Machine Learning

CMPT 726

Mo Chen
SFU School of Computing Science

2022-11-03

Non-Convexity

As long as the MLP has one or more hidden layer, the following expression is the gradient w.r.t. first layer weights.

$$\frac{\partial L}{\partial \vec{w}_{0,j,\cdot}^{\top}} = \begin{pmatrix} \vec{0} & \cdots & \vec{0} & \vec{x} & \vec{0} & \cdots & \vec{0} \end{pmatrix} \prod_{l=1}^{L} \left[\begin{pmatrix} g'(z_{l,1}) & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & g'(z_{l,n_l-1}) & 0 \end{pmatrix} W_l^{\top} \right] \frac{\partial L}{\partial \hat{\vec{y}}}$$

Consider setting all weight matrices (consisting of both weights and biases) to zero:

$$\frac{\partial L}{\partial \vec{w}_{0,j,\cdot}^{\top}} = \vec{0}, \qquad \forall j$$

Similarly, for any $1 \leq l \leq L$, $\frac{\partial L}{\partial \vec{w}_{l,j,\cdot}^{\top}} = \vec{0}, \ \forall j$

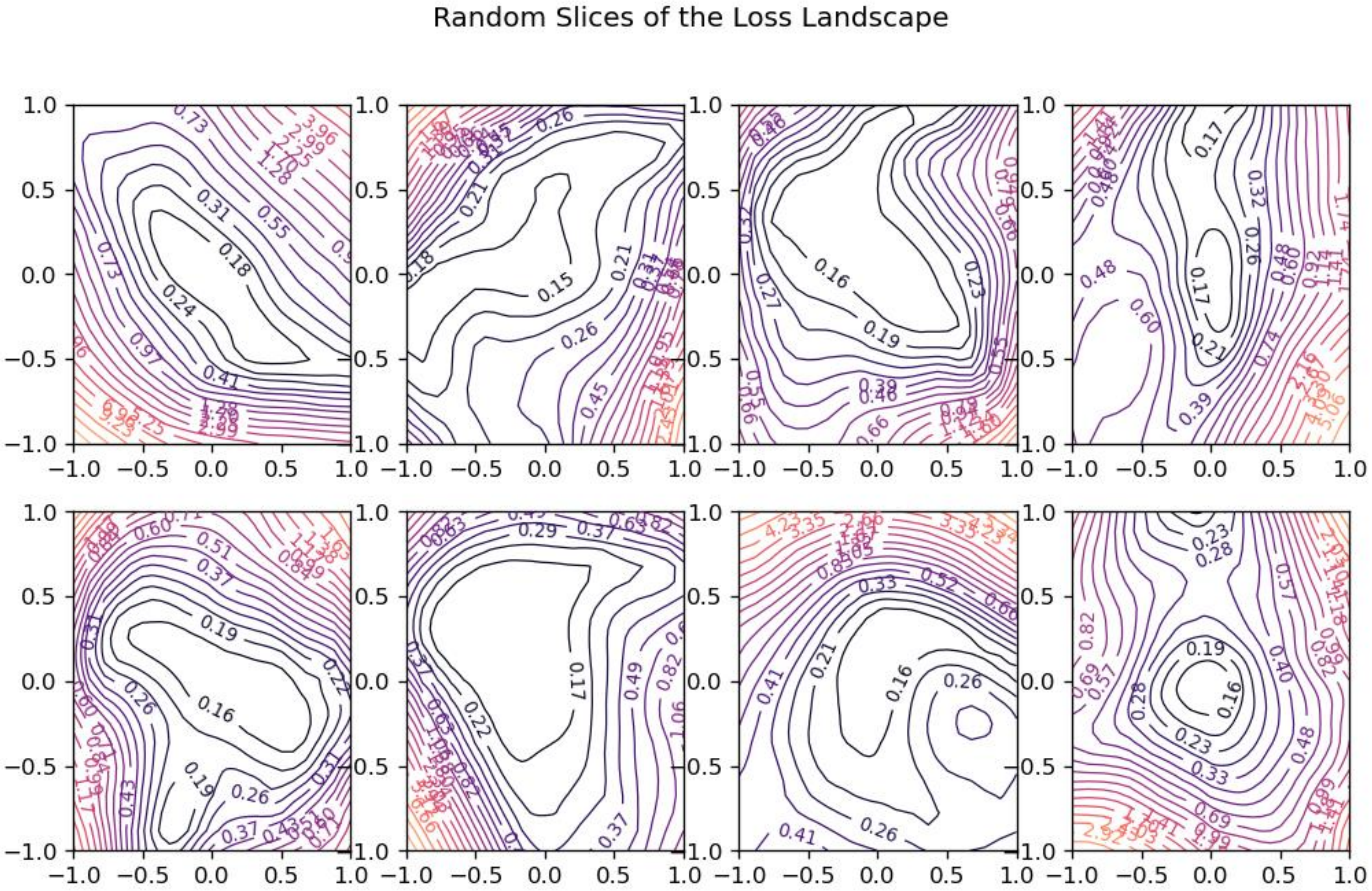
Non-Convexity

All-zero weights and biases is therefore always a critical point, regardless of the training data and the loss function.

In general, it is not global minimum. For convex functions, all critical points must be global minima.

This implies that the objective function for training a neural network cannot be convex.

Gradient descent can therefore get stuck in a local minimum.



Random Slices of the Loss Landscape

Credit: Ryan Holbrook

Residual Connections

Recall:

$$\left\|\frac{\partial L}{\partial \vec{w}_{0,j,\cdot}^{\top}}\right\|_2 \leq \sigma_{1,1}\left(\begin{pmatrix}\vec{0} & \cdots & \vec{0} & \vec{x} & \vec{0} & \cdots & \vec{0}\end{pmatrix}\right)\prod_{l=1}^{L}\left[\sigma_{1,1}\left(\begin{pmatrix} g'(z_{l,1}) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & g'(z_{l,n_l})\end{pmatrix}\right)\sigma_{1,1}\left(W_l^{\top}\right)\right]\left\|\frac{\partial L}{\partial \vec{\hat{y}}}\right\|_2$$

When the activation function does not saturate on both sides, the largest singular value $\sigma_{1,1}\left(\begin{pmatrix} g'(z_{l,1}) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & g'(z_{l,n_l})\end{pmatrix}\right)$ is typically large.

However, $\sigma_{1,1}\left(W_l^{\top}\right)$ can still be small, especially when weight decay is applied.

If $\sigma_{1,1}\left(W_l^{\top}\right) < 1\ \forall l$, gradient magnitude decays exponentially in the number of layers. So gradients can still vanish in a deep neural network even when activation functions are chosen appropriately. So, deep neural networks with ReLU activations can still be difficult to train.

Residual Connections

Can we make deep neural networks as easy to train as shallow neural networks?

Recall the form of an MLP:

$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \ \psi(W_{L-1}\vec{h}_{L-1})$ and $\vec{h}_{L-1}$ $= \psi\left(W_{L-2}\vec{h}_{L-2}\right), \dots,$ and $\vec{h}_1 = \psi(W_0\vec{x})$.

Suppose $\dim(\vec{h}_L) \geq \dim(\vec{h}_{L-1}) \geq \cdots \geq \dim(\vec{h}_1)$ and the activation function is ReLU, a shallow neural network is a special case of a deep neural network.

Idea: Embed identity matrices in the weight matrices of some layers and make all other columns zero.

