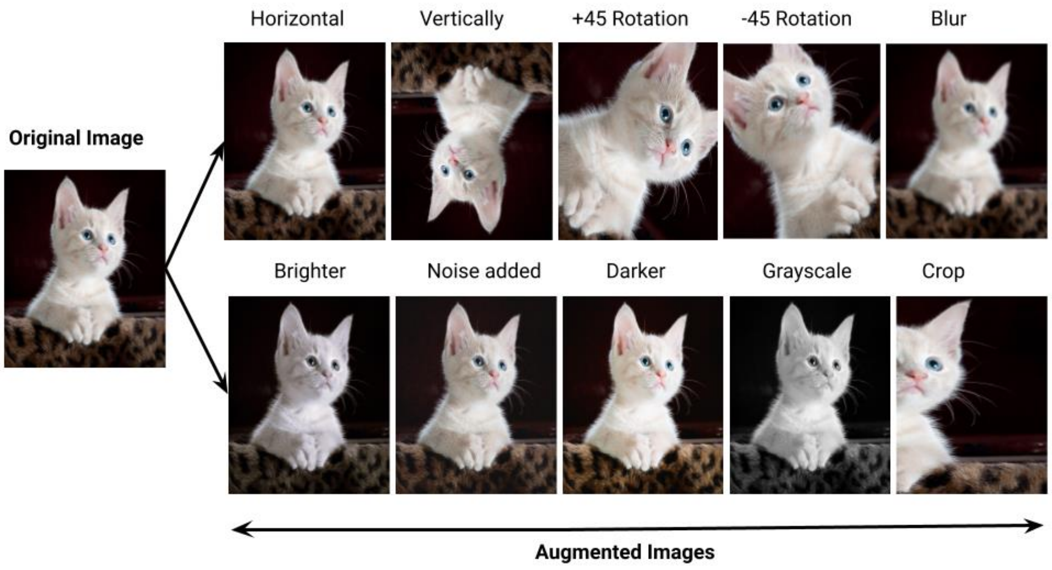


Pose **Appearance** **Illumination**

Data Augmentation

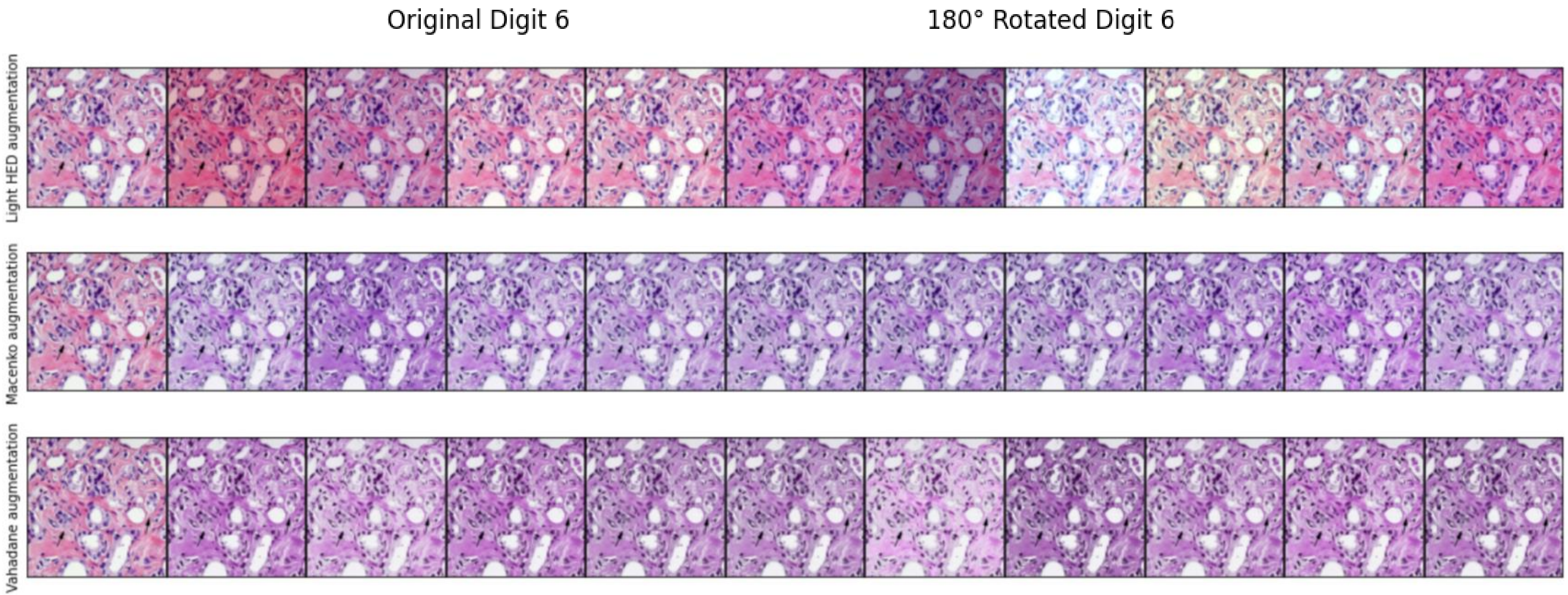
- **A classifier has to be invariant to a wide variety of transformations**
- **Augmentation: simulating plausible transformations**

📦 **Libraries:** torchvision.transforms, Kornia, Albumentations



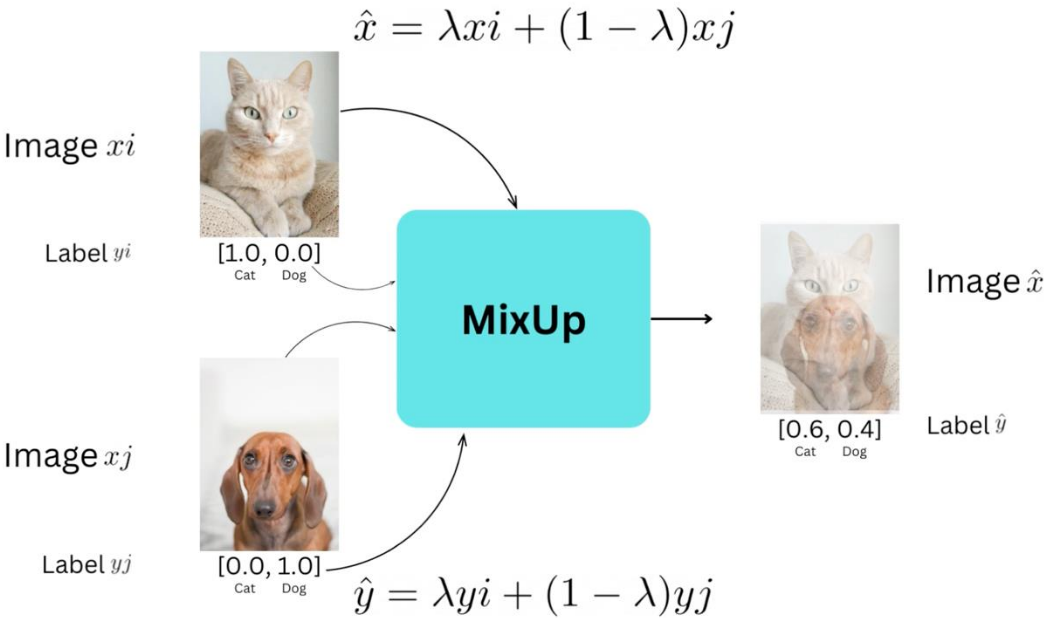
Valid & Plausible Transformations for Data Augmentation

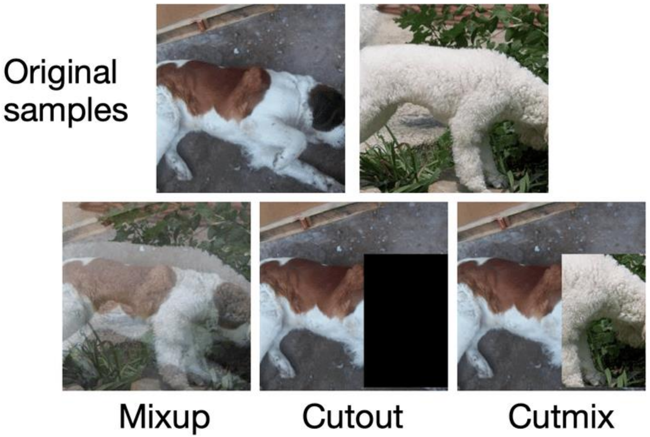
- **Any operation that does not alter the original label**



Stainlib: a Python library for augmentation of histopathology images

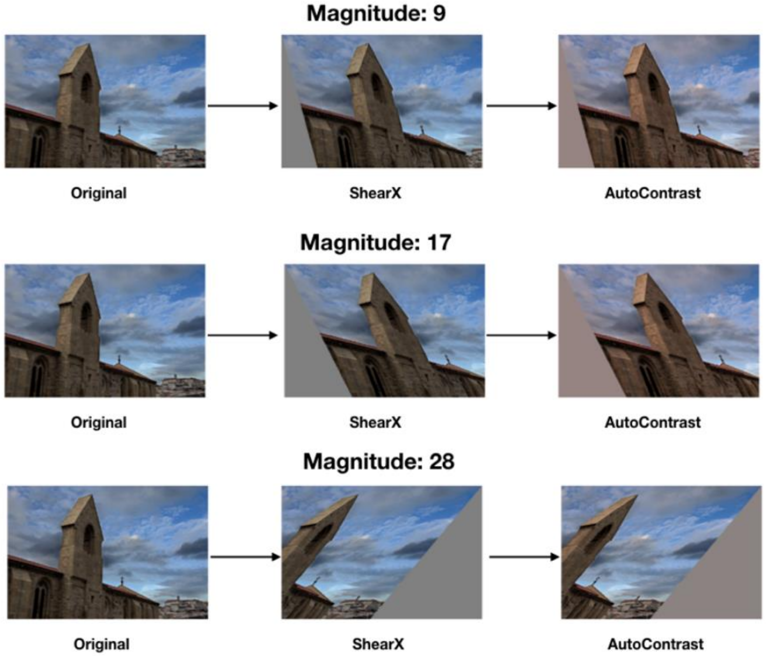
Data Augmentation: Advanced (Mixup & Variants)





Mixup: Beyond Empirical Risk Minimization

Data Augmentation: Advanced (RandAugment)




```
transforms = [
'Identity', 'AutoContrast', 'Equalize',
'Rotate', 'Solarize', 'Color', 'Posterize',
'Contrast', 'Brightness', 'Sharpness',
'ShearX', 'ShearY', 'TranslateX', 'TranslateY']

def randaugment(N, M):
"""Generate a set of distortions.

   Args:
     N: Number of augmentation transformations to
          apply sequentially.
     M: Magnitude for all the transformations.
"""

  sampled_ops = np.random.choice(transforms, N)
  return [(op, M) for op in sampled_ops]
```

Figure 2. Python code for RandAugment based on numpy.

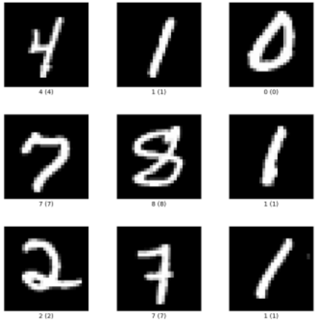
RandAugment: Practical automated data augmentation with a reduced search space

Vision Benchmarks, ResNet, BatchNorm & Transfer Learning

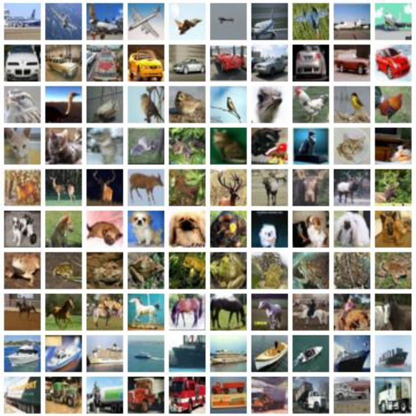
Key Datasets as Benchmarks for Image Classification

- **Example datasets:**
  - **MNIST (handwritten digits), 1990s-today: 60,000 images**
  - **CIFAR 10 & CIFAR 100, 2009: ~60,000 images**
  - **ILSVRC (ImageNet-1K), 2009: 1.2 million training images, 1000 categories**

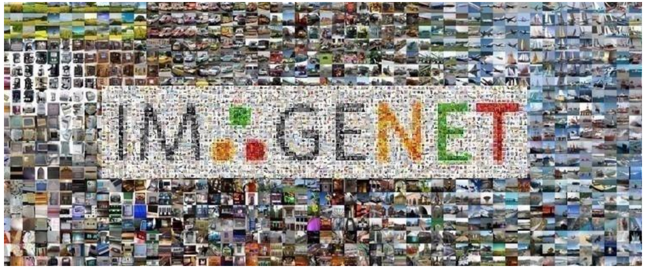
MNIST



CIFAR - 10



ImageNet- 1K

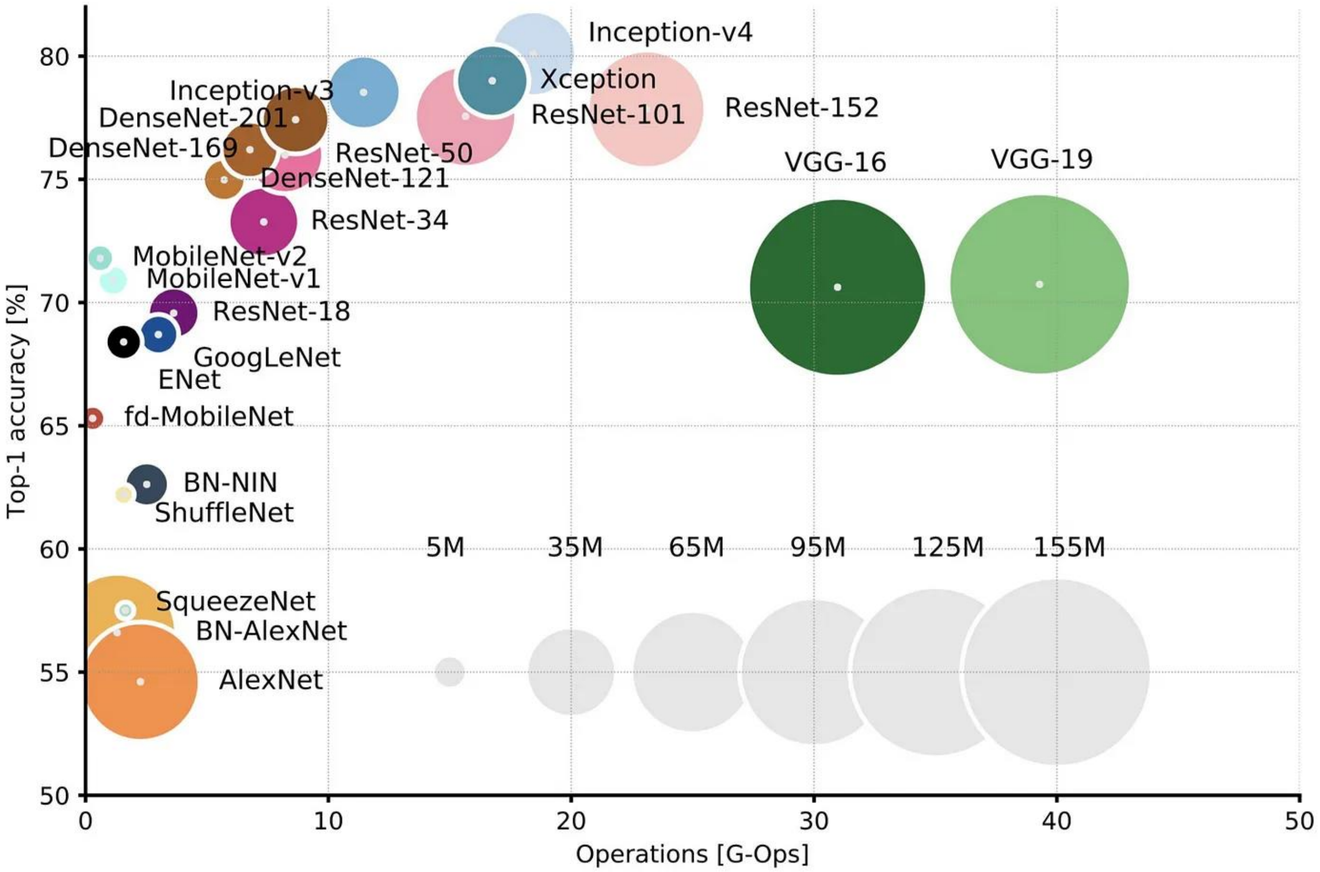


Google's JFT-300M

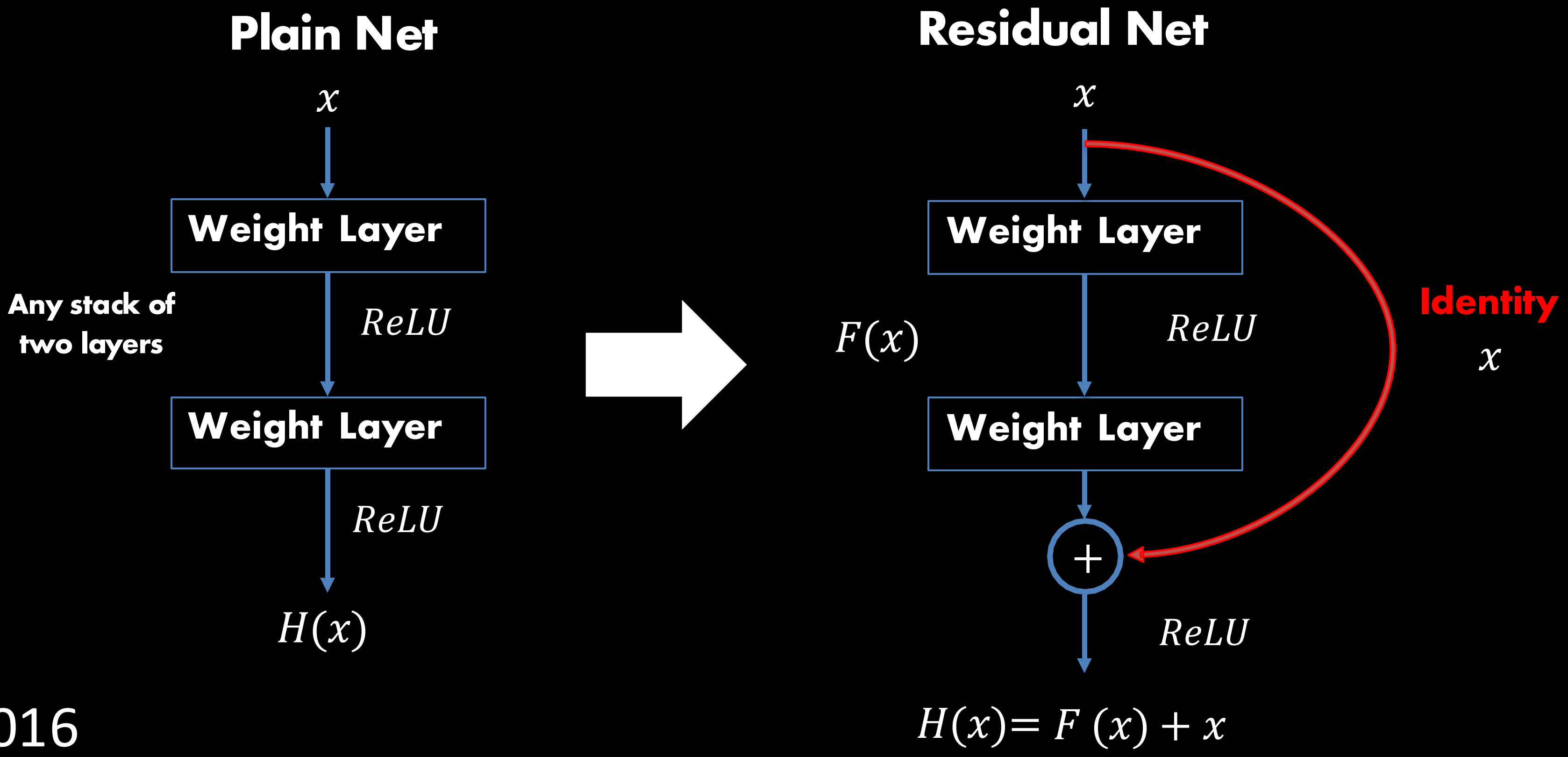
ImageNet- 21K

LAION- 400M

CNN Architectures: Accuracy vs. Complexity



Residual Networks (ResNets)

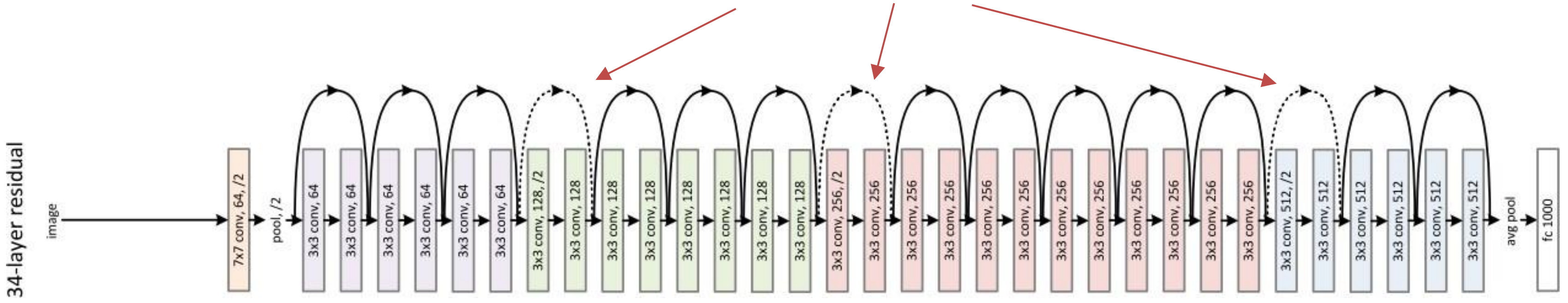


2016

Deep Residual Learning for Image Recognition

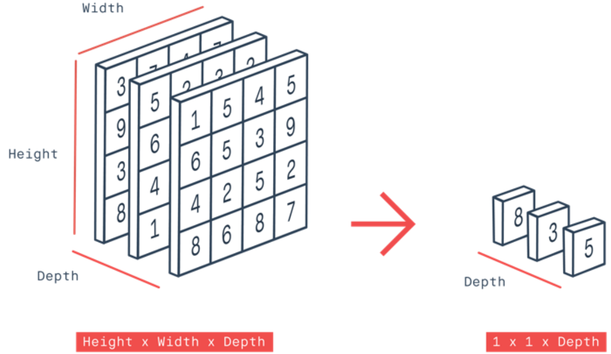
ResNets

Skip Connection



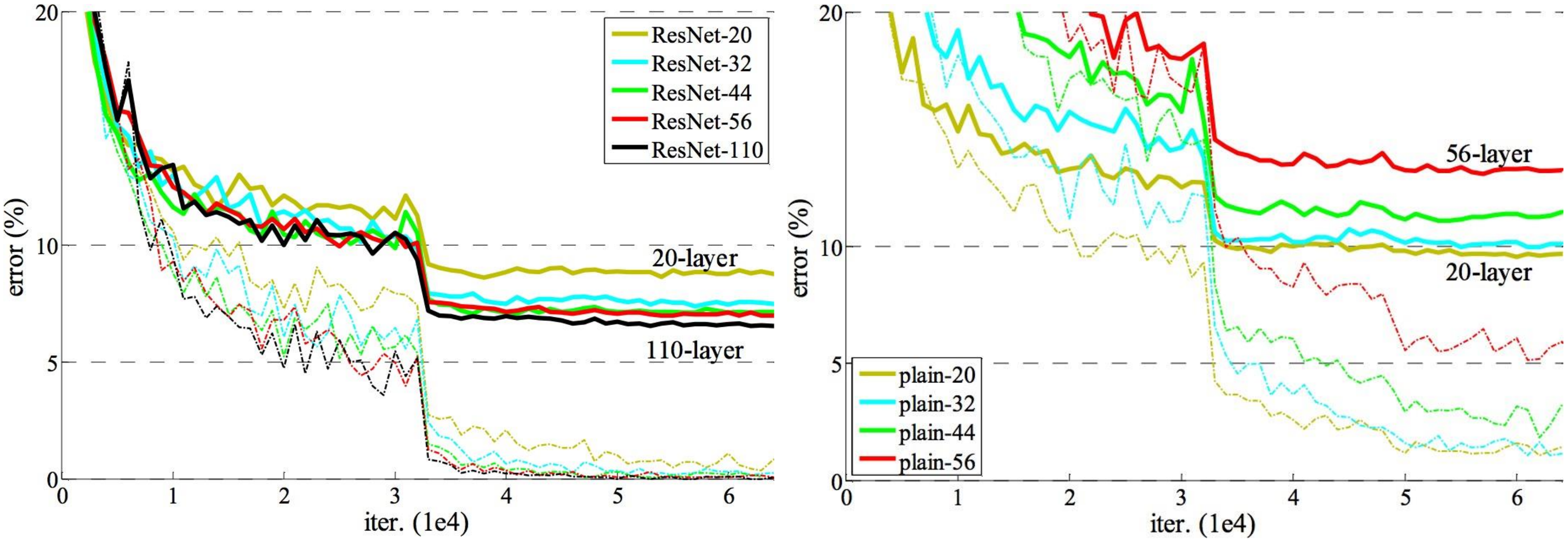
Why is this model important?

- **Frequently used today**
- **Skip connections and use of batch normalization**
- **Use of global average pooling instead of FC layers**



Depth vs. Performance

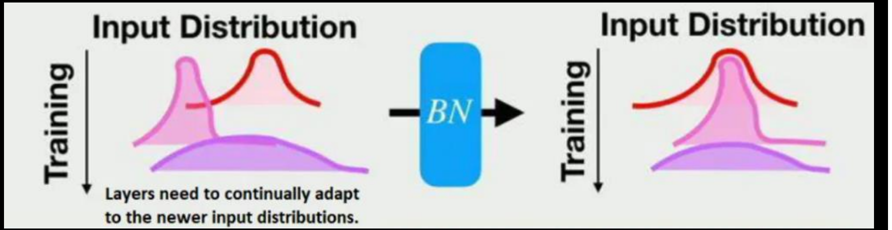
dashed: train
solid: test



CIFAR-10 Experiments

Batch Normalization

Reduces “internal covariate shift”



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe SIOFFE@GOOGLE.COM
Christian Szegedy SZEGEDY@GOOGLE.COM
Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.82% top-5 test error, exceeding the accuracy of human raters.

minimize the loss

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\mathrm{x}_i, \Theta)$$

where $\mathrm{x}_{1\ldots N}$ is the training data set. With SGD, the training proceeds in steps, at each step considering a *mini-batch* $\mathrm{x}_{1\ldots m}$ of size m. Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch $\frac{1}{m}\sum_{i=1}^{m} \frac{\partial \ell(\mathrm{x}_i,\Theta)}{\partial \Theta}$ is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a mini-batch can be more efficient than m computations for individual examples on modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate and the initial parameter values. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

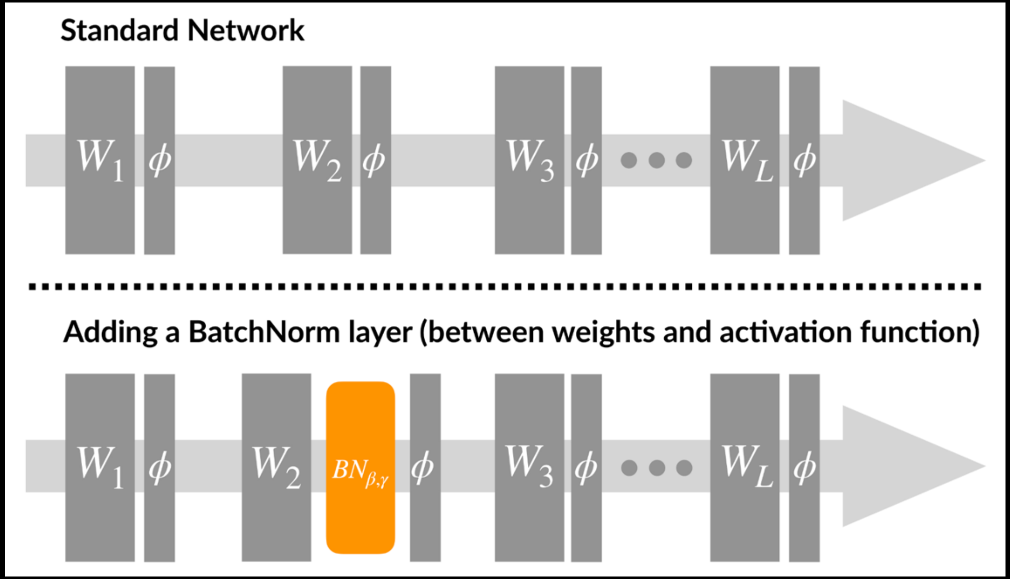
The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience *covari-ate shift* (Shimodaira, 2000). This is typically hand

Batch Normalization

Reduces "internal covariate shift"

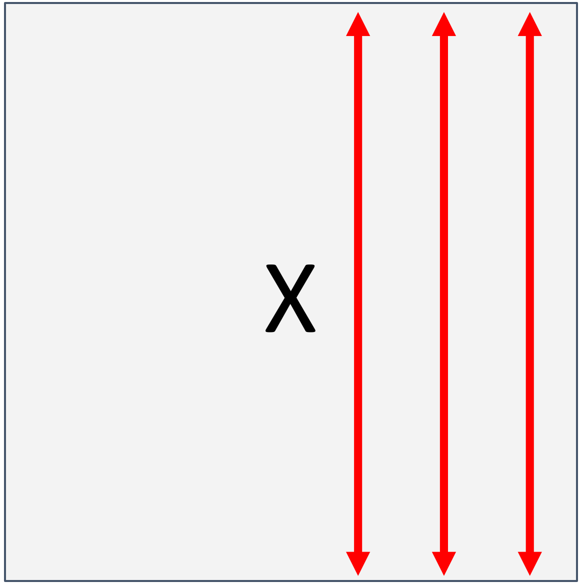
Normalizes activations batch-wise; fully differentiable for backprop

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$



Batch Normalization (Fully Connected Version)

Input: $x : N \times D$



N (mini-batch size)

D (num of features)

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-feature mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

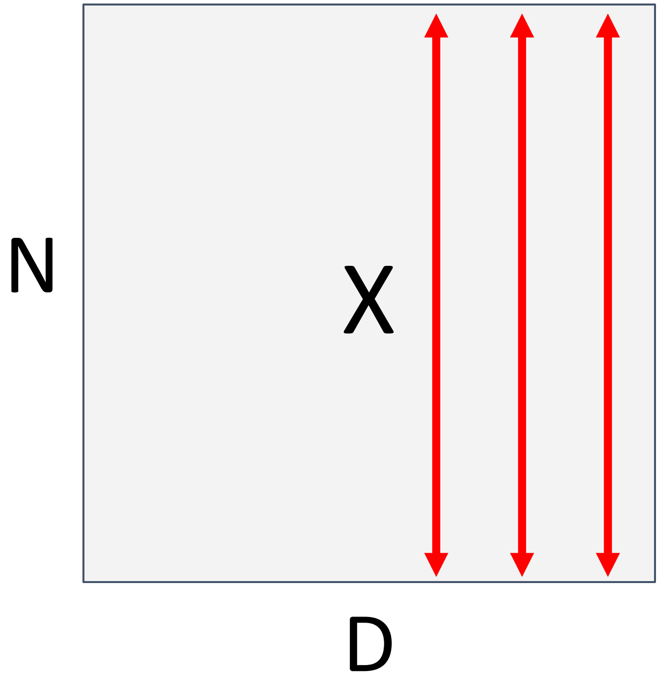
Per-feature std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, **Shape is** N x D

Batch Normalization (Fully Connected Version)

Input: $x : N \times D$



$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

Per-feature mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-feature std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, **Shape is** N x D

Problem: What if zero-mean, unit variance is too hard of a constraint?

Batch Normalization (Fully Connected Version)

Input: $x : N \times D$

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-feature mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-feature std, shape is D

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, **Shape is** N x D

Learning $\gamma = \sigma$, $\beta = \mu$ **will recover the identity function**!

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batchnorm eliminates the need for bias terms

Batch Normalization : Test-Time

Minibatch-dependent estimates

Input: $x : N \times D$

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-feature mean, shape is D

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-feature std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, **Shape is** N x D

Learning $\gamma = \sigma$, $\beta = \mu$ **will recover the identity function!**

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batch Normalization : Test-Time

Input: $x : N \times D$

$\mu_j =$ (Running) average of values seen during training

Per-feature mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta : D$

$\sigma_j^2 =$ (Running) average of values seen during training

Per-feature std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x,
Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output,
Shape is N x D

-During testing, batchnorm becomes a fixed linear (affine) transformation.
-Can be fused with previous weight layer with no extra overhead

Batch Normalization for ConvNets

Batch Normalization for fully-connected networks

$$\mathbf{x}: N \times D$$

Normalize $\downarrow$ (over N)

$$\mu, \sigma: 1 \times D$$

$$\gamma, \beta: 1 \times D$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Batch Normalization for convolutional networks (Spatial Batchnorm, BatchNorm2D)

$$\mathbf{x}: N \times C \times H \times W$$

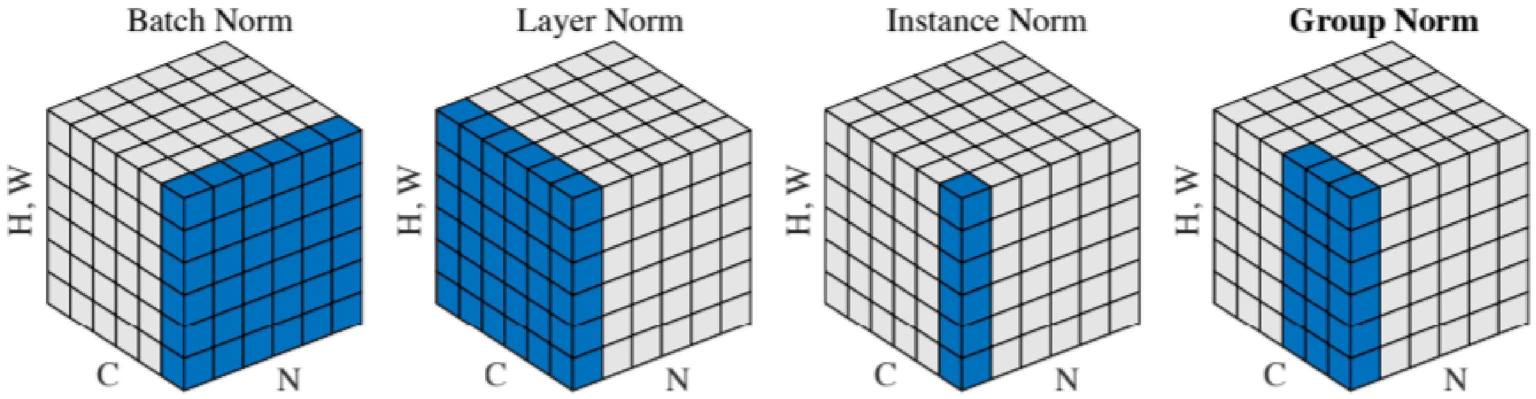
Normalize

$$\mu, \sigma: 1 \times C \times 1 \times 1$$

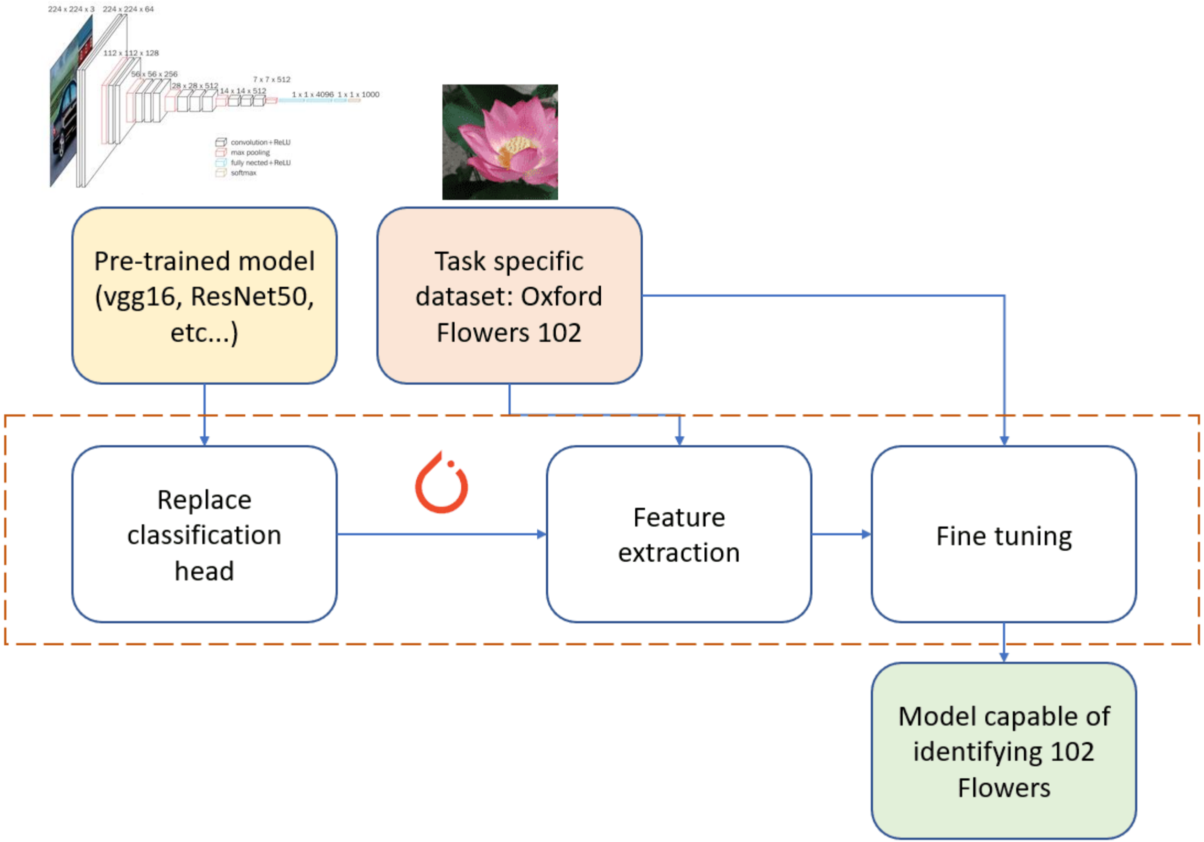
$$\gamma, \beta: 1 \times C \times 1 \times 1$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Other Normalizations