Residual Connections

Example: Three-layer MLP, $\hat{\vec{y}} = W_2 \psi\big(W_1 \psi(W_0 \vec{x})\big)$

We can make it equivalent to a two-layer MLP by setting $W_1 = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$

Then $W_1 \psi(W_0 \vec{x}) = \begin{pmatrix} \max\big(W_0 \vec{x}, \vec{0}\big) \\ \vec{0} \end{pmatrix}$. Hence, $\psi\big(W_1 \psi(W_0 \vec{x})\big) = \begin{pmatrix} \max\big(\max\big(W_0 \vec{x}, \vec{0}\big), \vec{0}\big) \\ \max\big(\vec{0}, \vec{0}\big) \\ 1 \end{pmatrix} = \begin{pmatrix} \max\big(W_0 \vec{x}, \vec{0}\big) \\ \vec{0} \\ 1 \end{pmatrix}$

Let $W_2 = \begin{pmatrix} W_2' & W_2'' & \vec{b}_2 \end{pmatrix}$

$$\hat{\vec{y}} = W_2 \psi\big(W_1 \psi(W_0 \vec{x})\big) = \begin{pmatrix} W_2' & W_2'' & \vec{b}_2 \end{pmatrix} \begin{pmatrix} \max\big(W_0 \vec{x}, \vec{0}\big) \\ \vec{0} \\ 1 \end{pmatrix}$$

$$= W_2' \max\big(W_0 \vec{x}, \vec{0}\big) + \vec{b}_2 = \begin{pmatrix} W_2' & \vec{b}_2 \end{pmatrix} \psi(W_0 \vec{x}) \coloneqq \widetilde{W}_2 \psi(W_0 \vec{x})$$

Residual Connections

Notice that $\sigma_{1,1}(W_1^\mathsf{T}) = 1$, so when the weight matrix in the three-layer MLP is set this way, the gradient won't necessarily vanish.

However, it is difficult to ensure that $\sigma_{1,1}(W_1^\mathsf{T})$ stays close to 1 throughout training. Two options:

(1) Set W_1^T manually and freeze this weight matrix during training (common debugging trick; also has another purpose - more on this later). However, might as well use a shallow neural network.

(2) Reparameterize the neural network so that even when $\sigma_{1,1}(W_1^\mathsf{T})$ is close to zero, gradient wouldn't vanish.

Residual Connections

Consider the following reparameterization:

$$W_1' = W_1 - \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \Rightarrow W_1 = W_1' + \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \coloneqq W_1' + J$$

$$\hat{\hat{y}} = W_2 \psi\big(W_1 \psi(W_0 \vec{x})\big) = W_2 \psi\big((W_1' + J)\psi(W_0 \vec{x})\big) = W_2 \psi\big(W_1' \psi(W_0 \vec{x}) + J\psi(W_0 \vec{x})\big)$$

From two slides ago: $J\psi(W_0 \vec{x}) = \begin{pmatrix} \max\big(W_0 \vec{x}, \vec{0}\big) \\ \vec{0} \end{pmatrix}$.

So $\hat{\hat{y}} = W_2 \psi\left(W_1' \psi(W_0 \vec{x}) + \begin{pmatrix} \max\big(W_0 \vec{x}, \vec{0}\big) \\ \vec{0} \end{pmatrix}\right) = \begin{pmatrix} W_2' & W_2'' & \vec{b}_2 \end{pmatrix} \psi\left(W_1' \psi(W_0 \vec{x}) + \begin{pmatrix} \max\big(W_0 \vec{x}, \vec{0}\big) \\ \vec{0} \end{pmatrix}\right)$

When W_1' is the all-zero matrix, $\hat{\hat{y}} = \begin{pmatrix} W_2' & W_2'' & \vec{b}_2 \end{pmatrix} \psi\left(\vec{0} + \begin{pmatrix} \max\big(W_0 \vec{x}, \vec{0}\big) \\ \vec{0} \end{pmatrix}\right) = \begin{pmatrix} W_2' & \vec{b}_2 \end{pmatrix} \psi(W_0 \vec{x}) = \widetilde{W}_2 \psi(W_0 \vec{x})$

Residual Connections

So when W_1' is the all-zero matrix, the three-layer MLP just becomes equivalent to a two-layer MLP, which can be trained more easily than a three-layer MLP.

Observe that in this case $\sigma_{1,1}(W_1'^{\top}) = 0$, so after reparameterization, we no longer have to worry about $\sigma_{1,1}(W_1'^{\top})$ being small.

To summarize, after reparamerization, the model becomes:

$$\hat{\vec{y}} = W_2\psi\left(W_1'\psi(W_0\vec{x}) + \begin{pmatrix}\max\left(W_0\vec{x},\vec{0}\right)\\ \vec{0}\end{pmatrix}\right)$$

If we allow the post-activations to pass through a layer without being multiplied by the weight matrix and add them directly to the pre-activations of the current layer, we can avoid vanishing gradients.

A layer of this form is said to consist of a residual connection from the post-activations of the previous layer to the pre-activations of the current layer.

Residual Connections

In general, a **residual connection** can take the pre- or post-activations of an earlier layer and add them to the pre- or post-activations of a later layer.

Recall the functional form of an MLP: $\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \psi(\vec{z}_L), \vec{z}_L = W_{L-1}\vec{h}_{L-1}, \ldots, \vec{h}_1 = \psi(\vec{z}_1)$ and $z_1 = W_0 \vec{x}$.

In the previous example, we made $\vec{z}_l = \begin{pmatrix} \vec{h}_{l-1} \\ \vec{0} \end{pmatrix} + W_{l-1}\vec{h}_{l-1}$ instead.

In general, we can make either $\vec{z}_l$ or $\vec{h}_l$ be $\begin{pmatrix} \vec{h}_{l'} \\ \vec{0} \end{pmatrix} + W_{l'}\vec{h}_{l'}$ or $\begin{pmatrix} \vec{z}_{l'} \\ \vec{0} \end{pmatrix} + W_{l'}\vec{h}_{l'}$ where $l' < l$.

A residual connection is a special case of a **skip connection**, which in general skips certain operations (e.g.: matrix multiplication) and directly **concatenates** (rather than adds) the pre- or post-activations of an earlier layer to the pre- or post-activations of a later layer.

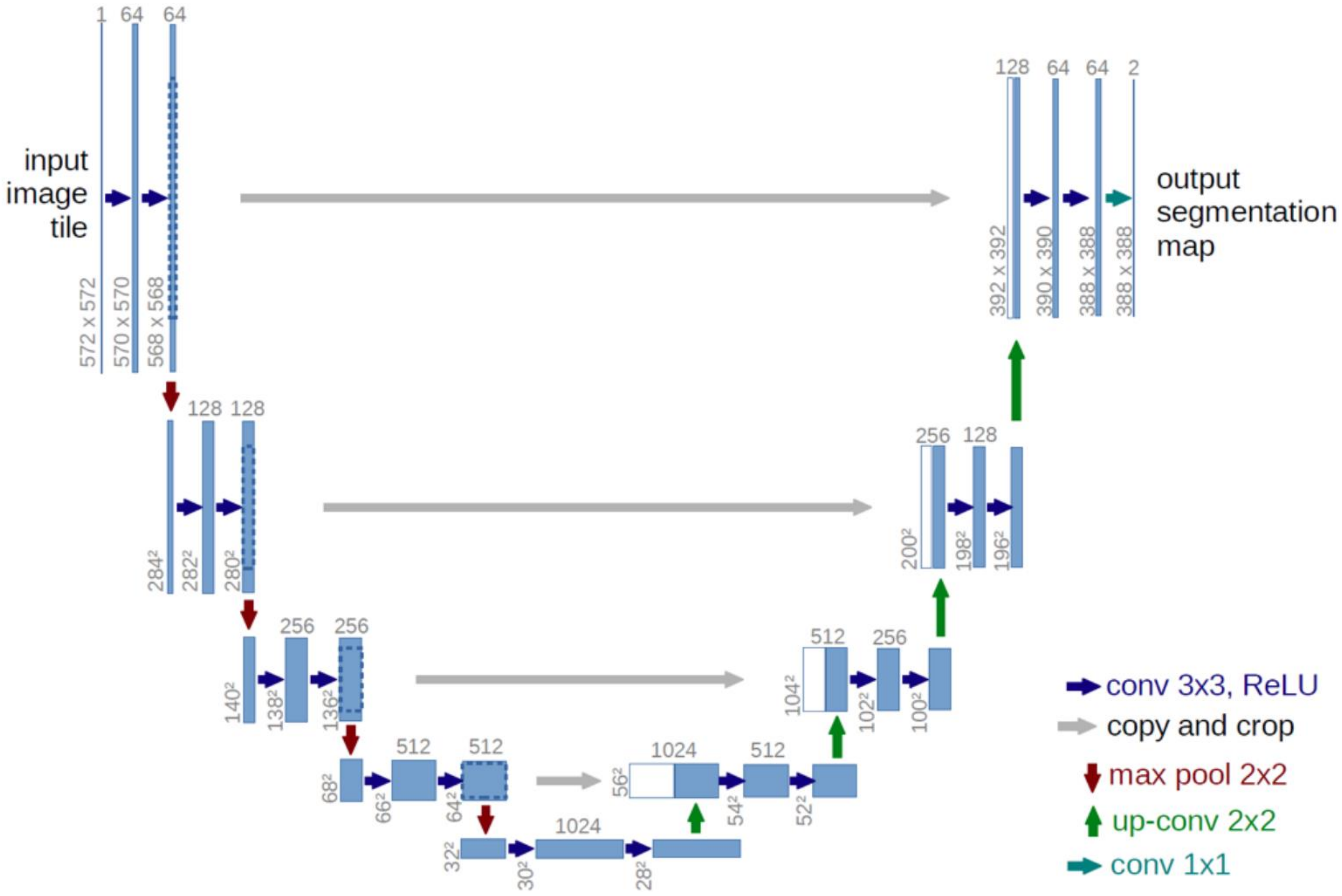
The resulting model is no longer an MLP, and is sometimes known as a **residual network** (though sometimes the terms refers to a particular neural network that is in popular use).