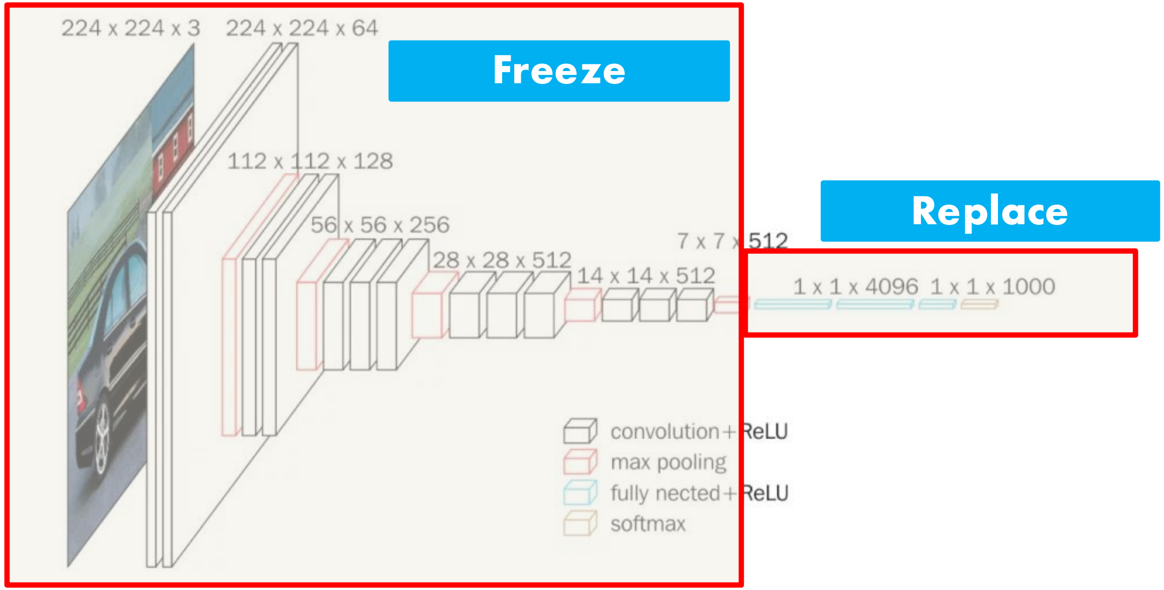


Transfer Learning: Fine-Tuning for a New Task

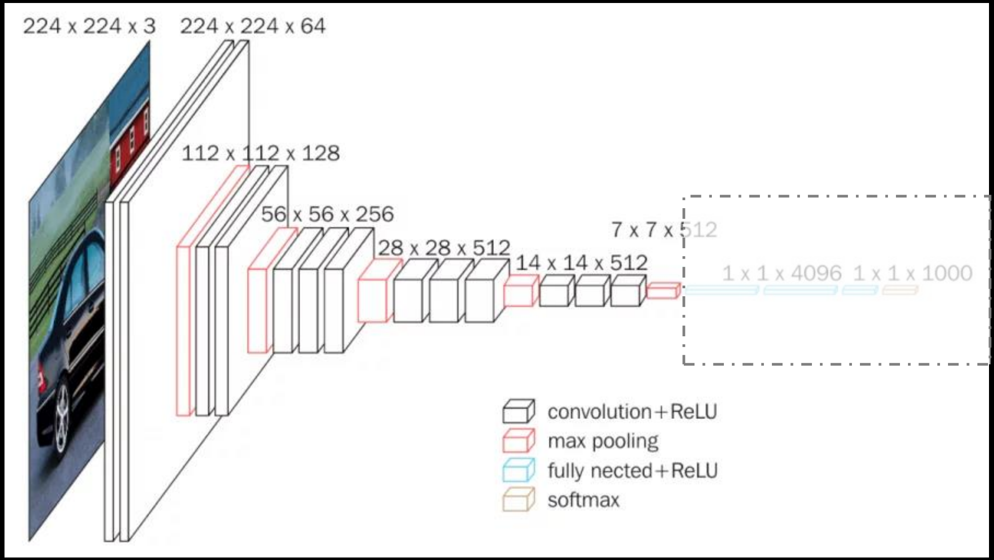


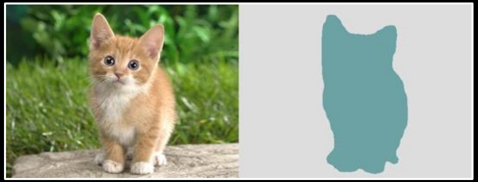
Transfer Learning

- Optimized learning with scarce data, freeze early layers, replace final classification layers
- Optionally fine-tune deeper layers if the new domain differs significantly



Versatile Applications of CNNs





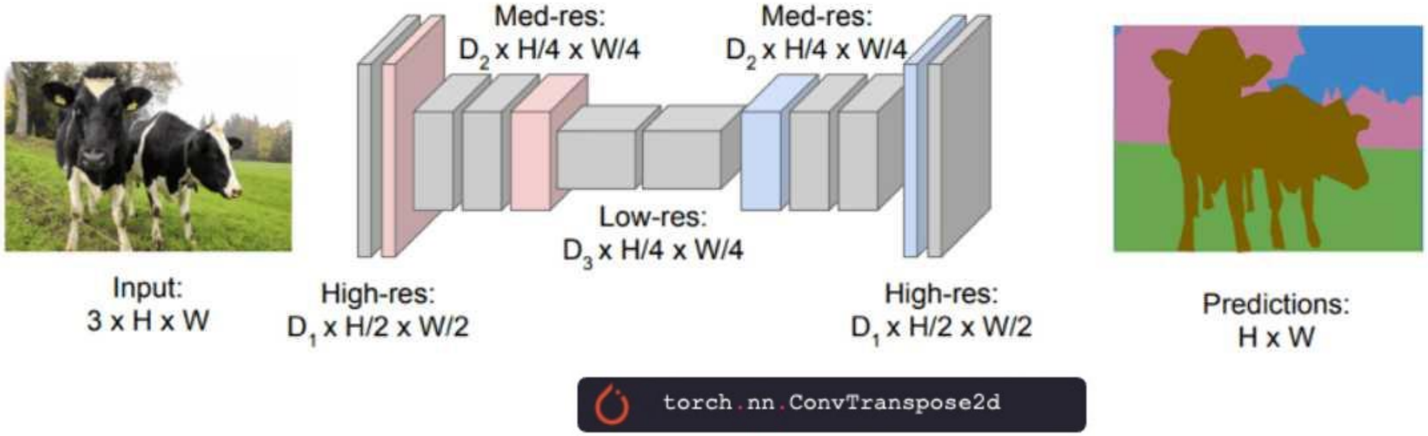
Segmentation

Object Detection

Semantic Segmentation: Fully Convolutional Networks

- Network designed with only convolutional layers, handling arbitrary input sizes
- Uses downsampling and upsampling operations (transpose convolutions)

📌 Recent approaches like U-Net and other encoder-decoder designs follow a similar paradigm



You Only Look Once (YOLO)

You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detec-ors. Compared to state-of-the-art detection systems, YOLO kes more localization errors but is less likely to predict sitives on background. Finally, YOLO learns very

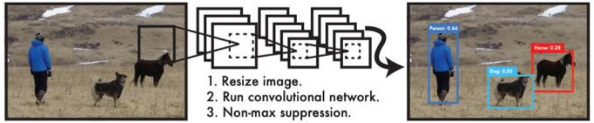
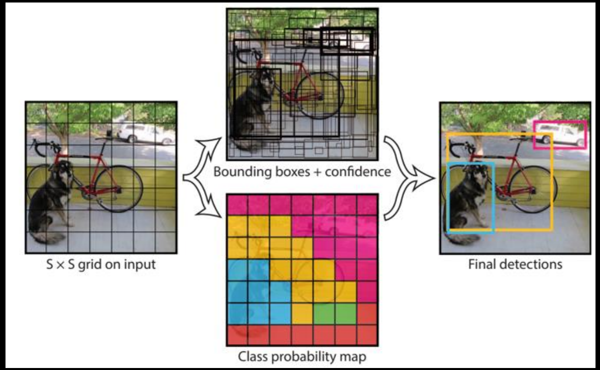
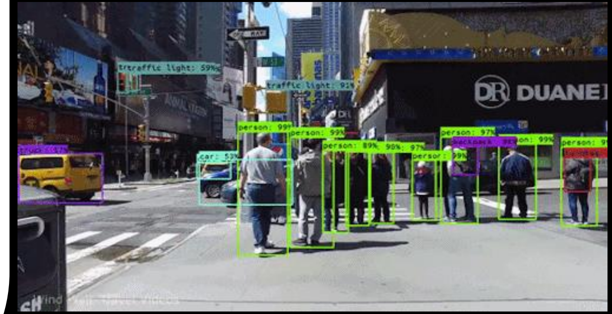


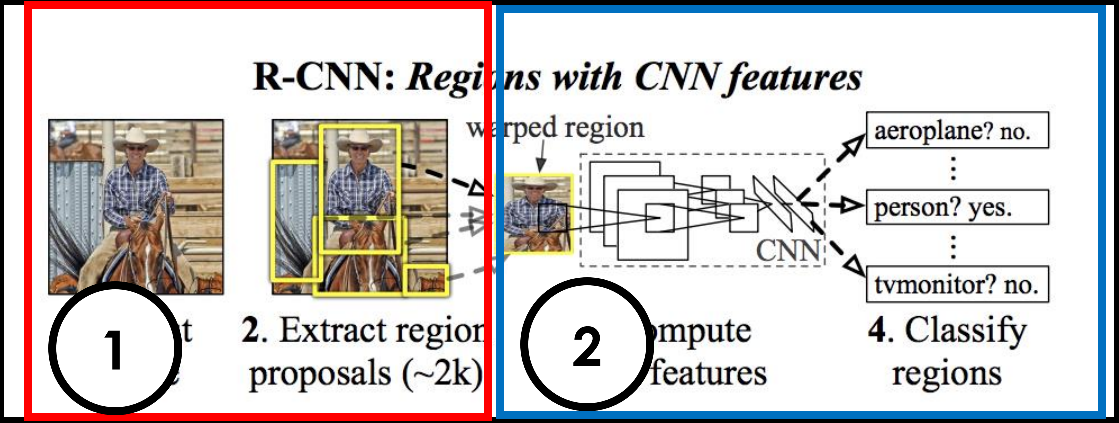
Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

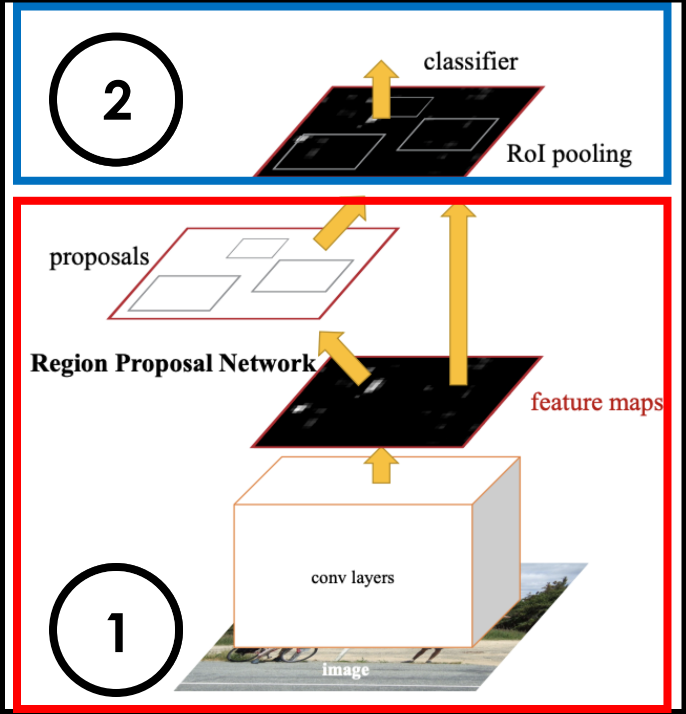
methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [13]. These complex pipelines are slow and hard to optimize because e





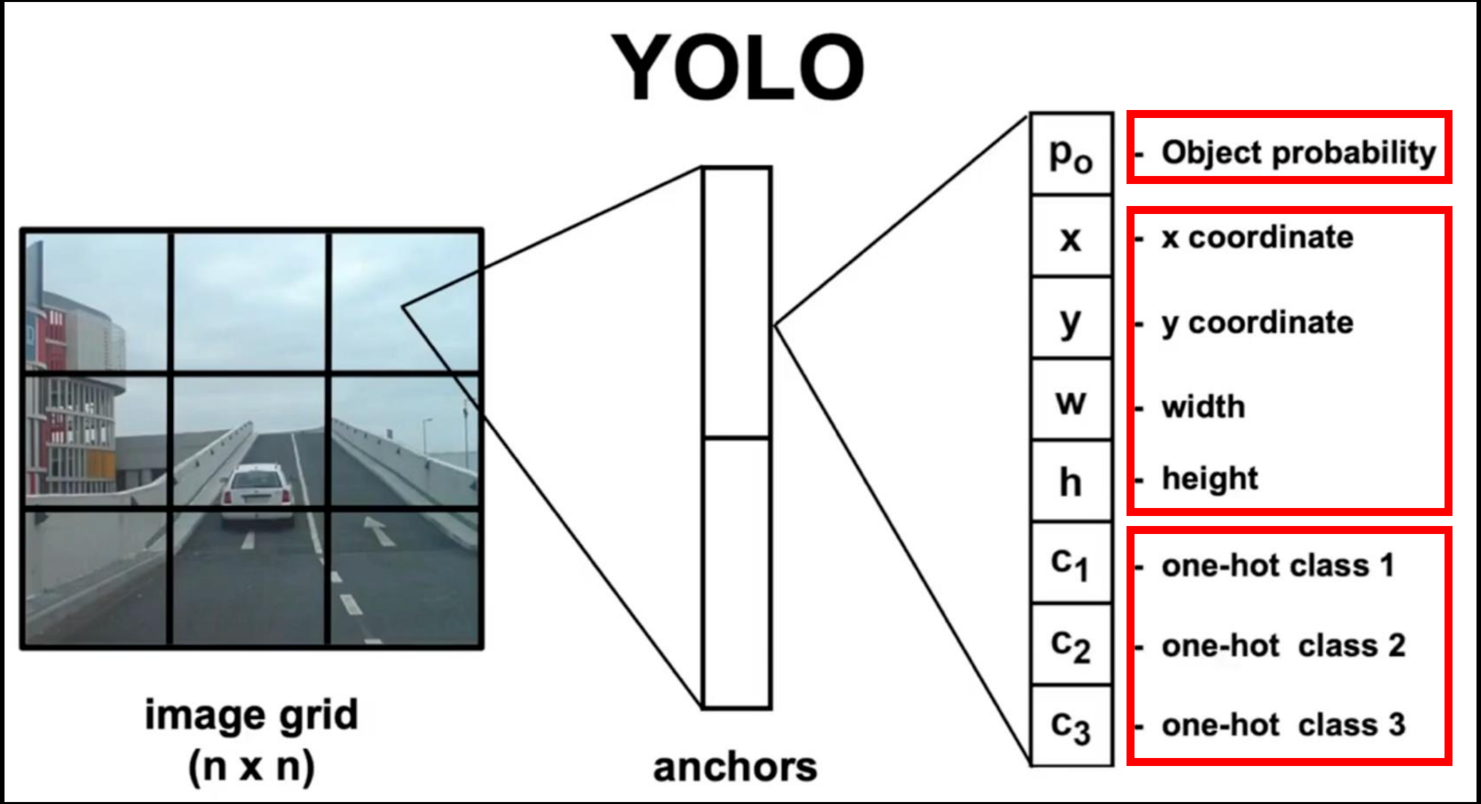
Prior Two-Step Object Detection Approaches





[Girshick et al., CVPR'14] Rich feature hierarchies for accurate object detection and semantic segmentation
[Ren et al., NIPS'15] Faster R-CNN: Towards real-time object detection with region proposal networks

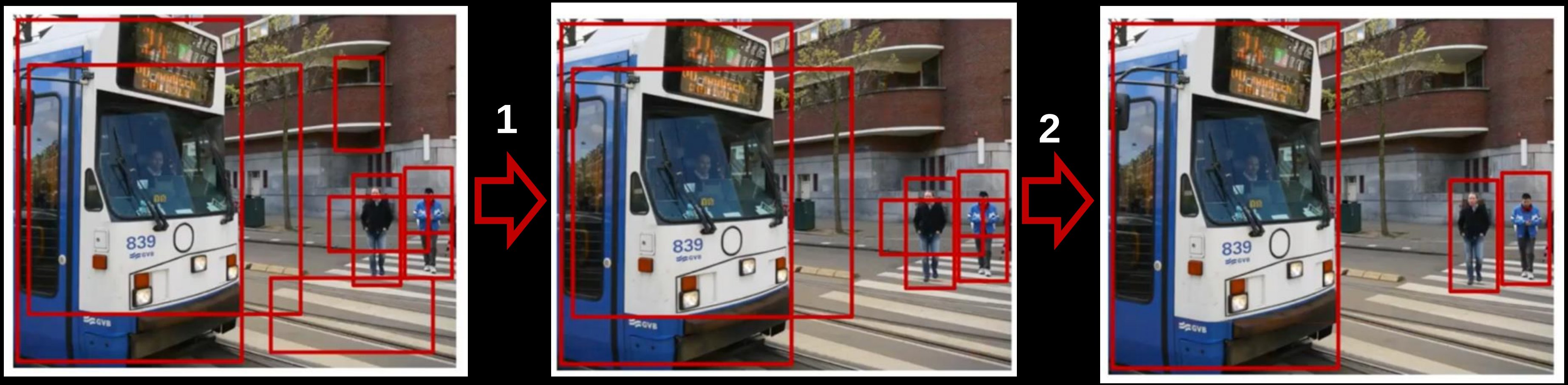
YOLO Grid-Based Prediction



Prediction Post-Processing in YOLO

1- Remove the low probability bounding boxes

2- Apply non-max suppression (NMS)



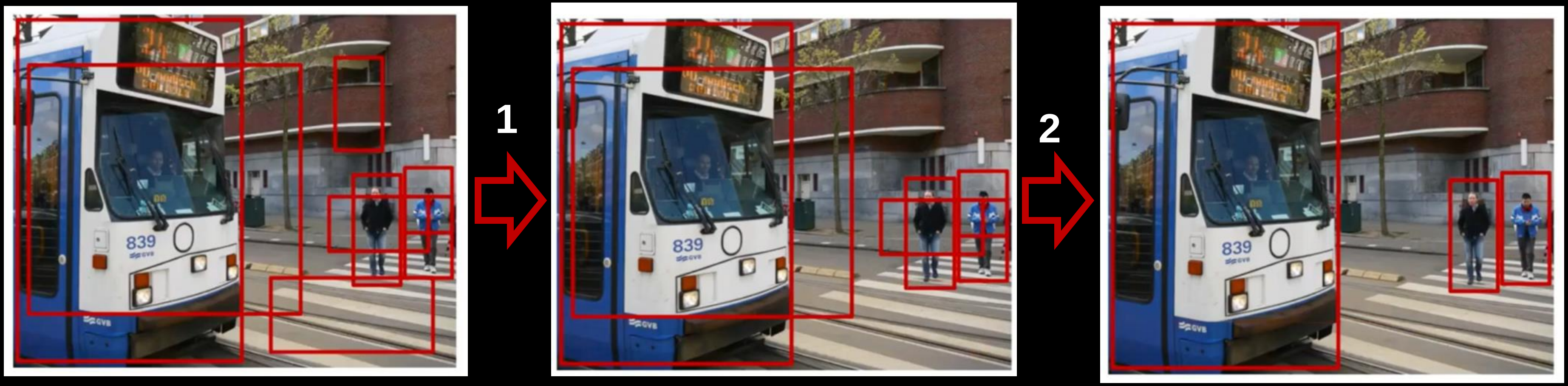
Prediction Post-Processing in YOLO

1- Remove the low probability bounding boxes

2- Apply non-max suppression (NMS)

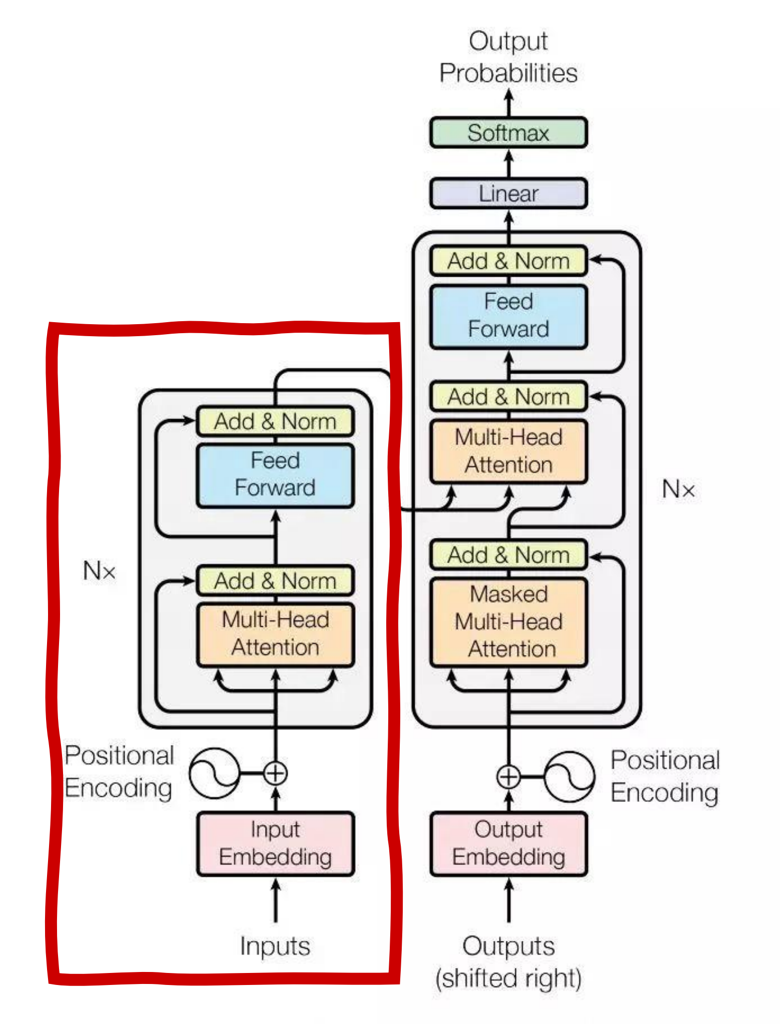
Limitations: struggles with small objects/crowded scene

🚀 YOLOv3, v4, v5, v7, and YOLOv8 add multi-scale predictions & stronger backbones.



Vision Transformer

Intuition and Overview



2017

Attention Is All You Need

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*], **Xiaohua Zhai**[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby**[*,†]

[*]equal technical contribution, [†]equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

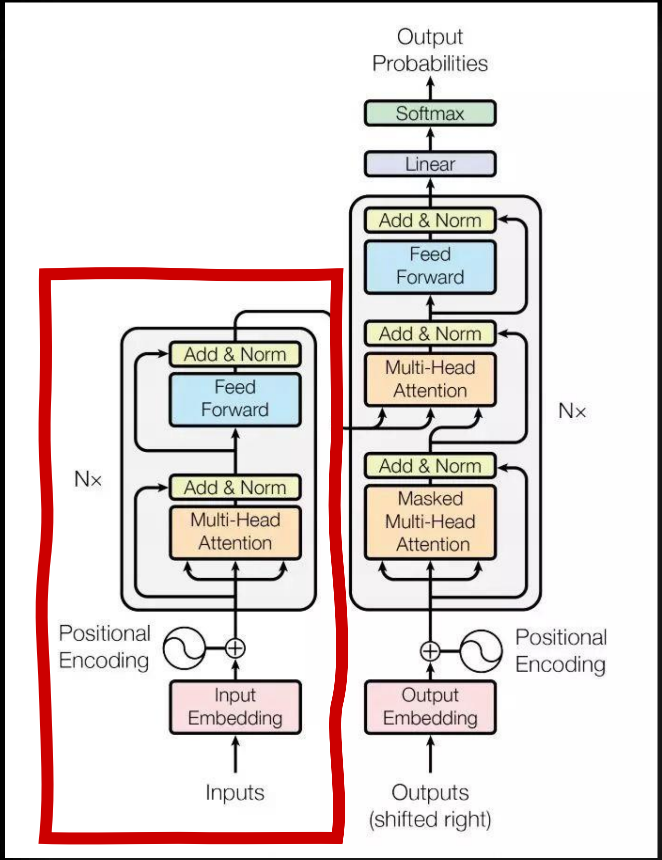
Abstract

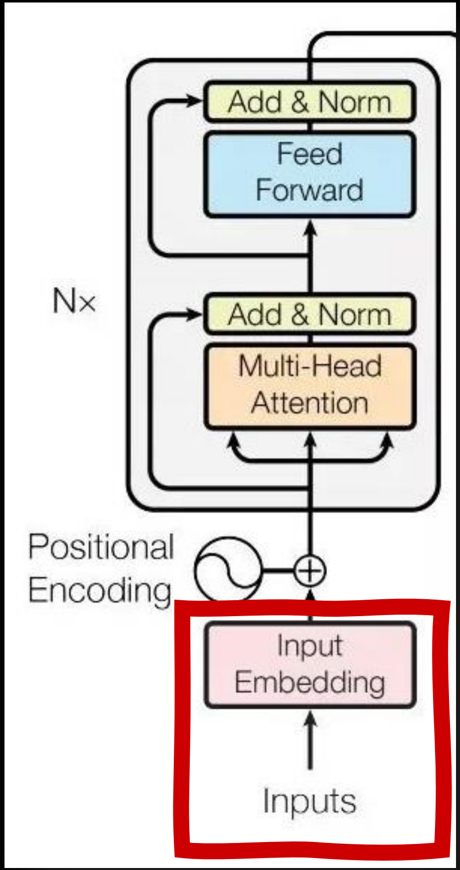
While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.[1]

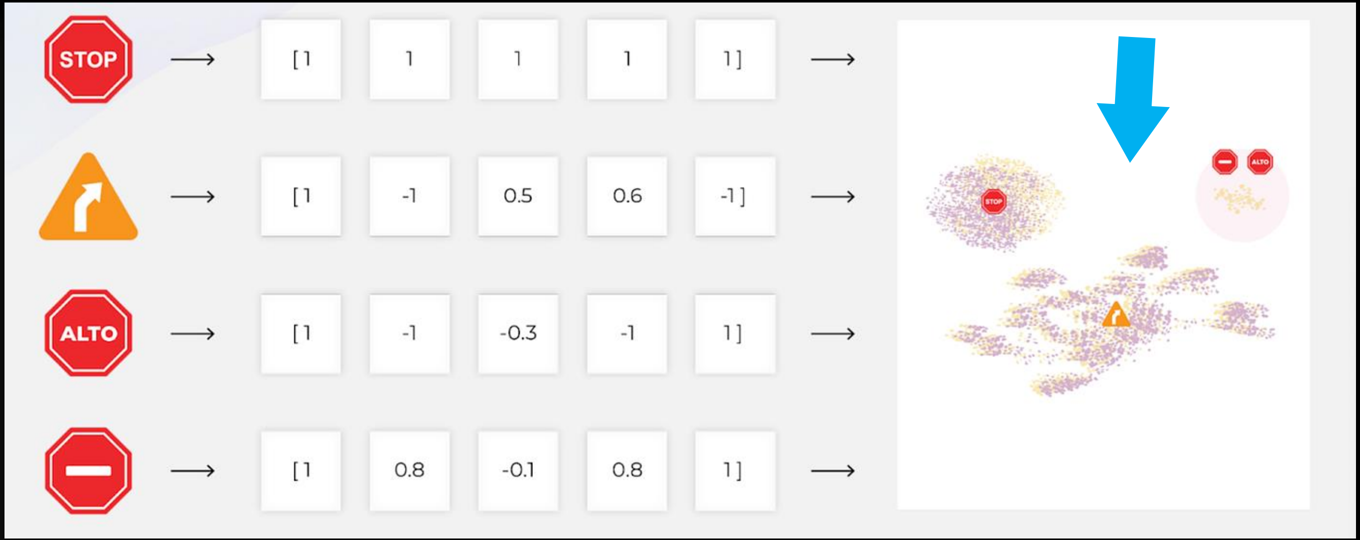
1 Introduction

Self-attention-based architectures, in particular Transformers (Vaswani et al., 2017), have become the model of choice in natural language processing (NLP). The dominant approach is to pre-train on a large text corpus and then fine-tune on a smaller task-specific dataset (Devlin et al., 2019). Thanks to Transformers' computational efficiency and scalability, it has become possible to train models of unprecedented size, with over 100B parameters (Brown et al., 2020; Lepikhin et al., 2020). With the models and datasets growing, there is still no sign of saturating performance.

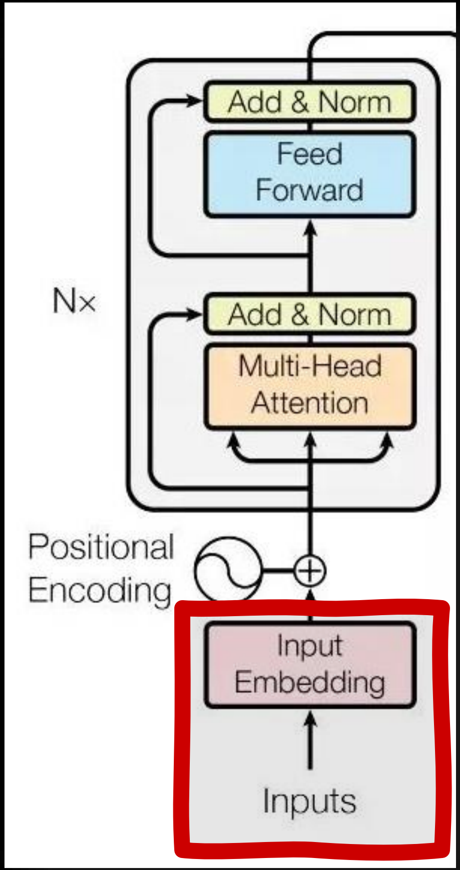
In computer vision, however, convolutional architectures remain dominant (LeCun et al., 1989; Krizhevsky et al., 2012; He et al., 2016). Inspired by NLP successes, multiple works try combining CNN-like architectures with self-attention (Wang et al., 2018; Carion et al., 2020), some replacing the convolutions entirely (Ramachandran et al., 2019; Wang et al., 2020a). The latter models, while theoretically efficient, have not yet been scaled effectively on modern hardware accelerators due to the use of specialized attention patterns. Therefore, in large-scale image recognition, classic ResNet-like architectures are still state of the art (Mahajan et al., 2018; Xie et al., 2020; Kolesnikov et al., 2020).







By Francisco Castillo Carrasco (towards data science)



Input Embedding

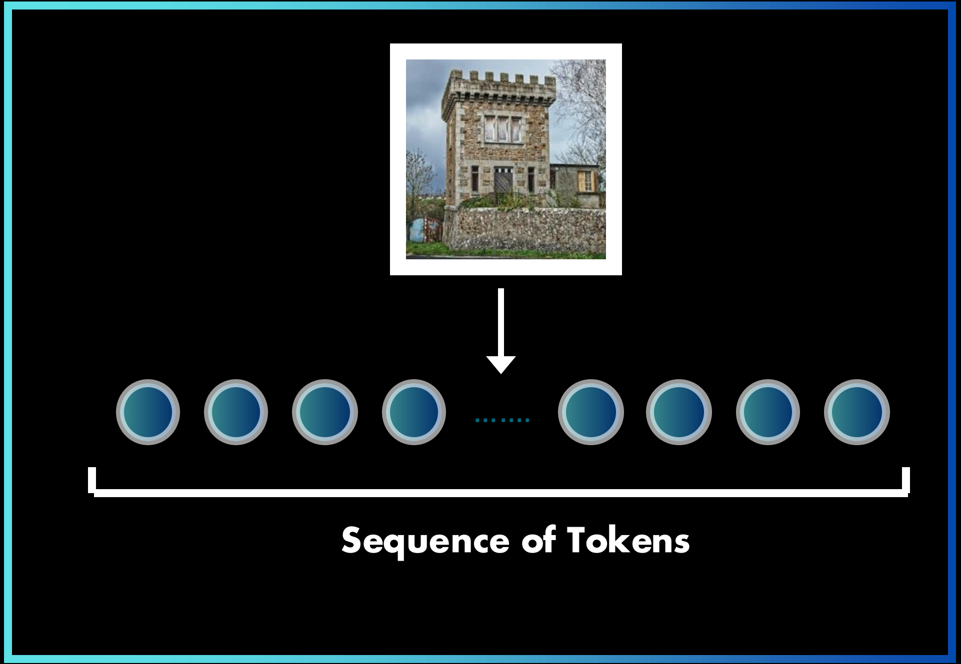
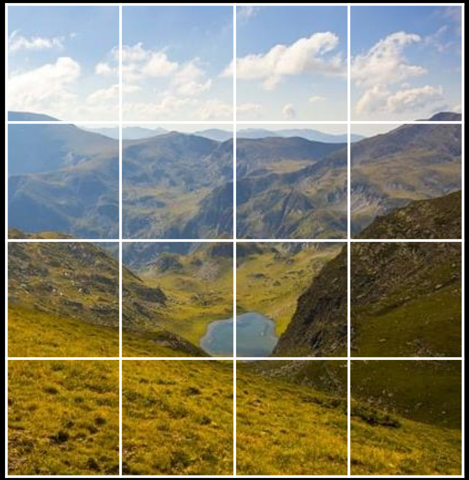
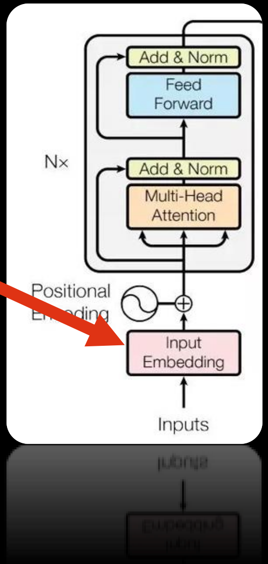
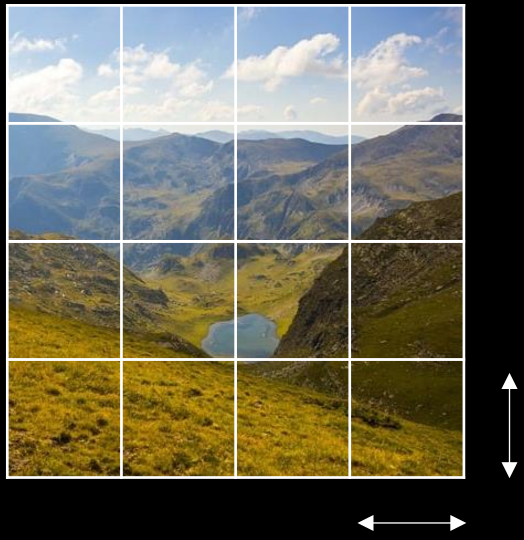


Image Patching

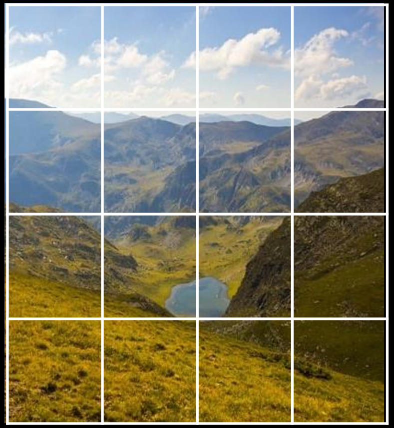


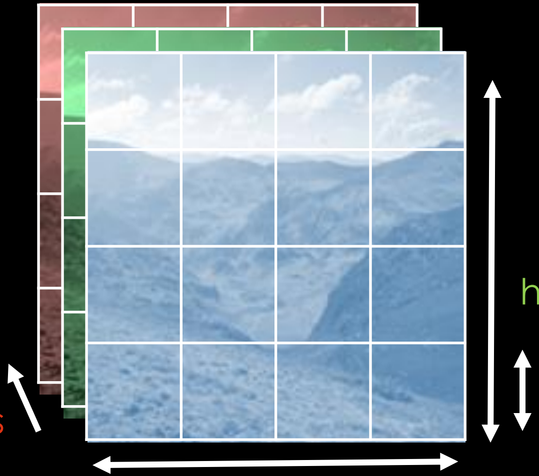

Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional
Input Embedding
Inputs