Architecture

The functional form of a neural network is known as an **architecture**.

An architecture defines:

- How many layers there are
- The number of units in a layer
- The activation function
- What mathematical operation each layer performs, e.g.:
    - Multiply the previous layer’s post-activations with a matrix (known as a **fully-connected** layer or **dense** layer)
    - A fully-connected layer with a skip connection from a previous layer



Credit: Olaf Ronneberger, Philipp Fischer and Thomas Brox

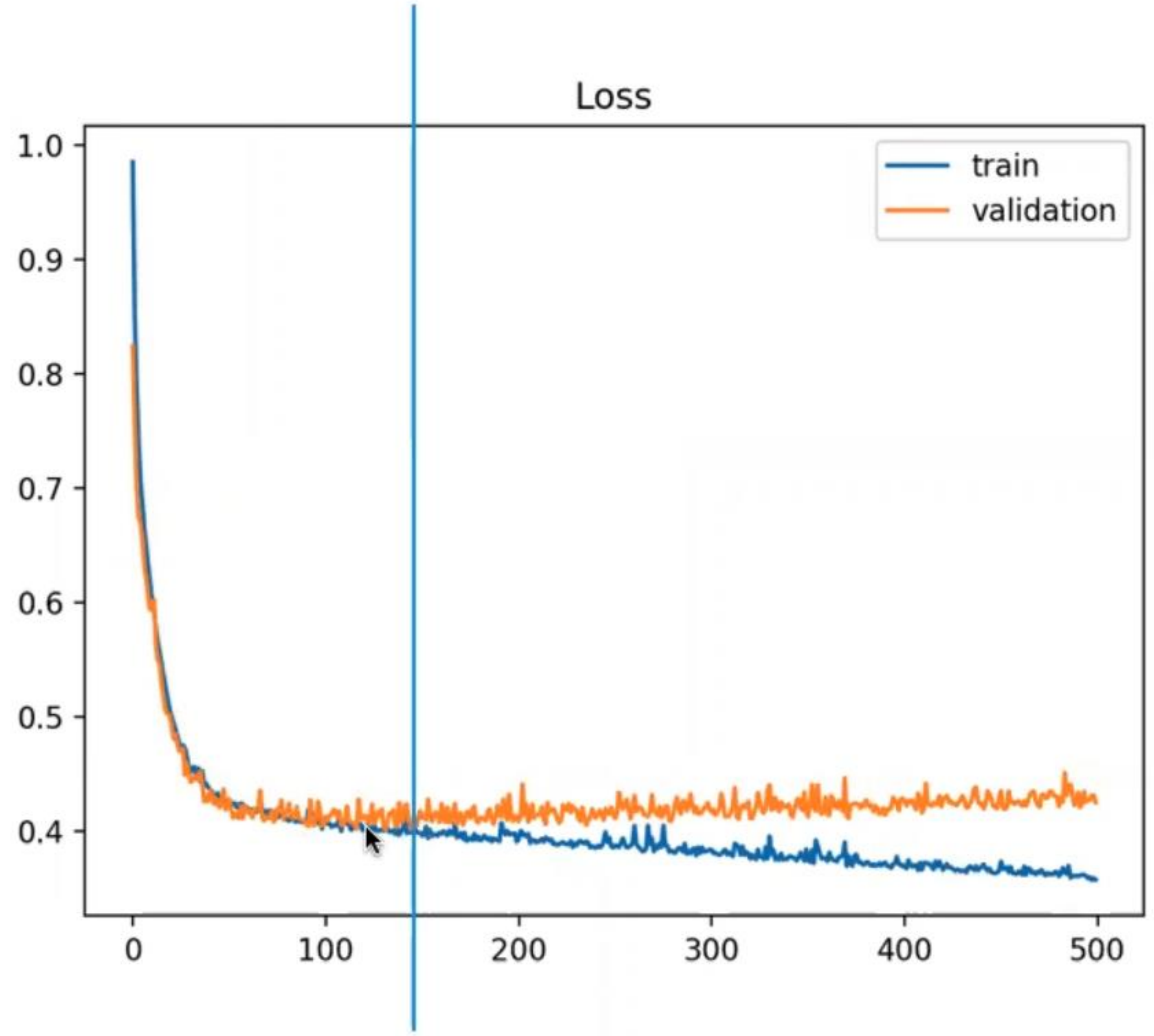
Early Stopping

Recall: Iterative optimization algorithms are terminated when the objective value or the parameter vector does not change much from iteration to iteration.

In a machine learning context, sometimes it is advisable to terminate them earlier to avoid overfitting.

Can keep track of validation loss and terminate when the validation loss starts to increase.