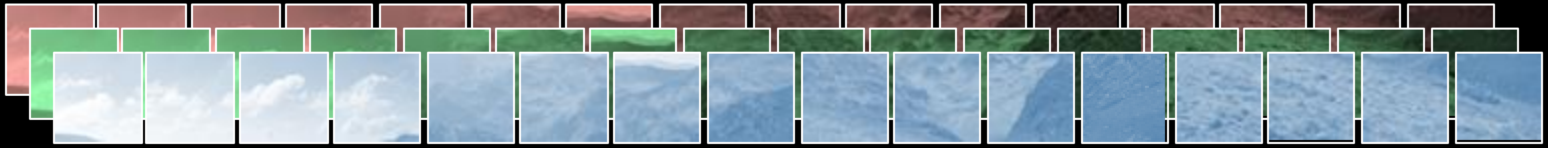
16 px

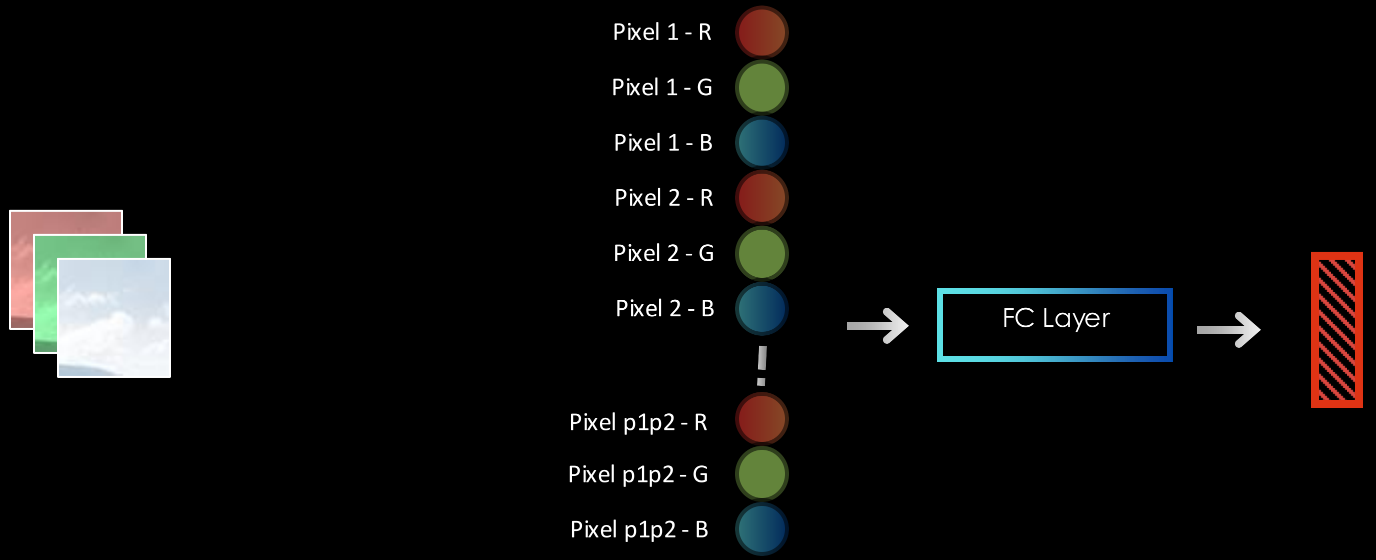
16 px





Number of Patches x c x p1 x p2



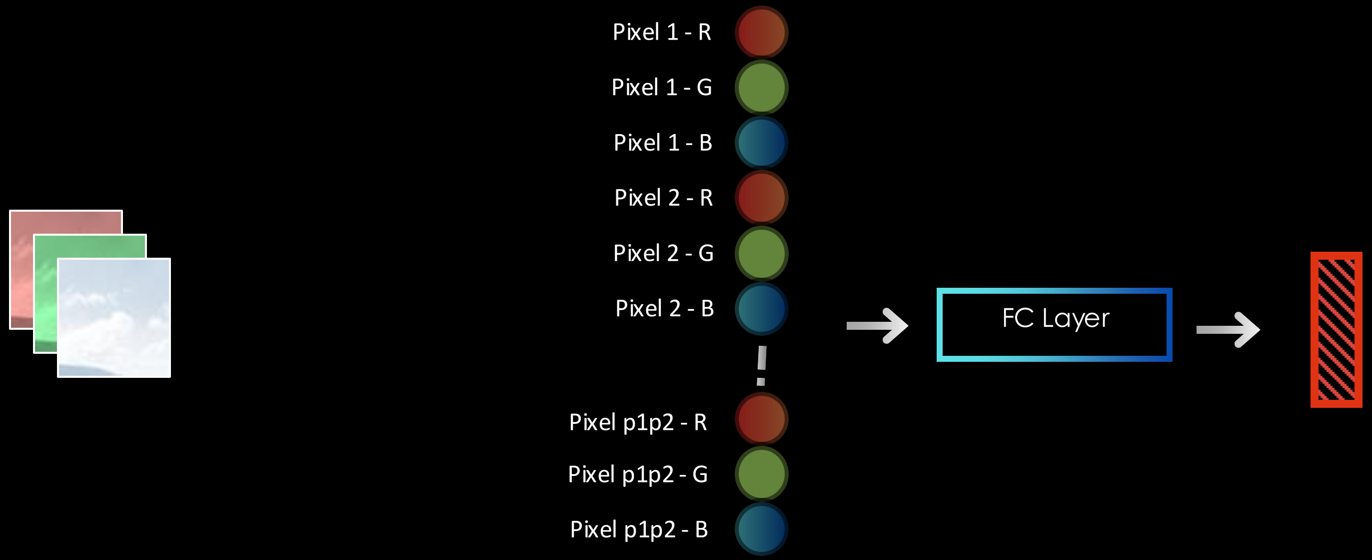


Number of Patches x c x p1 x p2

Number of Patches x (c x p1 x p2)

Number of Patches x D

D = Dimension at which transformer layers will operate



Number of Patches x c x p1 x p2

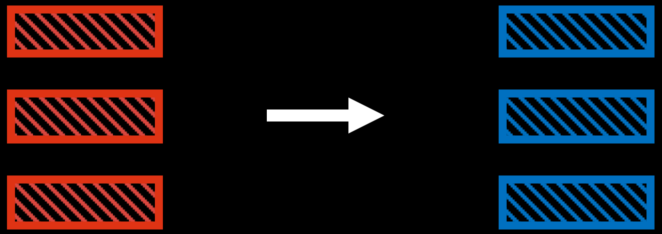
Number of Patches x (c x p1 x p2)

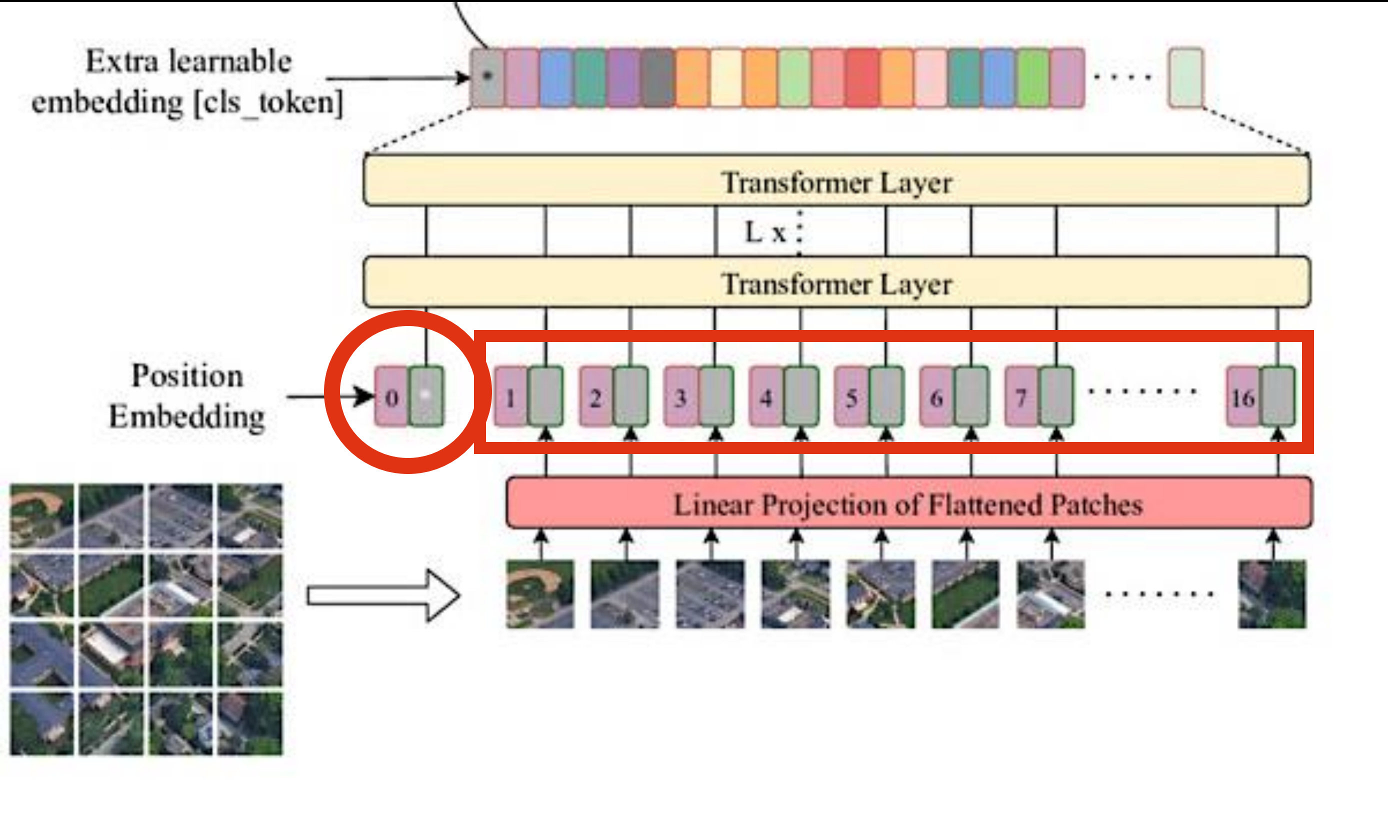
N x D

D = Dimension at which transformer layers will operate

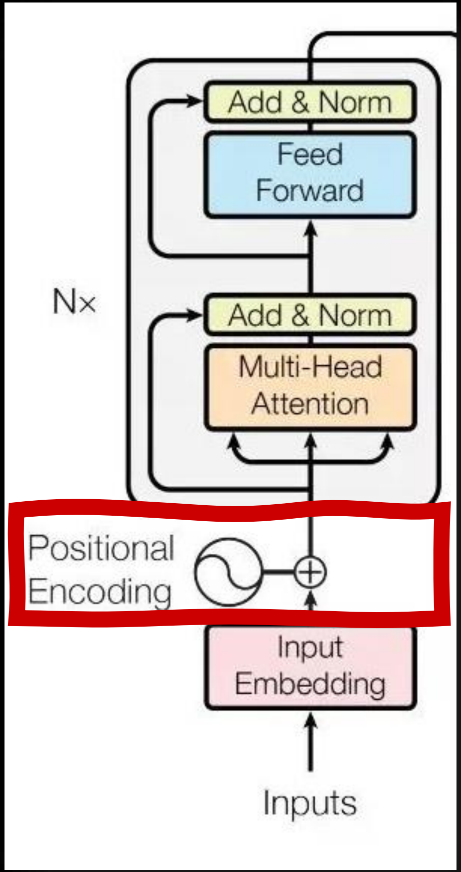
N = Number of patches (size of the sequence of tokens)

The CLS Token





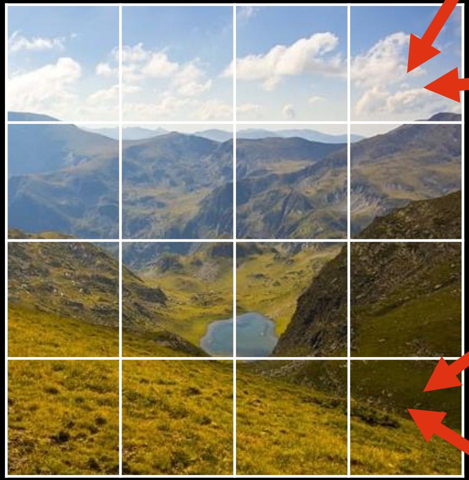
Source: A Transformer-Based Feature Segmentation and Region Alignment Method For UAV-View Geo-Localization

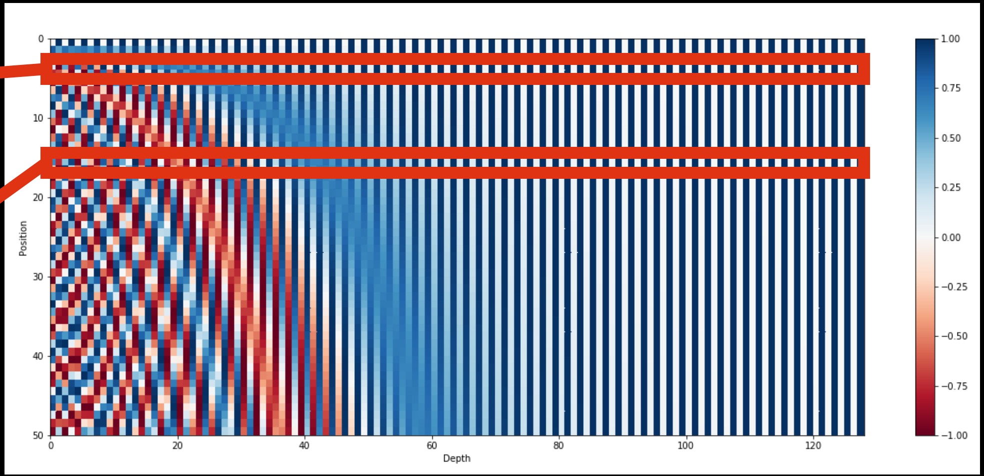


The Positional Embedding

Fixed vs. Learned Positional Embeddings

That's patch #4





Source: Amirhossein Kazemnejad's Blog

That's patch #16

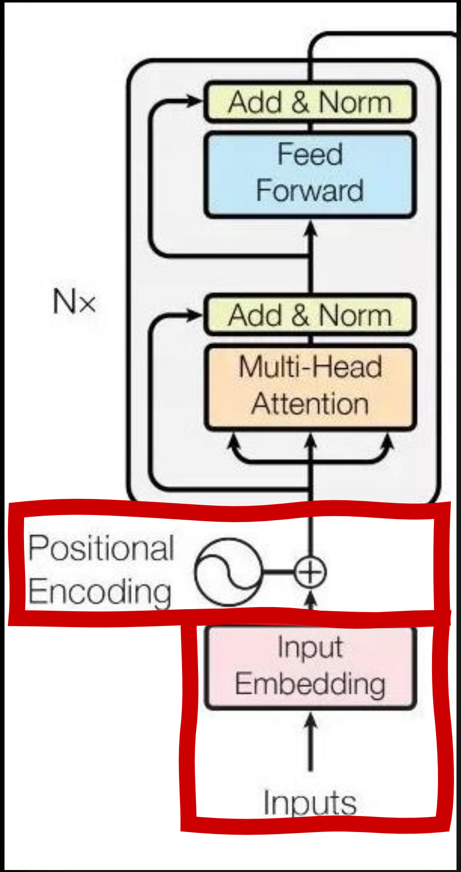
Fixed vs. Learned Positional Embeddings

Pos. Emb.	Default/Stem	Every Layer	Every Layer-Shared
No Pos. Emb.	0.61382	N/A	N/A
1-D Pos. Emb.	0.64206	0.63964	0.64292
2-D Pos. Emb.	0.64001	0.64046	0.64022
Rel. Pos. Emb.	0.64032	N/A	N/A

Table 8: Results of the ablation study on positional embeddings with ViT-B/16 model evaluated on ImageNet 5-shot linear.

2020

An Image is Worth 16× 16 Words



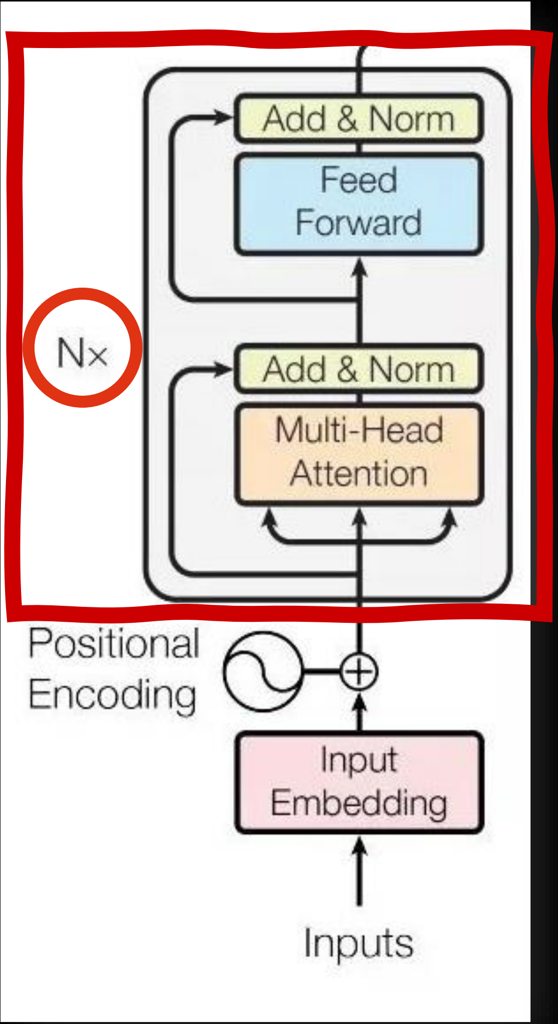
Patch Embedding

Convert Image into Sequence of Patches

Add CLS token to sequence of Patches

Add Positional Information to Patches

Attention Mechanism and Transformer Encoder


Add & Norm
Feed
Forward
Nx
Add & Norm
Multi-Head
Attention
Positional
Encoding
Input
Embedding
Inputs

Attention

Block used in transformers that fulfils two responsibilities:

- **Identifying what is relevant to an input out of everything in its context**
- **Add more meaning to the representation of an entity by using the representation of its context**

But why ?

season

season

Attention

Block used in transformers that fulfils two responsibilities:

- **Identifying what is relevant to an input out of everything in its context**
- **Add more meaning to the representation of an entity by using the representation of its context**

But why ?

Show's last season

I hate rainy season

Attention

Block used in transformers that fulfils two responsibilities:

- **Identifying what is relevant to an input out of everything in its context**
- **Add more meaning to the representation of an entity by using the representation of its context**

But why ?

Show's last season

Chicken didn't cross the road because it was

I hate rainy season

Chicken didn't cross the road because it was

Attention

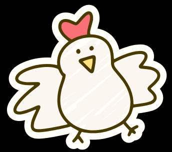
Block used in transformers that fulfils two responsibilities:

- **Identifying what is relevant to an input out of everything in its context**
- **Add more meaning to the representation of an entity by using the representation of its context**

But why ?

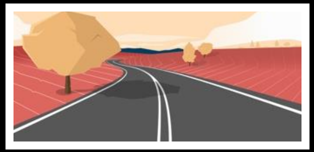
Show's last season

Chicken didn't cross the road because it was happy on this side itself

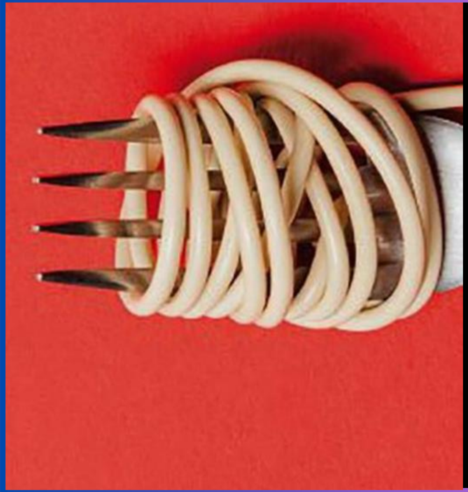


I hate rainy season

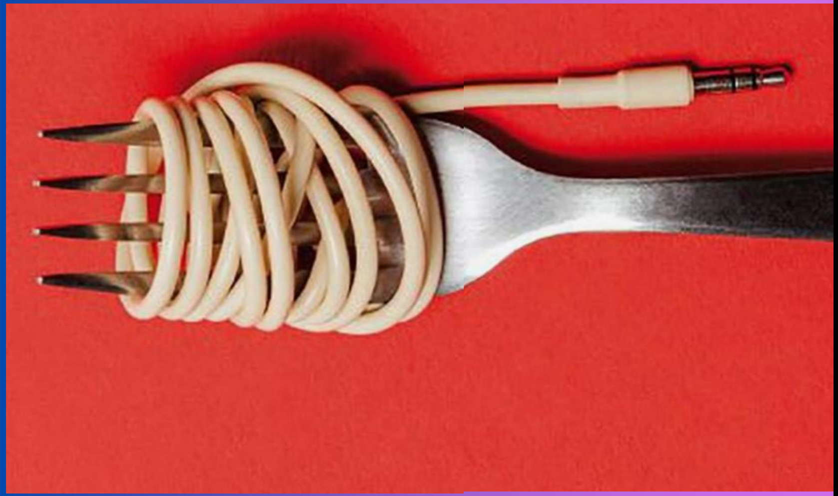
Chicken didn't cross the road because it was too wide



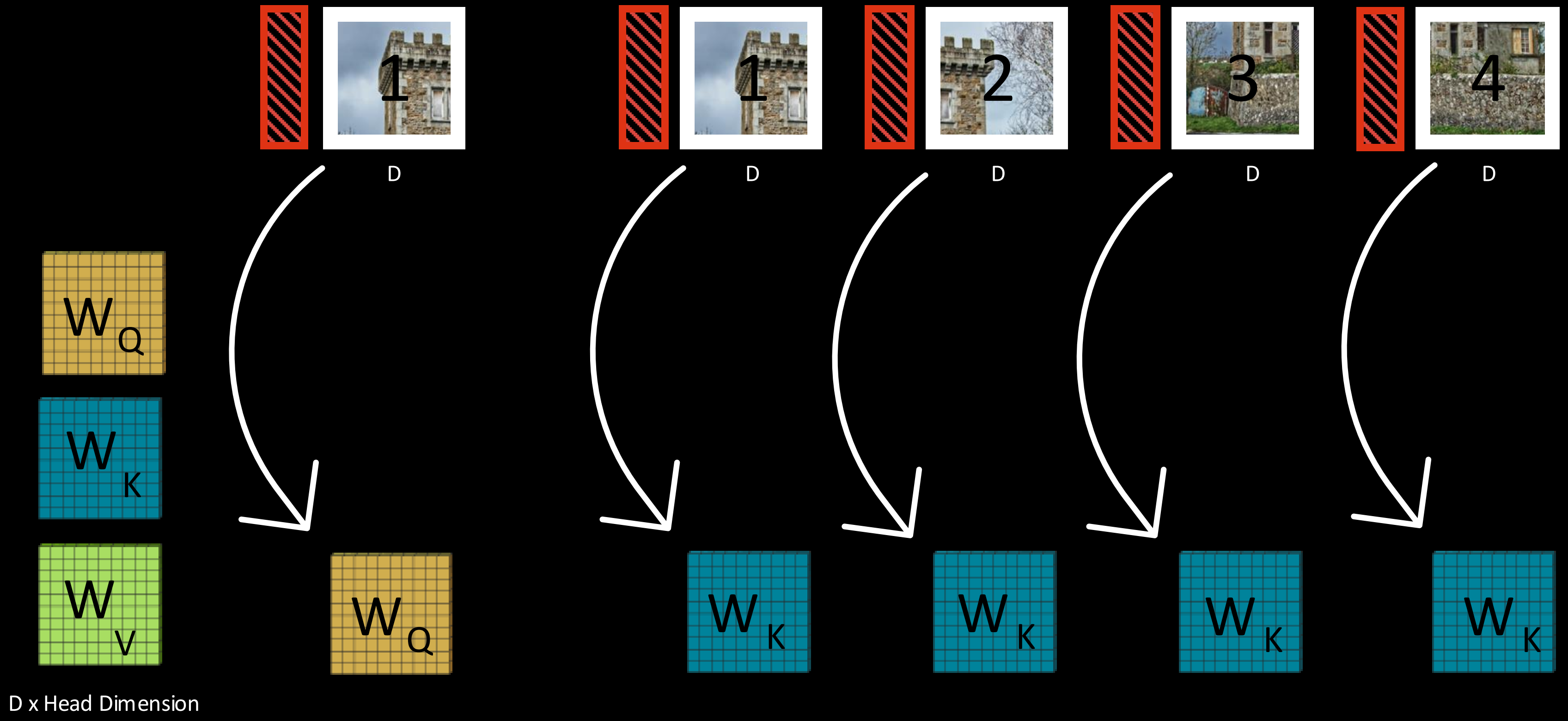
Attention



Attention



Determining Relevance

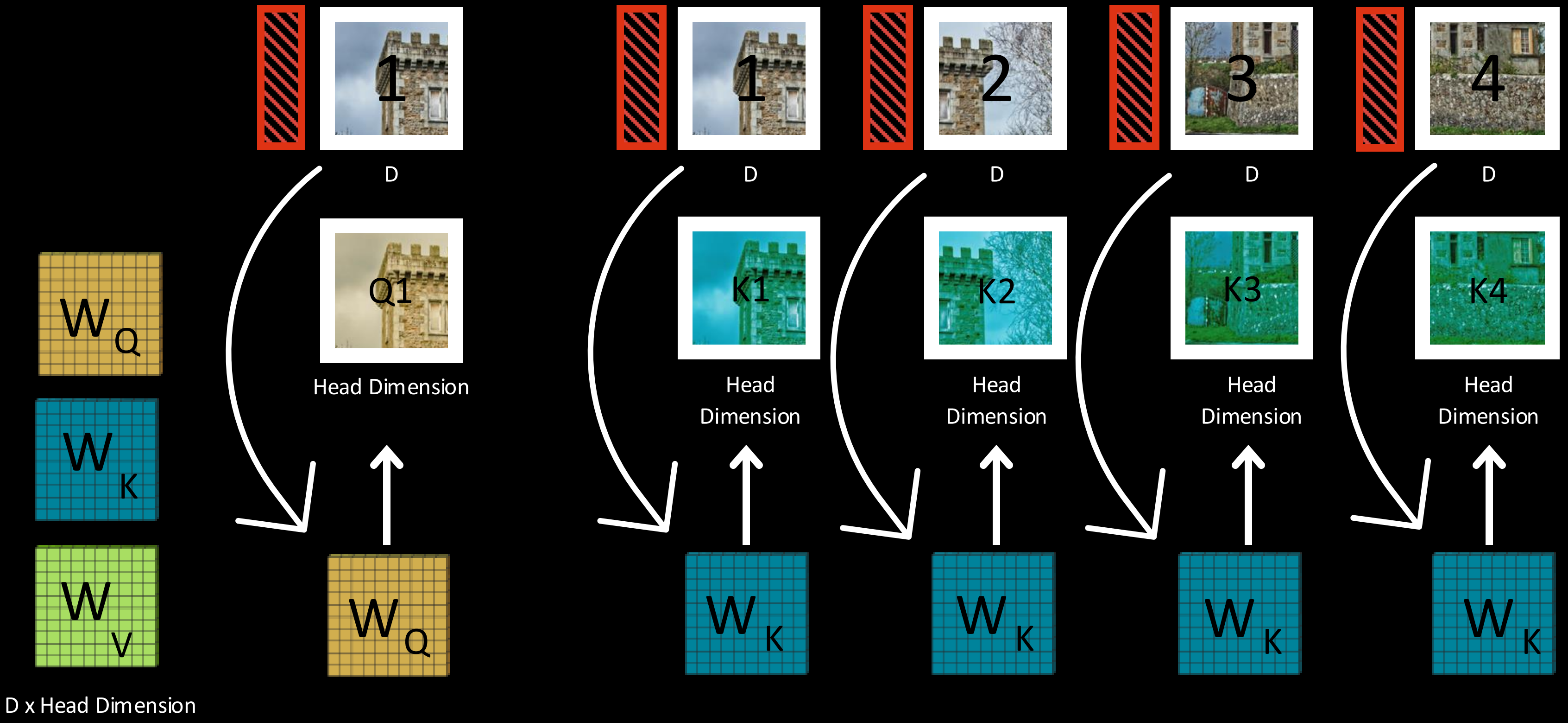


Query : Input representation. We are trying to quantify how much is every context item relevant to this representation

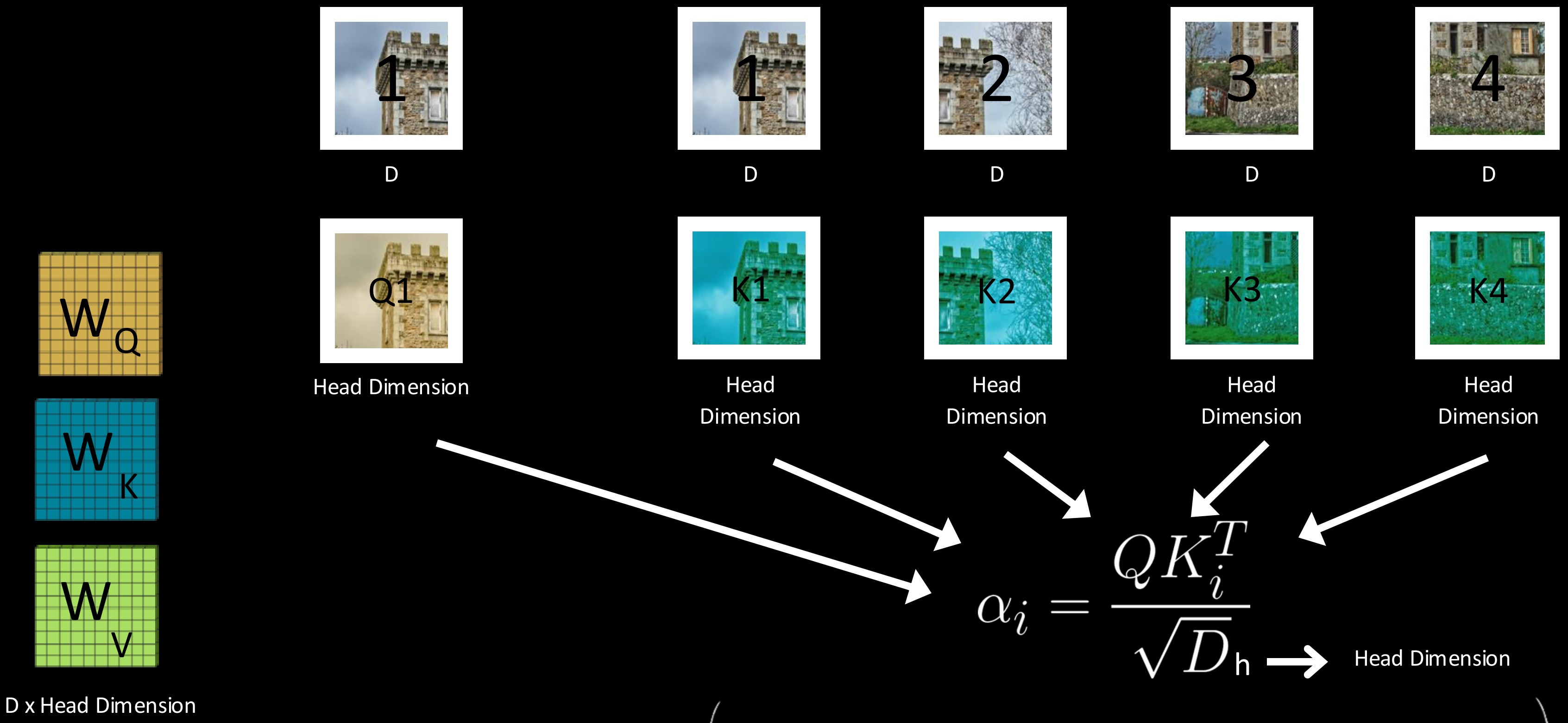
Key : Context representation. Used to quantify relevance to the query representation

Value : Context representation which will be used to add understanding of relevant context into the input

Determining Relevance



Determining Relevance

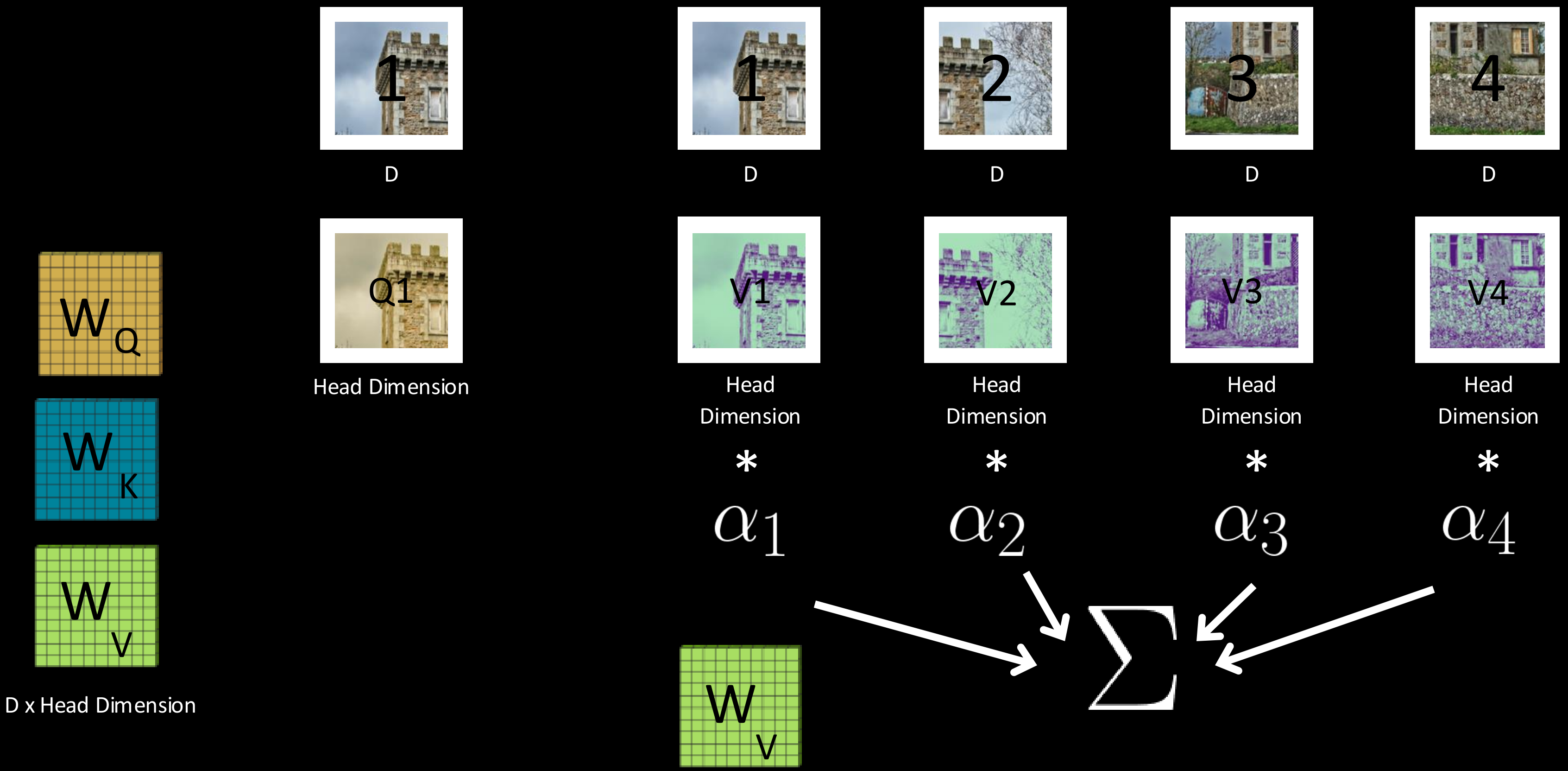


$$\alpha_i = \frac{QK_i^T}{\sqrt{D_h}}$$

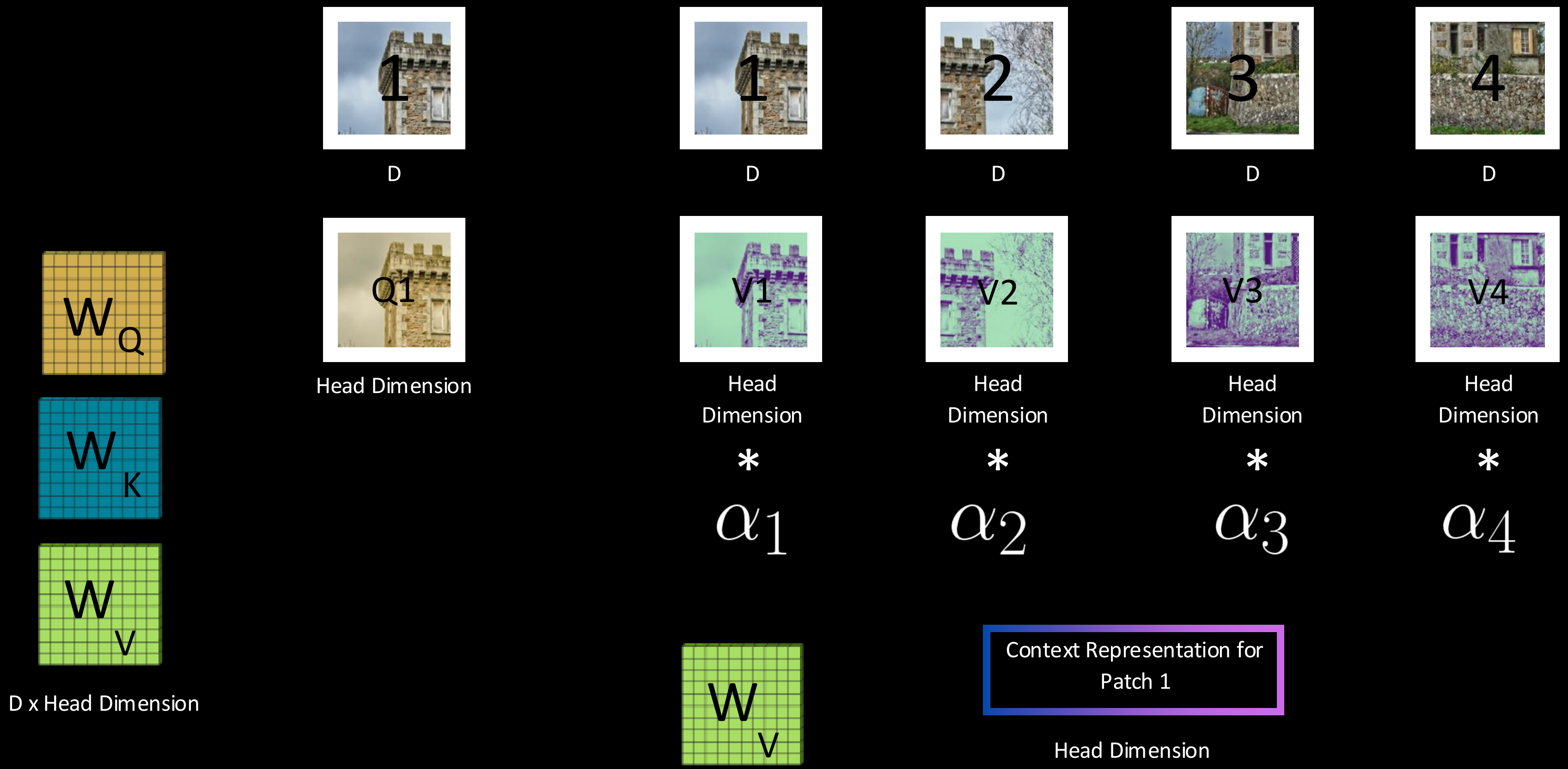
$D_h \rightarrow$ Head Dimension

$$\text{Softmax}\left(\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4\right)$$