Can stop here



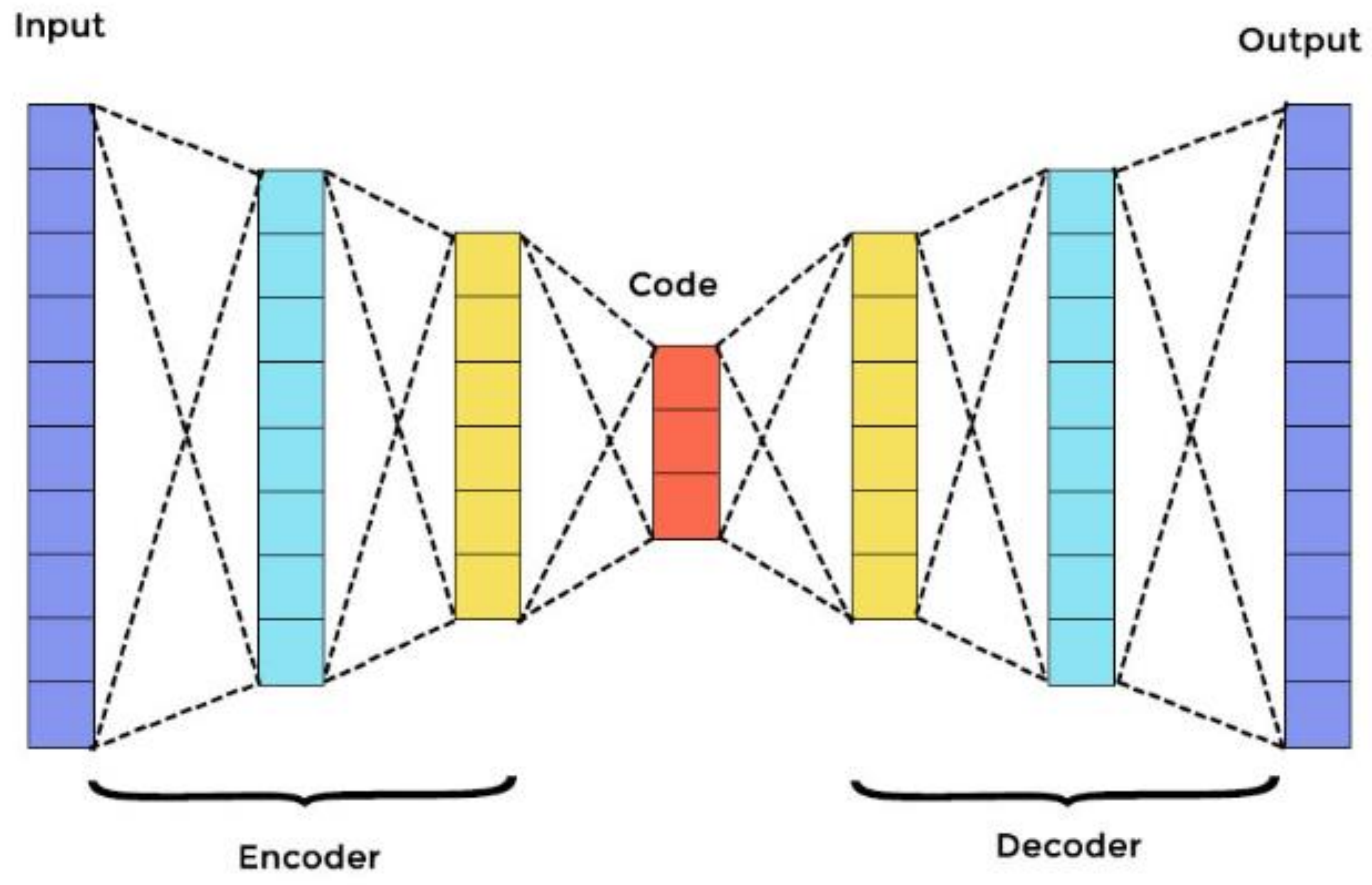
Credit: Jason Brownlee

Application: Autoencoders

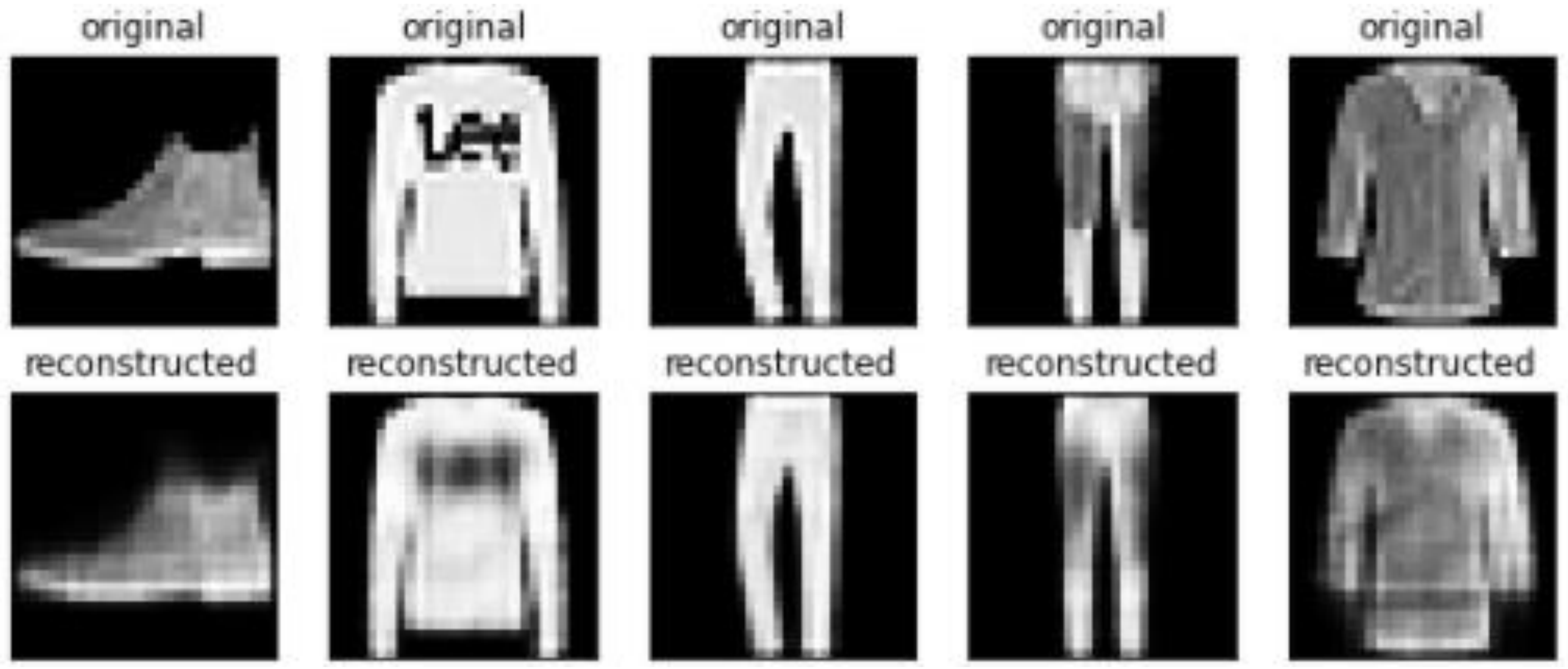
Model: $\hat{\vec{x}} = g\left(\vec{z}; \{W_l\}_{l=0}^{L}\right)$ where $\vec{z} = f\left(\vec{x}; \{W_l\}_{l=L+1}^{2L+2}\right)$, where $f(\cdot)$ and $g(\cdot)$ are neural networks. The dimensionality of $\vec{z}$ is typically smaller than $\vec{x}$.

f is known as an **encoder**, and g is known as a **decoder**. $\vec{z}$ is known as a **code**.

The architectures of the encoder f and the decoder g usually mirror each another. The decoder g sometimes has a sigmoid activation function if the data lies within a fixed range.



Credit: Educba.com



Credit: Tensorflow.org

Application: Autoencoders

Loss function: Same as multi-output linear regression, except that the target label is the same as the input.

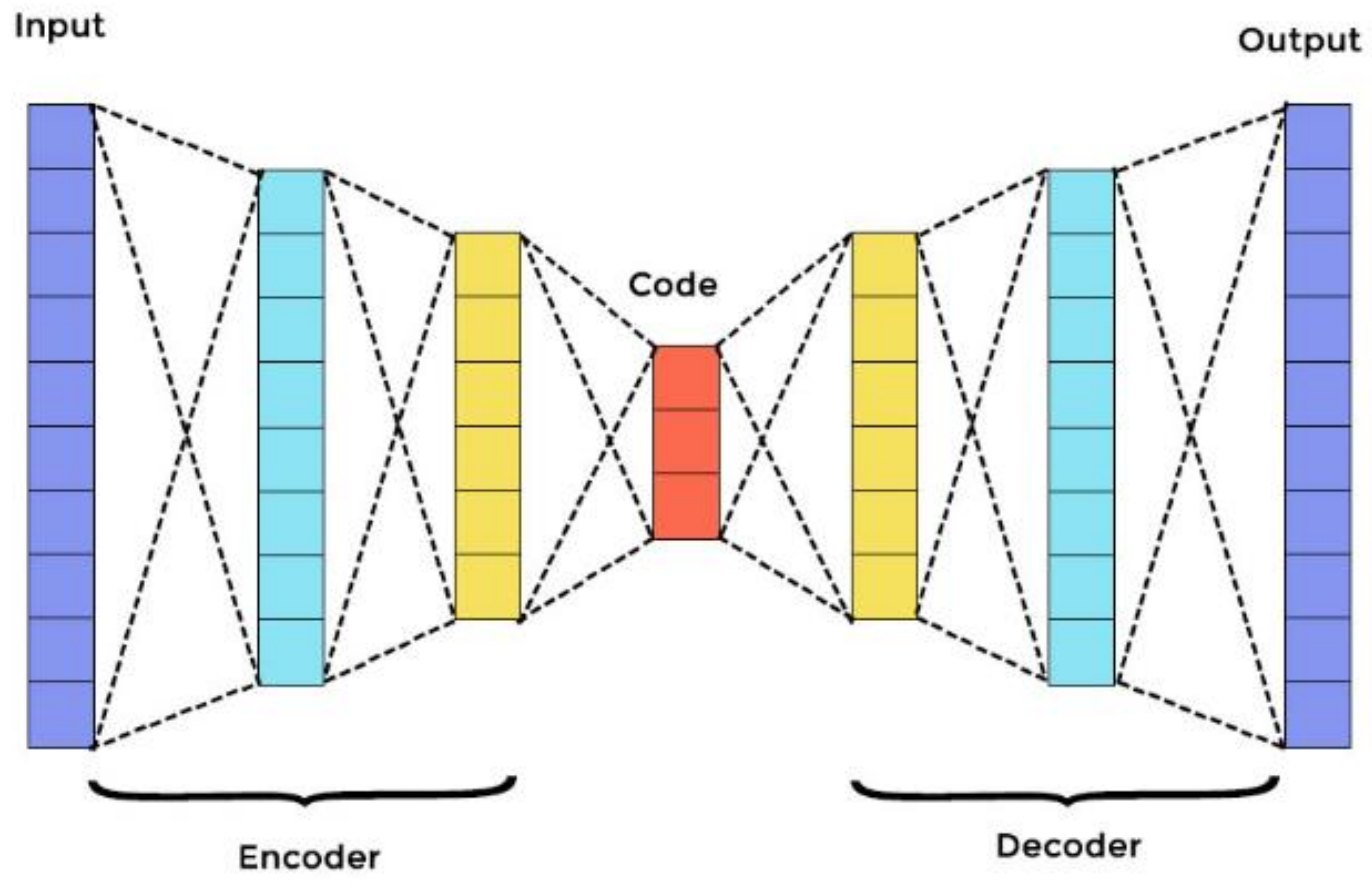
$$L\left(\{W_l\}_{l=0}^{2L+2}\right) = \sum_{i=1}^{N} \left\| \vec{x}_i - \hat{\vec{x}}_i \right\|_2^2 = \sum_{i=1}^{N} \left\| \vec{x}_i - g\left(f\left(\vec{x}_i\right)\right) \right\|_2^2$$

Example of an **unsupervised learning** method, in that labels are not required. In contrast, methods that require labels, are known as a **supervised learning** method.

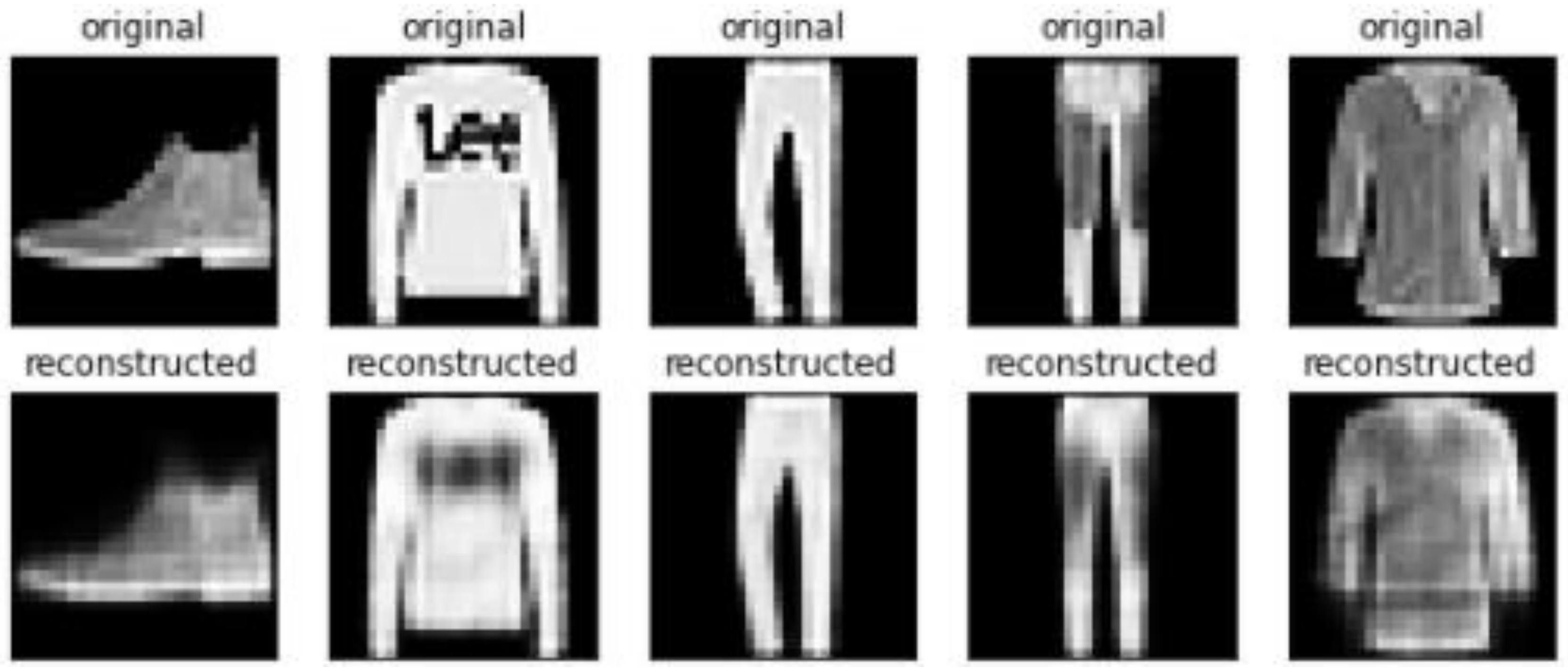
Used to:

(1) Compress data

(2) Visualize high-dimensional data in 2D



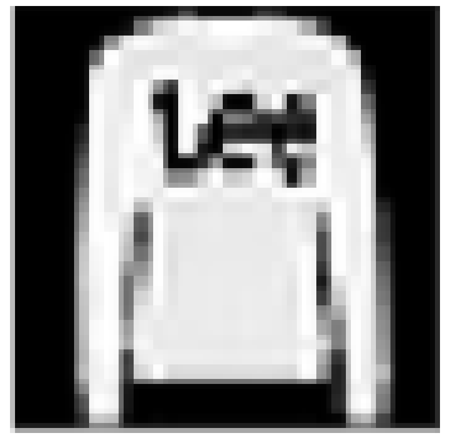
Credit: Educba.com



Credit: Tensorflow.org

Data Representation

$\vec{x}$



$\mathbb{R}^{h \times w}$

$$\begin{pmatrix} 12 & 0 & 129 \\ 212 & 182 & 247 \\ 38 & 255 & 81 \end{pmatrix}$$

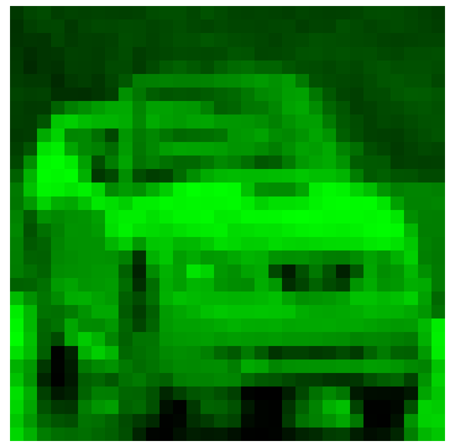
$\mathbb{R}^{hw}$

$$\begin{pmatrix} 12 \\ 0 \\ 129 \\ 212 \\ 182 \\ 247 \\ 38 \\ 255 \\ 81 \end{pmatrix}$$