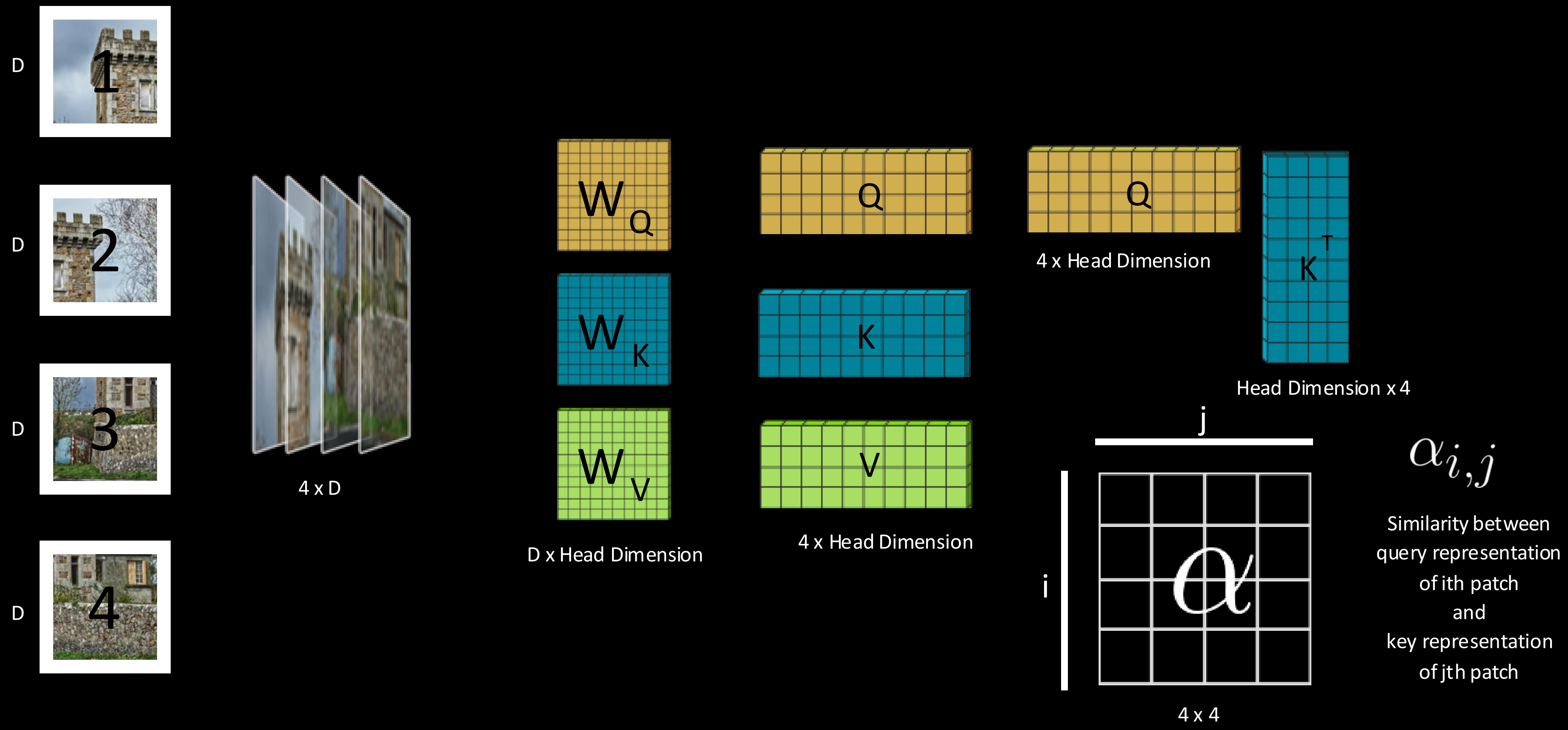
Context-Aware Input Updating



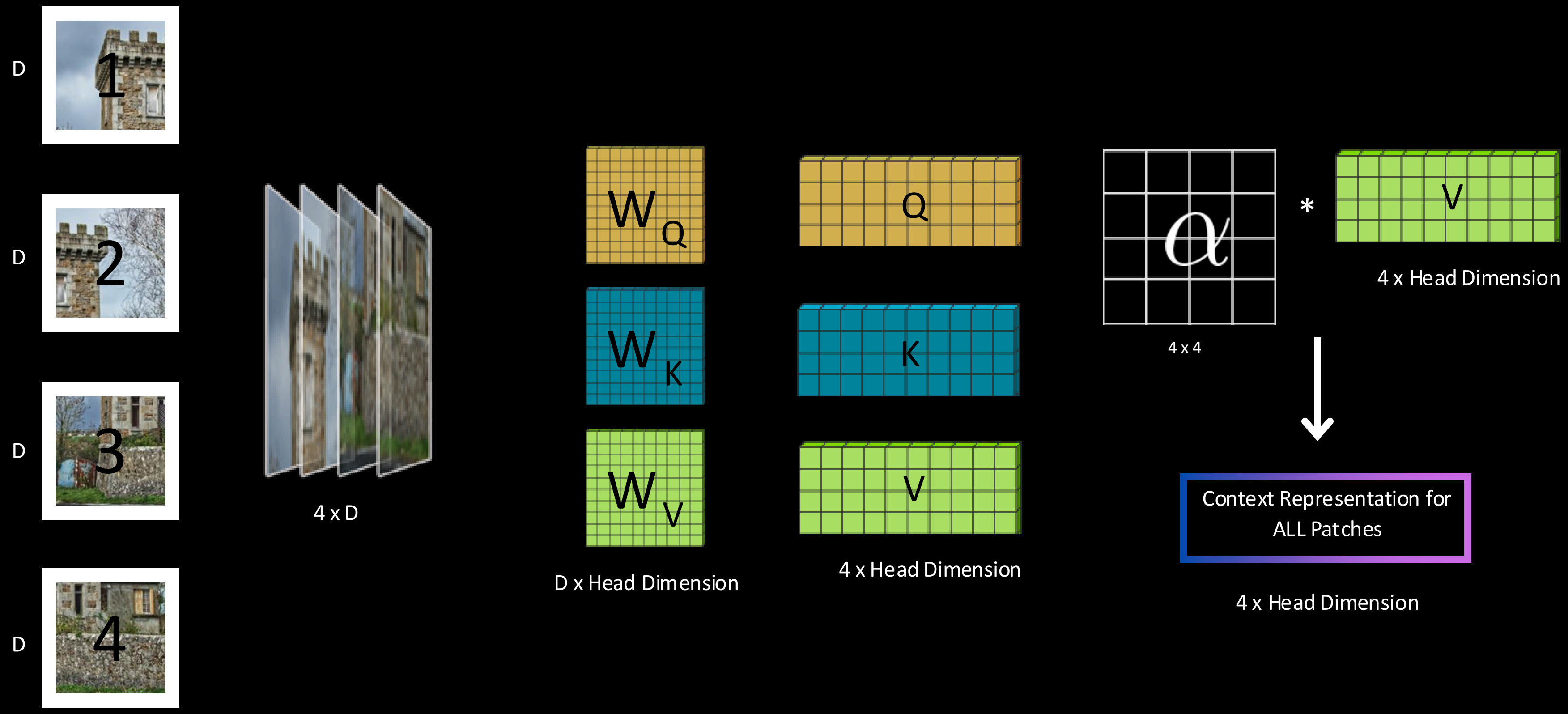
Context-Aware Input Updating



Context-Aware Input Updating (For ALL Patches)



Context-Aware Input Updating (For ALL Patches)



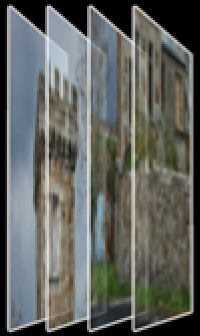
Multi Head Attention

D

D

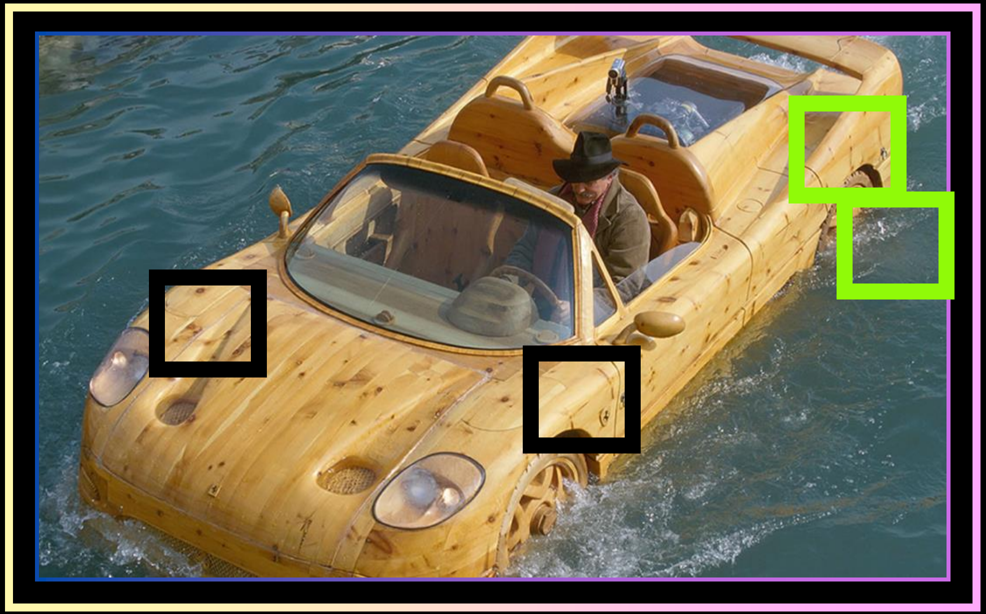
D

D



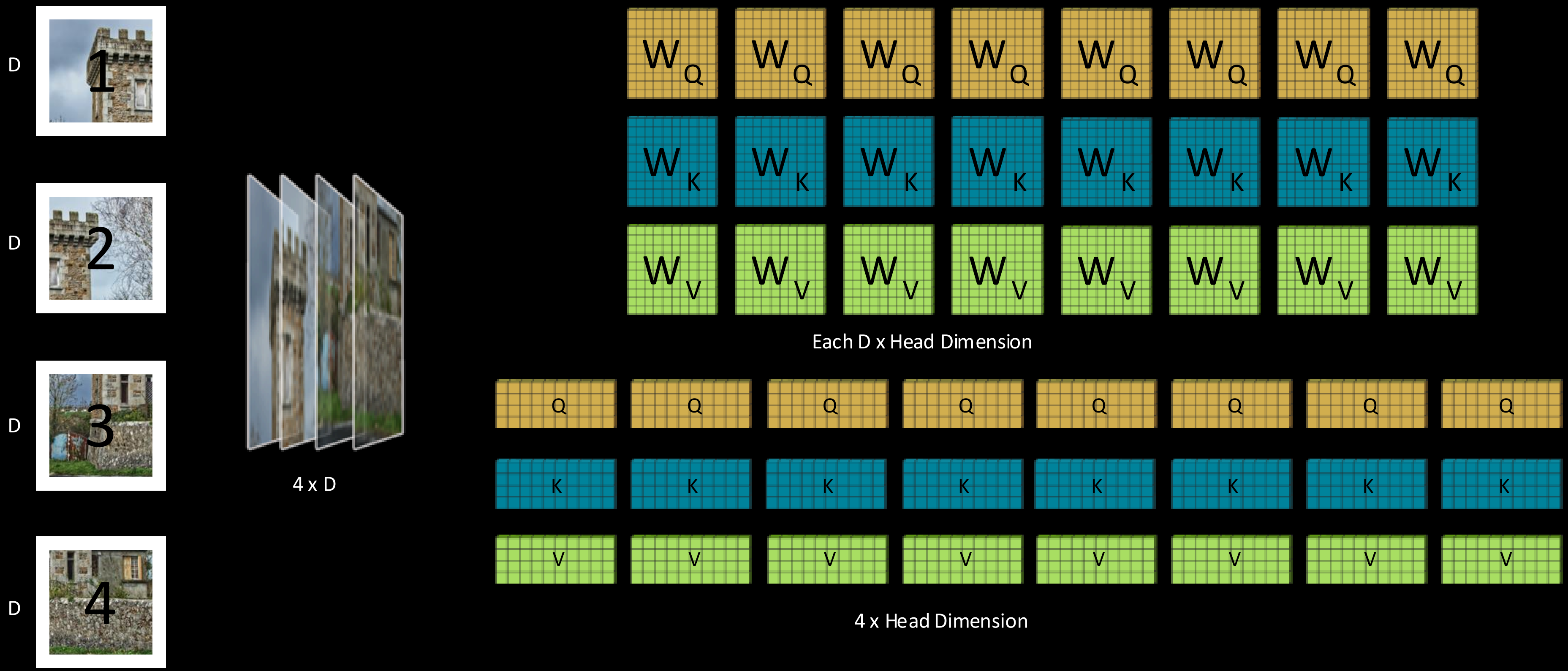
4 x D

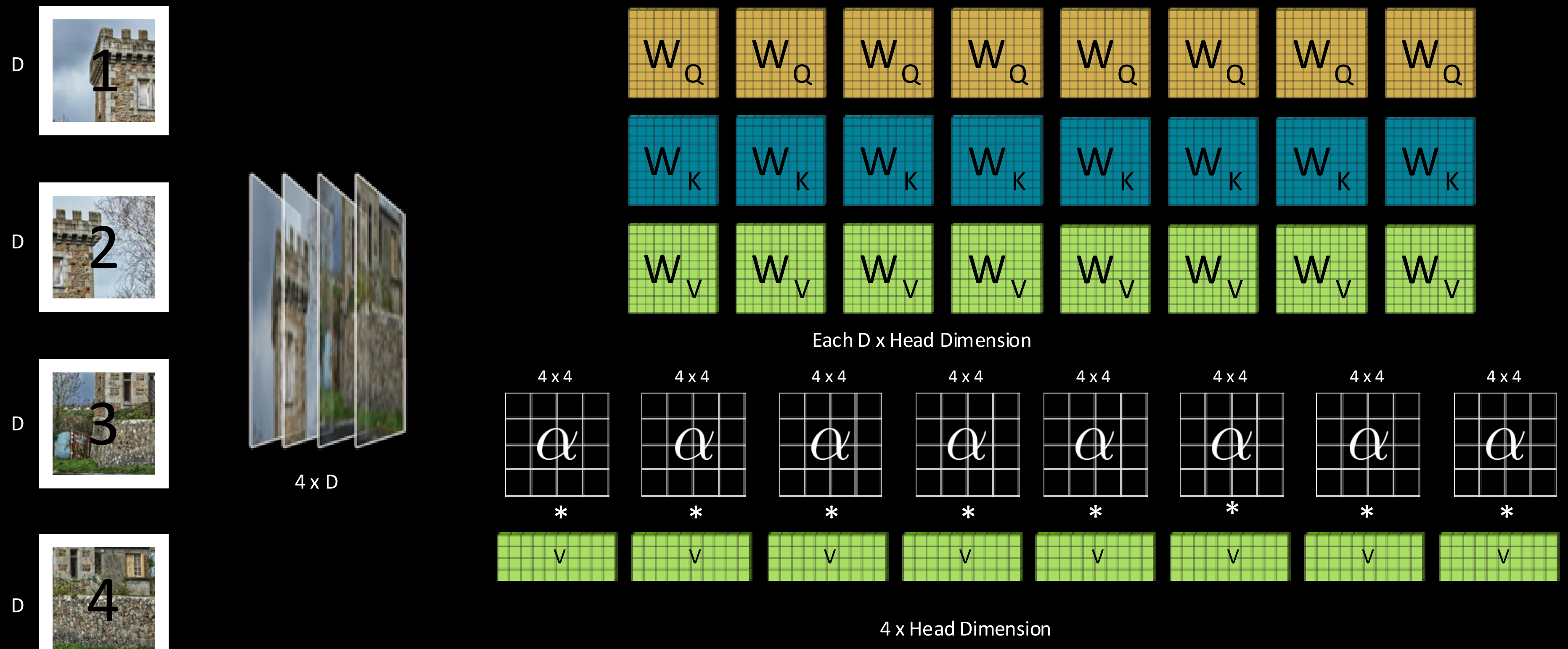
Why ?

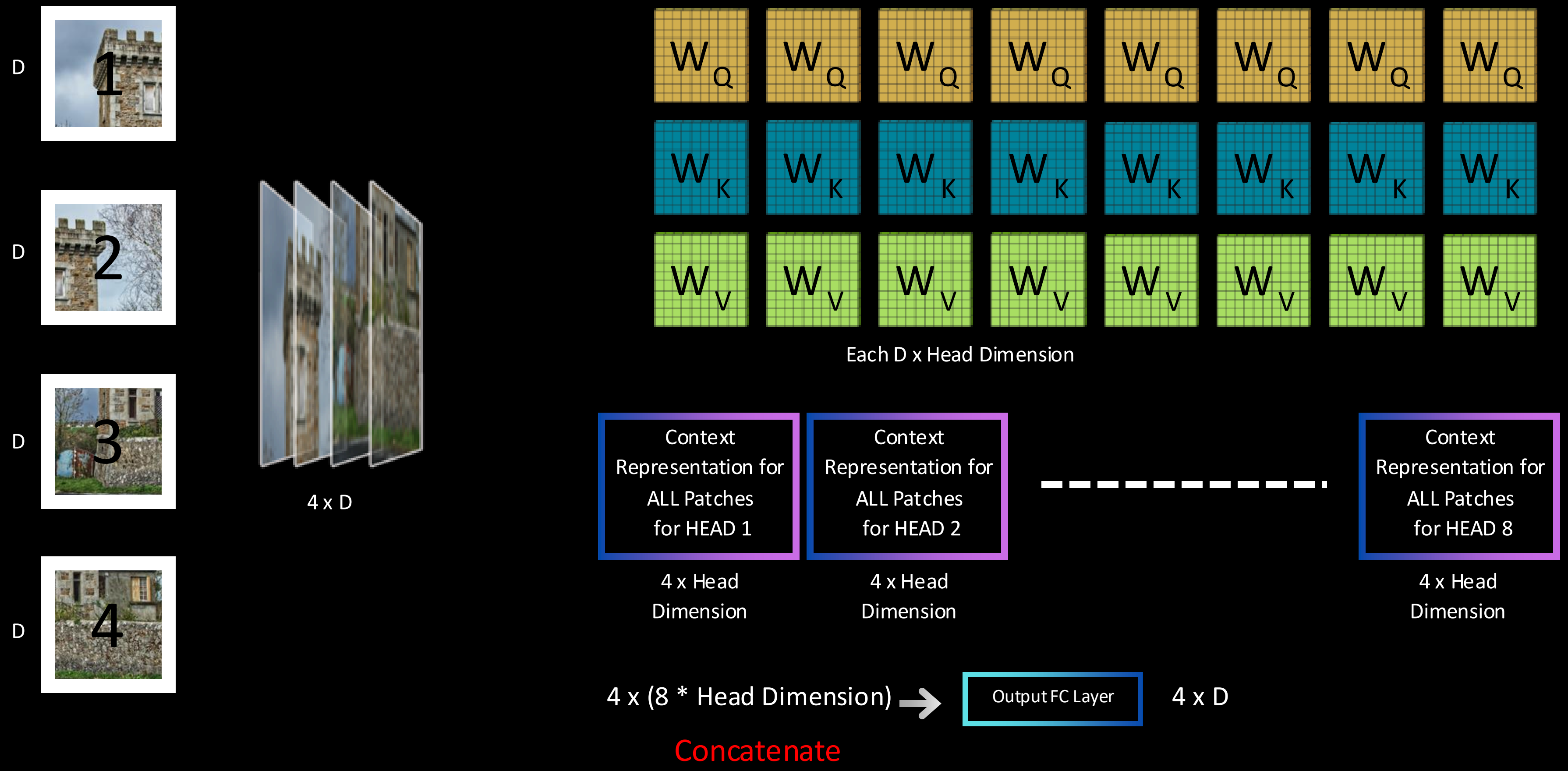


There can be multiple factors of relevance

Each head has unique weight matrices, despite the uniform color representation




D
1
D
2
D
3
D
4
4 x D
W_Q W_Q W_Q W_Q W_Q W_Q W_Q W_Q
W_K W_K W_K W_K W_K W_K W_K W_K
W_V W_V W_V W_V W_V W_V W_V W_V
Each D x Head Dimension
4 x 4 4 x 4 4 x 4 4 x 4 4 x 4 4 x 4 4 x 4 4 x 4
α α α α α α α α
* * * * * * * *
V V V V V V V V
4 x Head Dimension