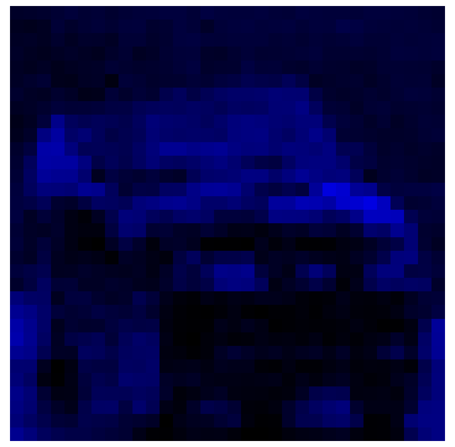
Data Representation

$\vec{x}$

Red



Green



Blue

$$\mathbb{R}^{h\times w\times 3}$$

$$\begin{pmatrix} 38 & 248 & 50 \\ 67 & 10 & 219 \\ 171 & 93 & 183 \end{pmatrix}$$

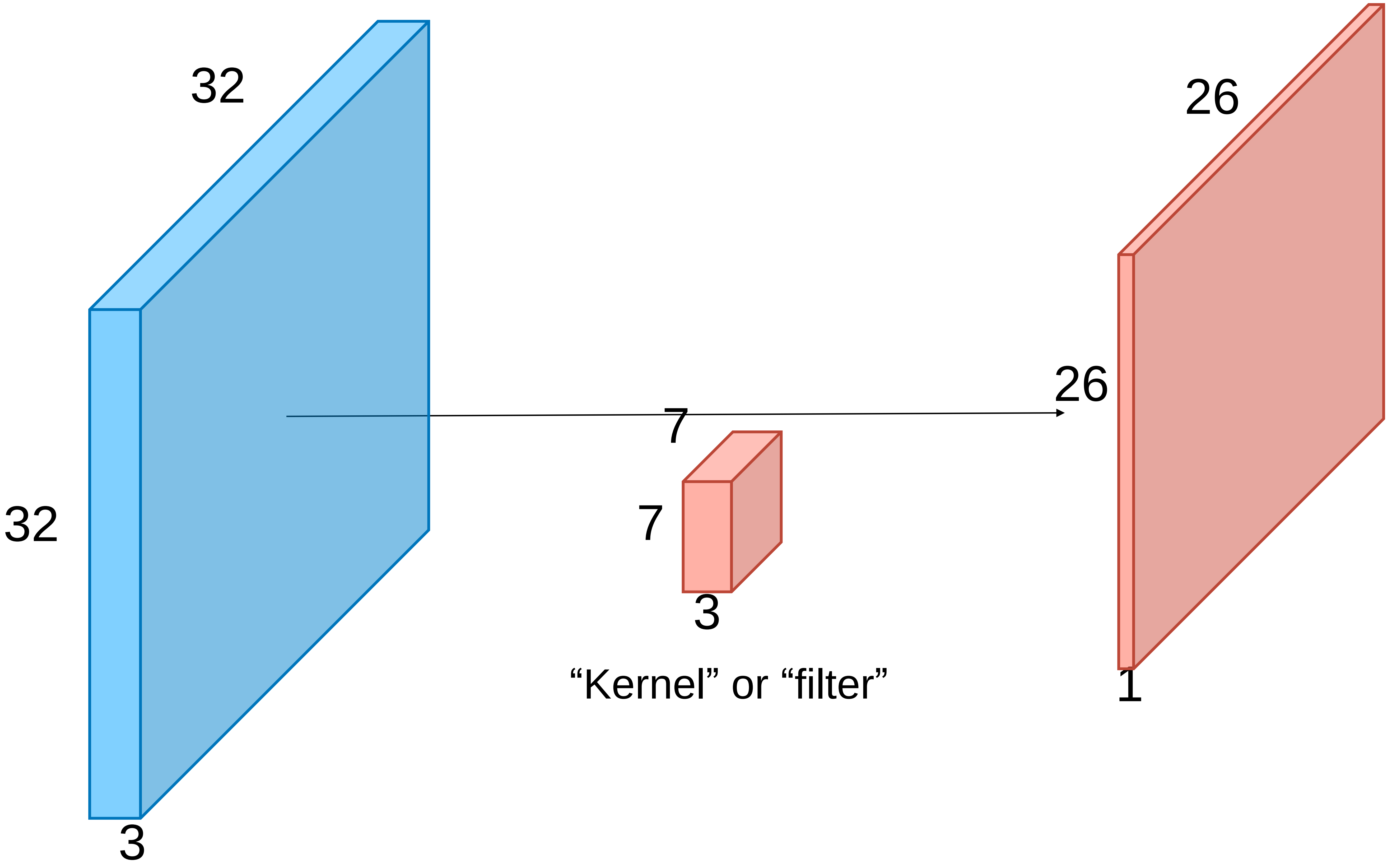
$$\begin{pmatrix} 104 & 49 & 241 \\ 53 & 201 & 17 \\ 162 & 81 & 130 \end{pmatrix}$$

$$\begin{pmatrix} 235 & 56 & 162 \\ 159 & 28 & 68 \\ 72 & 128 & 113 \end{pmatrix}$$

$$\mathbb{R}^{3hw}$$

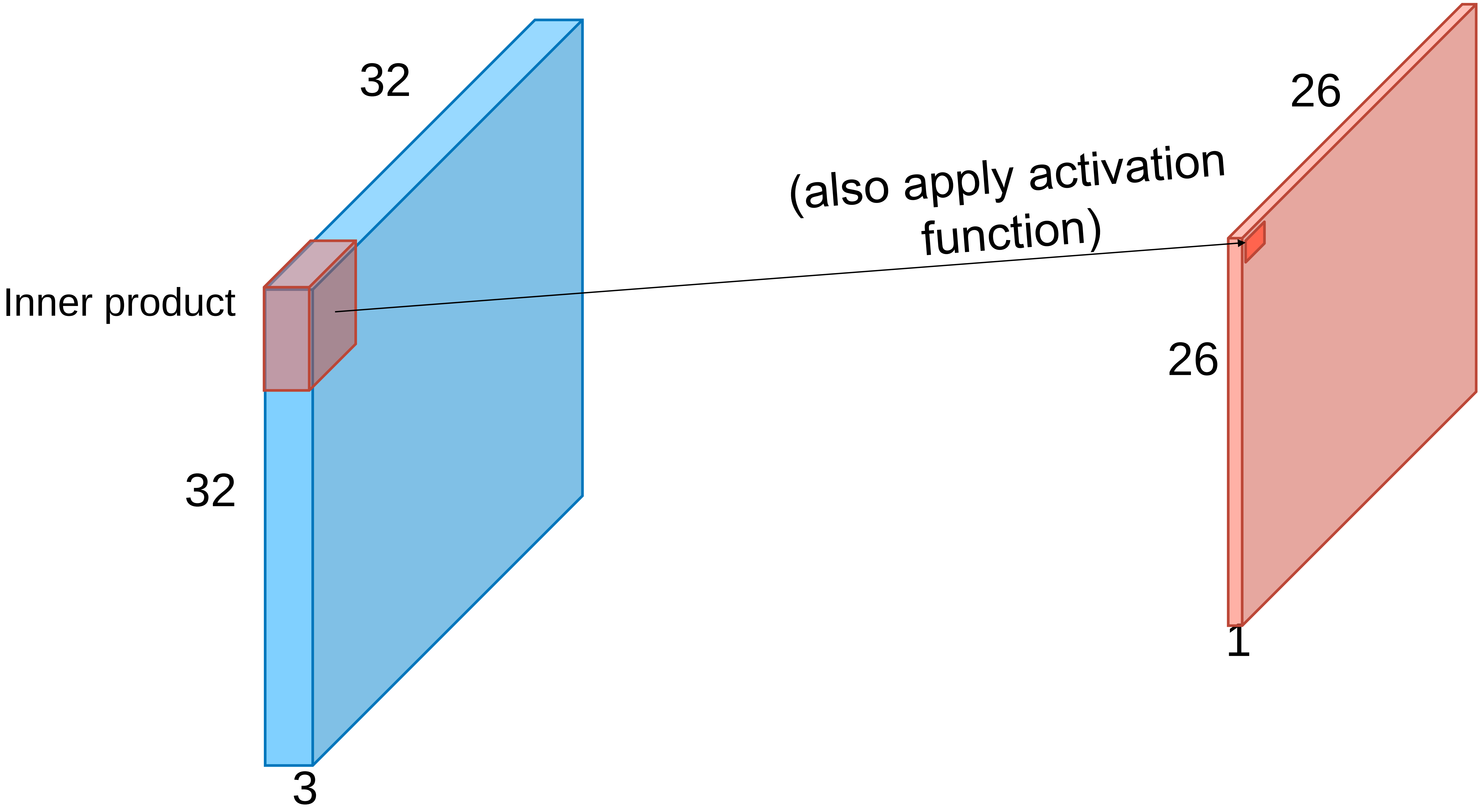
$$\begin{pmatrix} 38 \\ 248 \\ 50 \\ \vdots \\ 72 \\ 128 \\ 113 \end{pmatrix}$$

Convolutional Layers

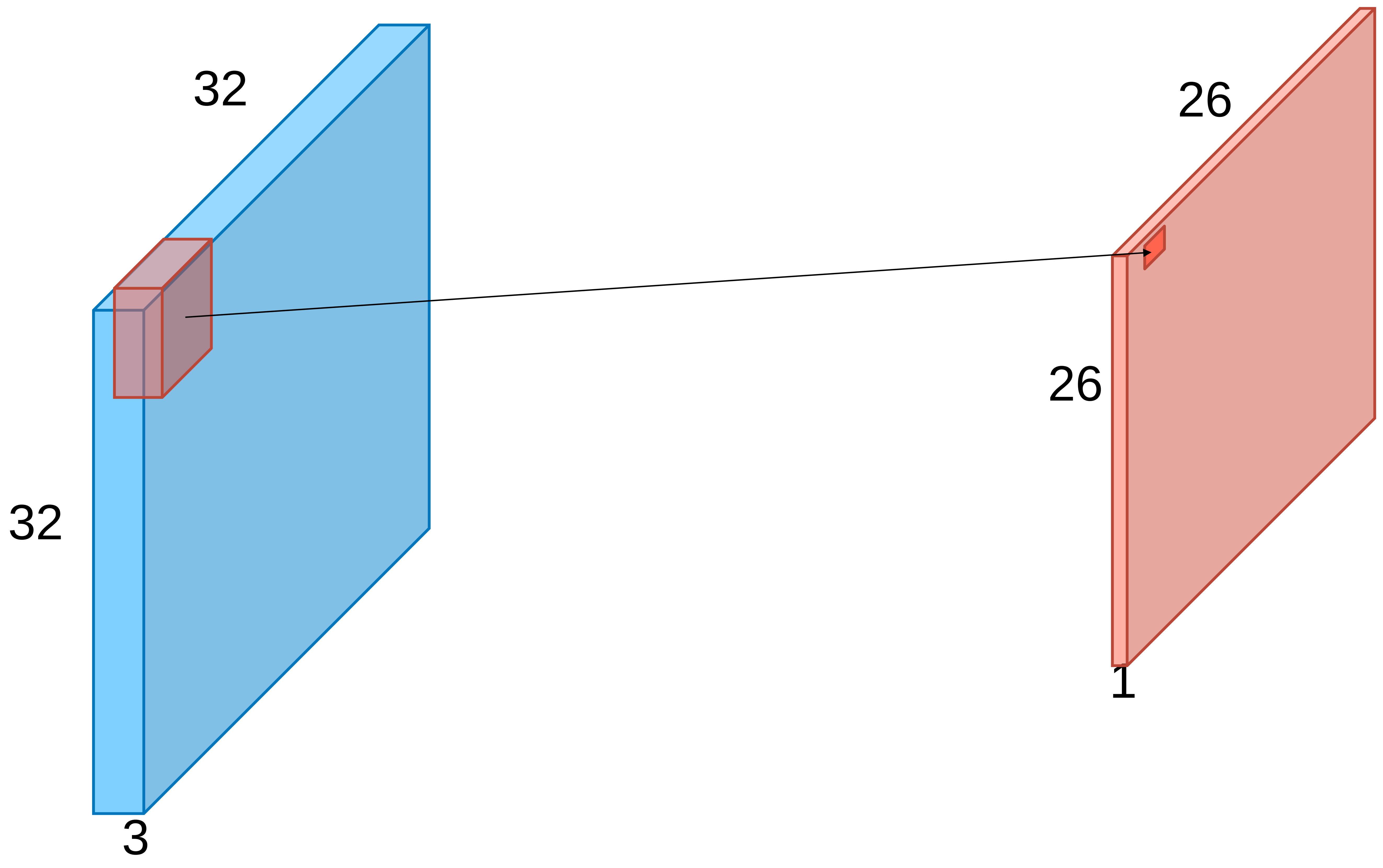


“Kernel” or “filter”