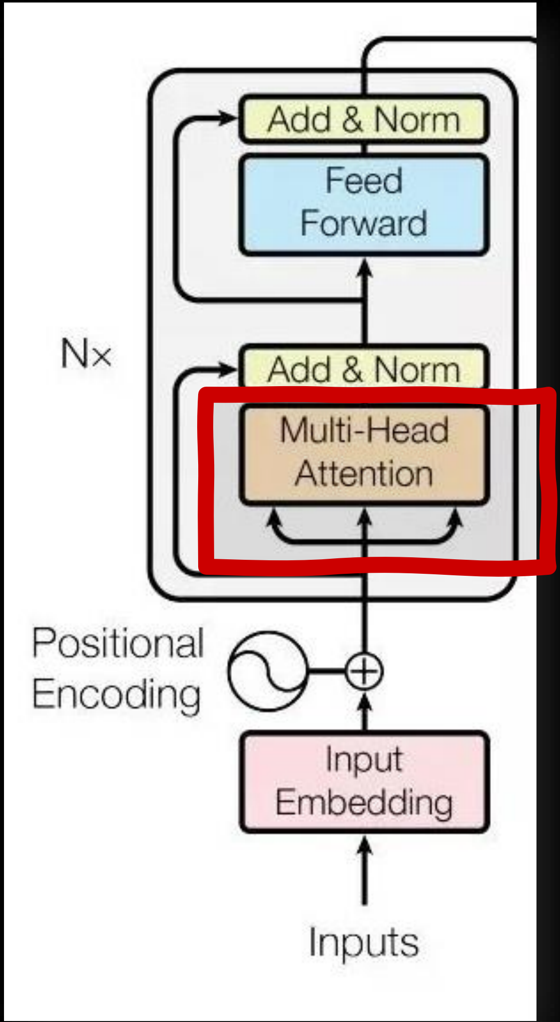
D
D
D
D
1
2
3
4
4 x D
W_Q W_Q W_Q W_Q W_Q W_Q W_Q W_Q
W_K W_K W_K W_K W_K W_K W_K W_K
W_V W_V W_V W_V W_V W_V W_V W_V
Each D x Head Dimension
Context Representation for ALL Patches for HEAD 1
4 x Head Dimension
Context Representation for ALL Patches for HEAD 2
4 x Head Dimension
Context Representation for ALL Patches for HEAD 8
4 x Head Dimension
4 x (8 * Head Dimension)
Output FC Layer
4 x D
Concatenate

Self Attention Block

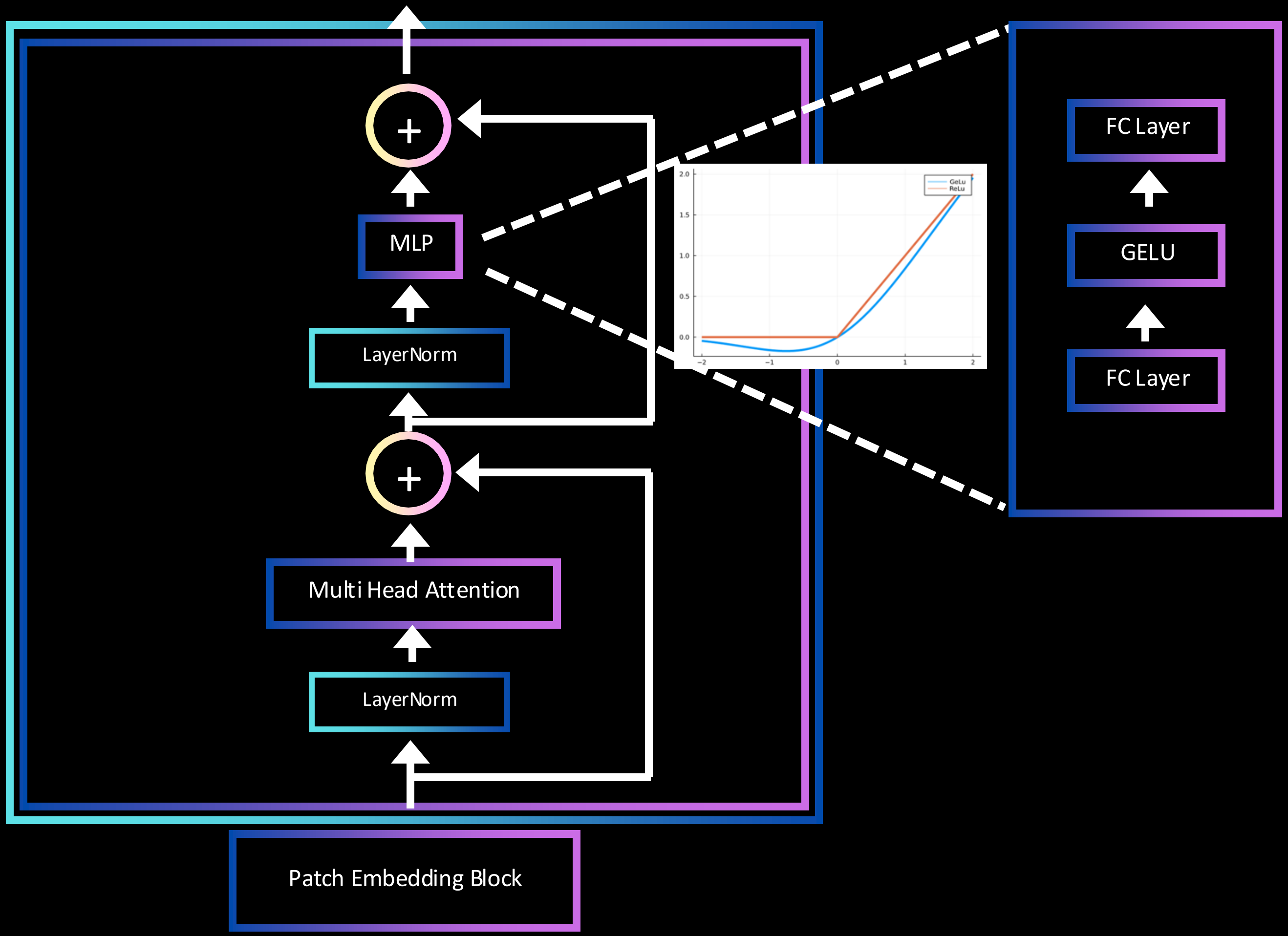
Project from D
to get QKV for each head

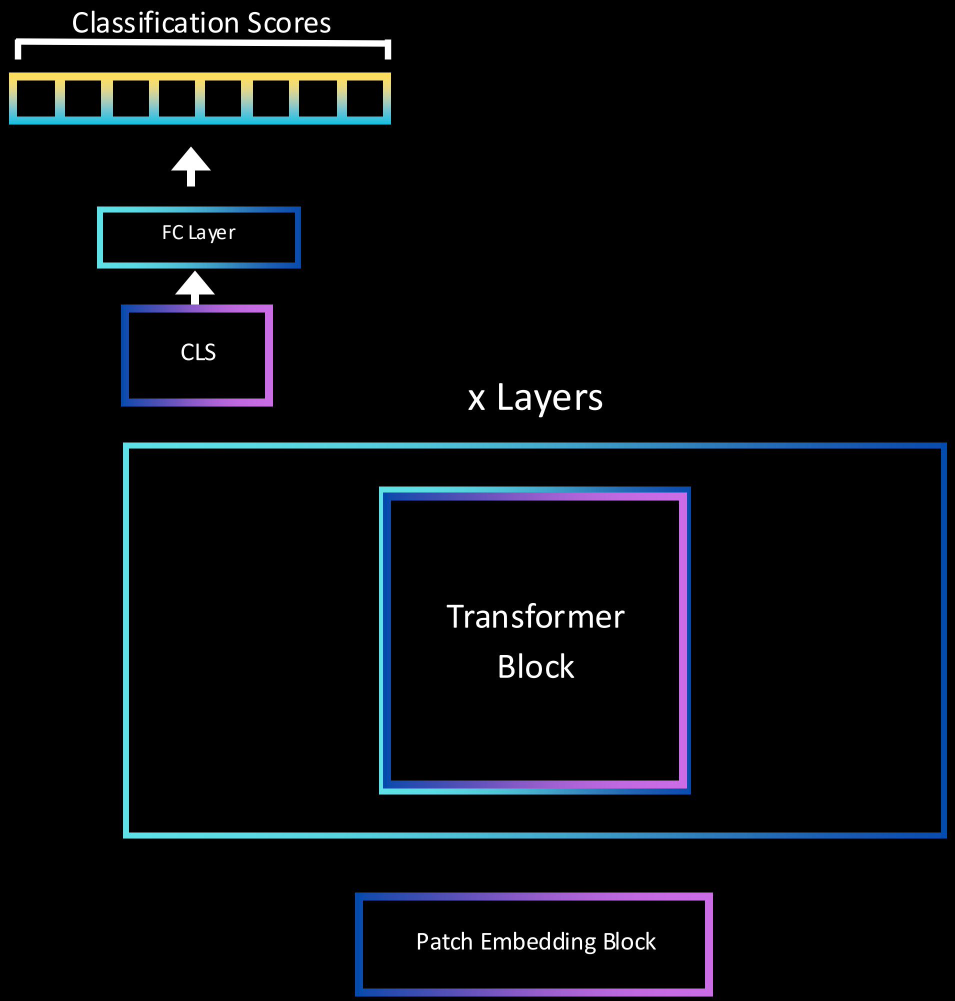
Compute Scaled Dot
Product Attention

Compute Weighted Value,
Concatenation of heads,
Project back to D



Transformer Block




Classification Scores
FC Layer
CLS
x Layers
Transformer
Block
Patch Embedding Block

ViT

CNN

CNNs	ViT
Strong inductive bias (translation invariance)	No strong inductive bias
Not too data-hungry	Data-hungry
Hierarchical structure (receptive field)	Global structure (attention)

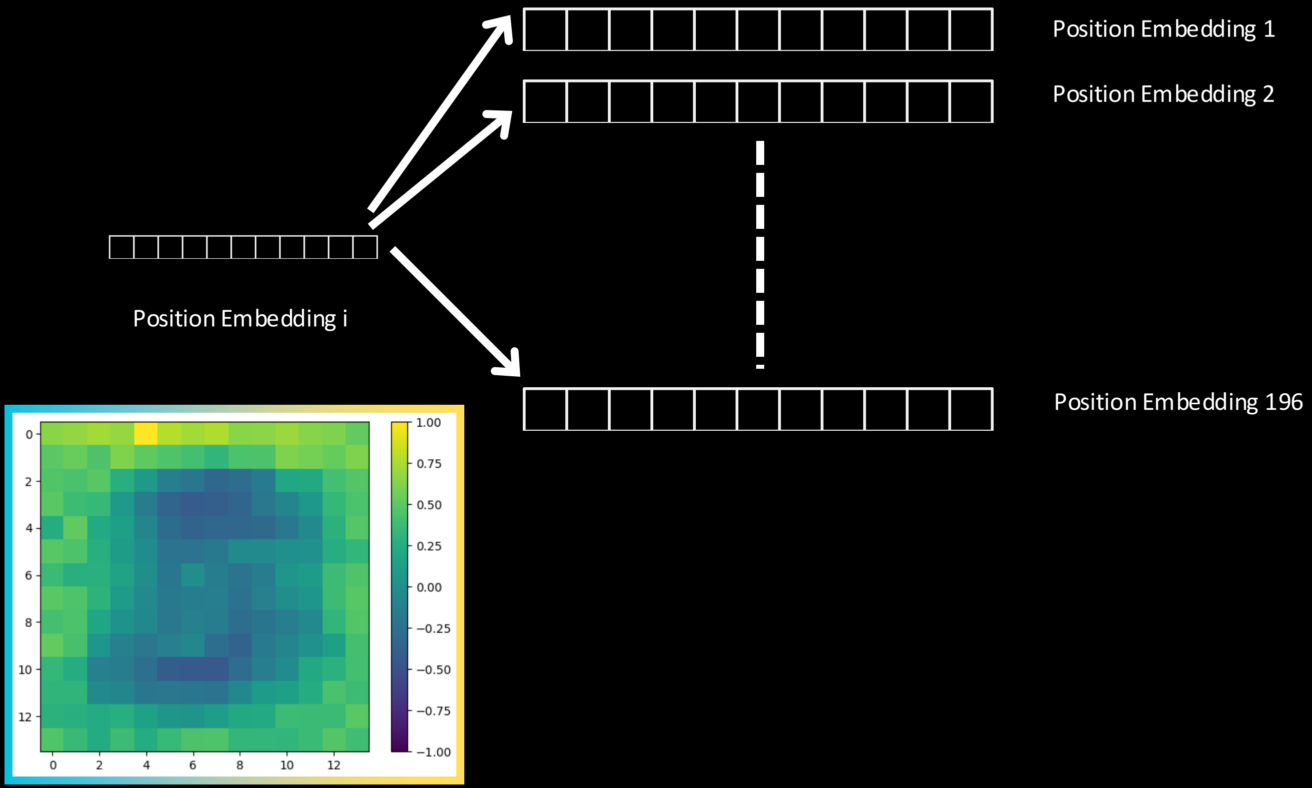
Attention Map Visualization

Source: Exploring Explainability for Vision Transformers
(Jacob Gildenblat)

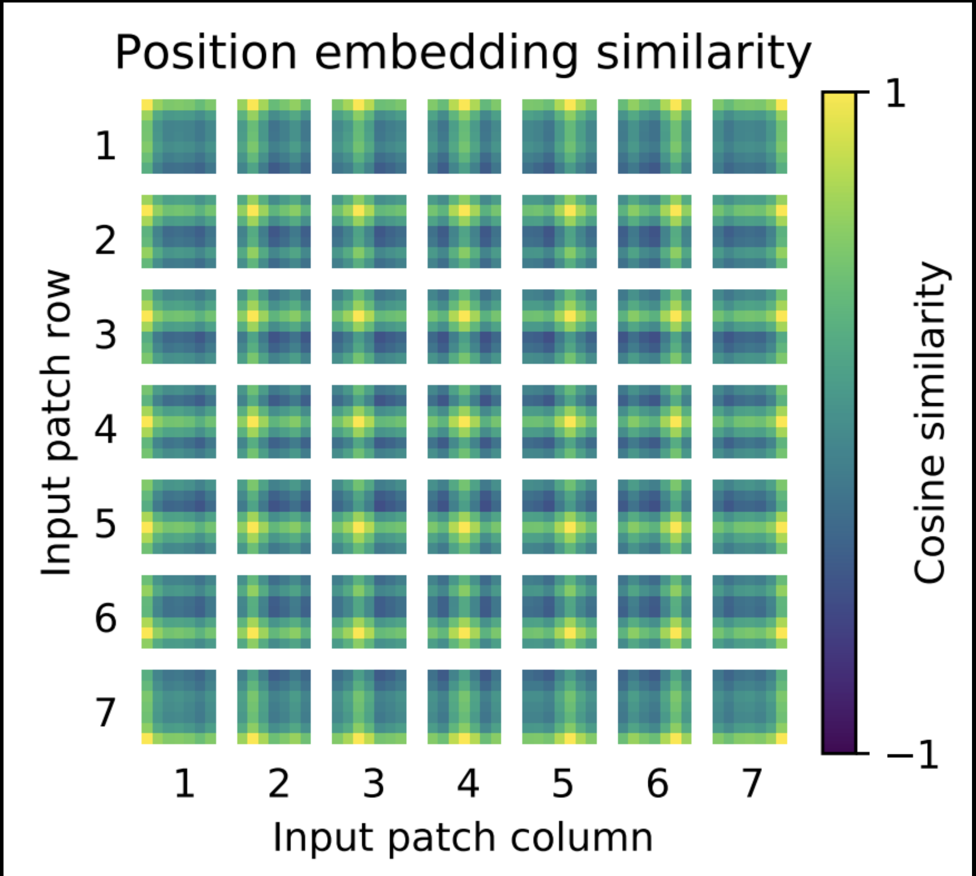
2020

Quantifying Attention Flow in Transformers

Position Embedding Visualization



Position Embedding Visualization



References

Books & Core Reading

- **Goodfellow, I., Bengio, Y., & Courville, A. (2016).** *Deep Learning.* MIT Press. → Chapter 6: Deep Feedforward Networks

Courses & Tutorials

- **Alexander Amini.** *Introduction to Deep Learning,* MIT 6.S191
- **Ali Farhadi.** *Introduction to Deep Learning,* CSE 490G1/599G1
- **Daniel Cremers.** *Introduction to Deep Learning,* IN2346
- **Sergey Levine.** *Designing, Visualizing, and Understanding Deep Neural Networks,* UC Berkeley, CS W182/282A

References

Key Papers

- **He, K., Zhang, X., Ren, S., & Sun, J. (2016).** *Deep Residual Learning for Image Recognition.* CVPR.→ Introduces **ResNet**
- **Ioffe, S., & Szegedy, C. (2015).** *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* ICML.→ BatchNorm
- **Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016).** *You Only Look Once: Unified, Real-Time Object Detection.* CVPR.→ Original **YOLO** paper
- **Dosovitskiy, A., et al. (2021).** *An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale.* ICLR.→ Vision Transformers (ViT)
- **Vaswani, A., et al. (2017).** *Attention Is All You Need.* NeurIPS.→ Foundational Transformer

References

Additional Useful Links

- **CS25: Transformers United V4** (YouTube)
  → https://www.youtube.com/playlist?list=PLoROMvodv4rNiJRchCzutFw5ItR_Z27CM
- **CS231n: Convolutional Neural Networks for Visual Recognition**
  → https://cs231n.github.io/convolutional-networks/
  https://cs231n.github.io/understanding-cnn/
- **The Illustrated Transformer** by Jay Alammar
  → https://jalammar.github.io/illustrated-transformer/

Data Augmentation Libraries:

TorchVision, Kornia, Albumentations
→ Check their official documentation and GitHub repositories