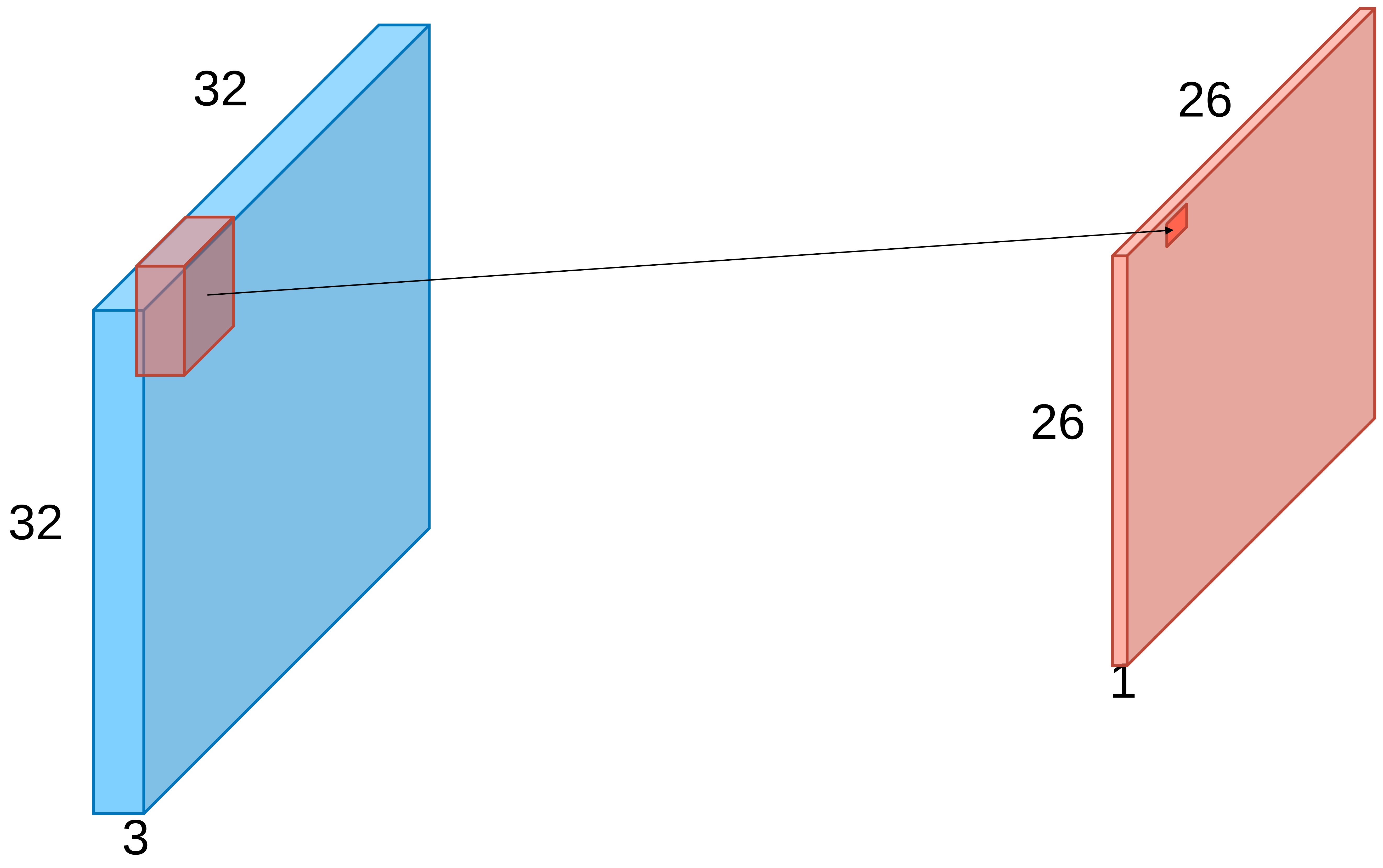
Convolutional Layers



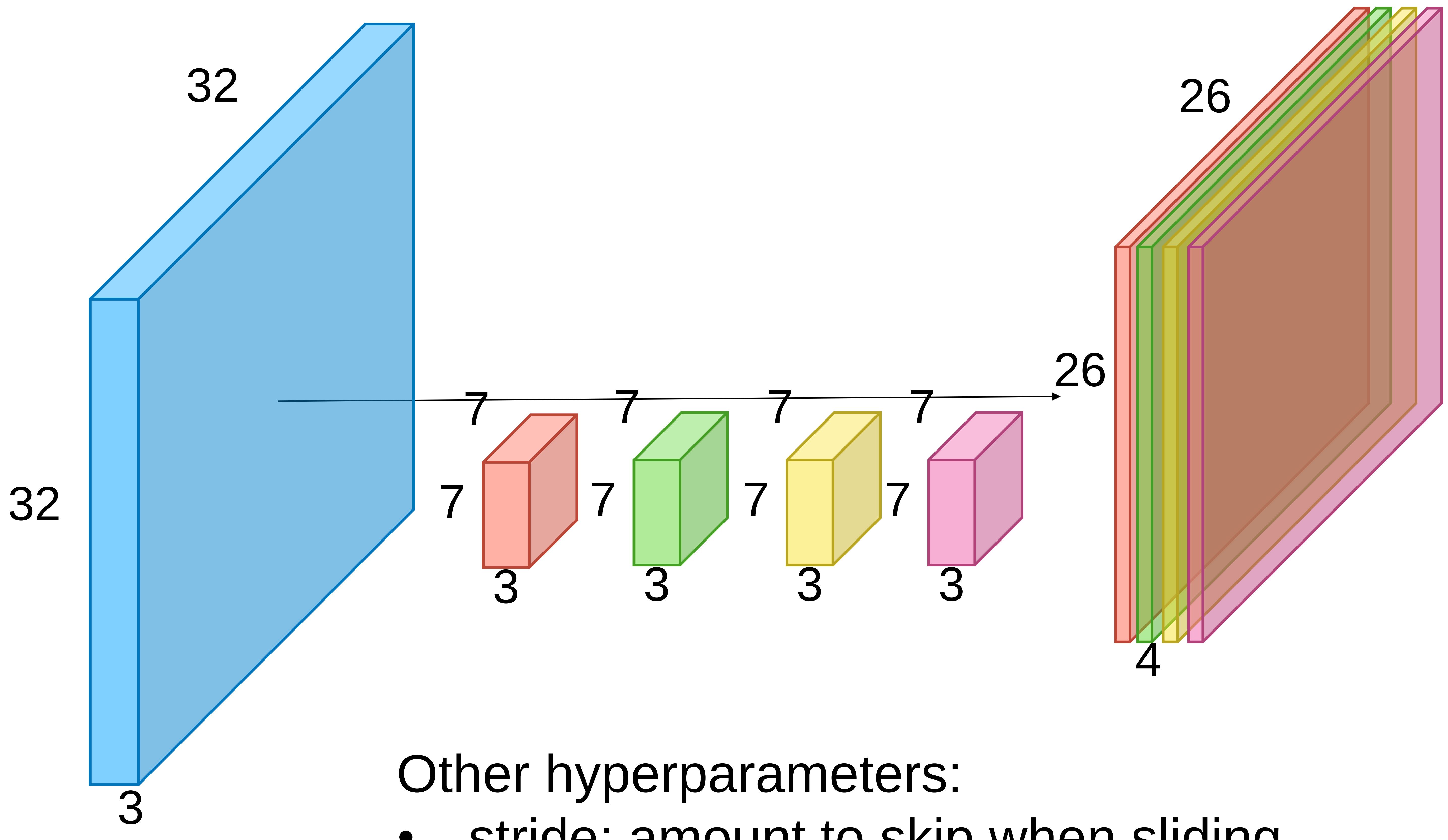
Convolutional Layers



Convolutional Layers



Convolutional Layers



Other hyperparameters:

- stride: amount to skip when sliding
- zero-padding: pad previous layer with zeros
- dilation: skip over pixels of previous layer

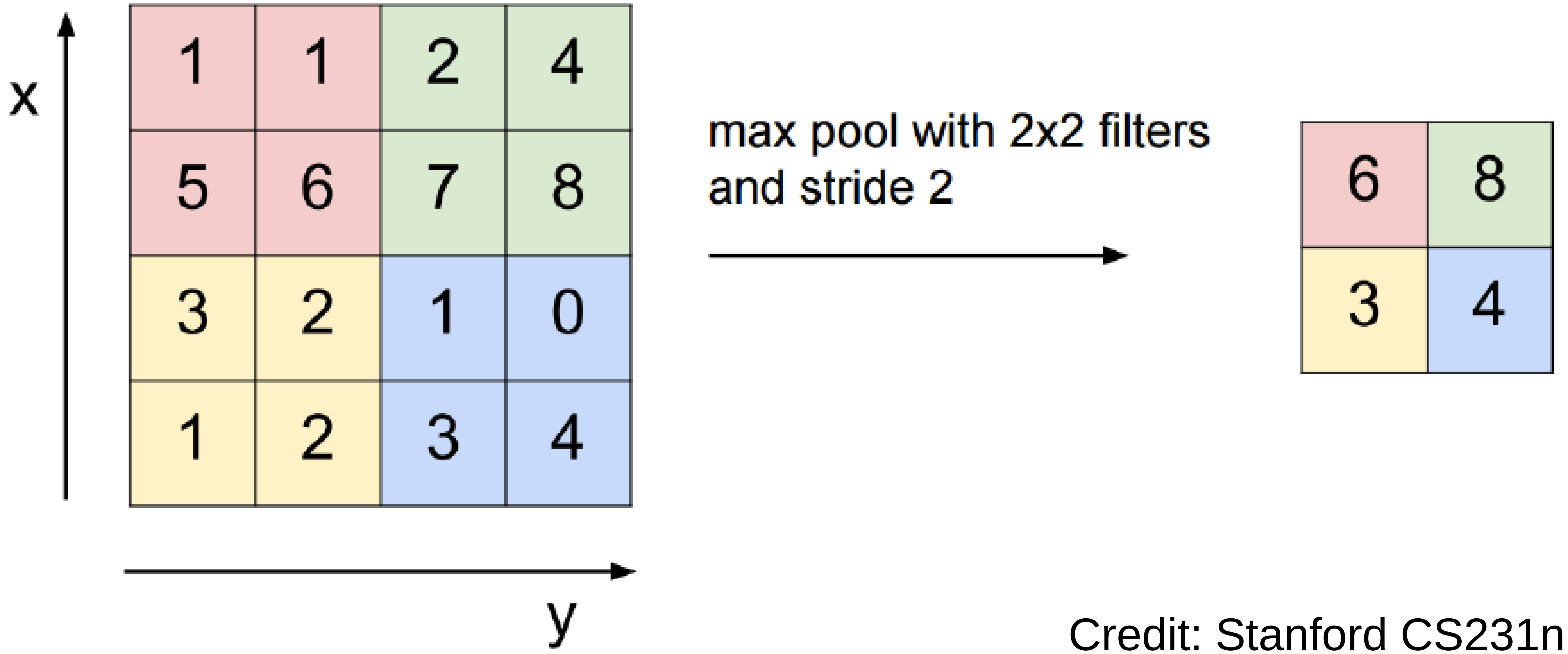
Pooling Layers

Fixed function

- Down-sampling
- Controls overfitting

Variants:

- Max pooling (most common)
- Average pooling



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

6	8
3	4

Credit: Stanford CS231n