Introduction to Neural Networks

CA Session
CS 231A

Krishnan Srinivasan - 12.02.2021

Agenda

Intro to Neural Networks

- Background and Applications
- Fully-connected Neural Networks (MLP)
- Convolutional Neural Networks (CNN)
- Backpropagation Algorithm
- PyTorch Example

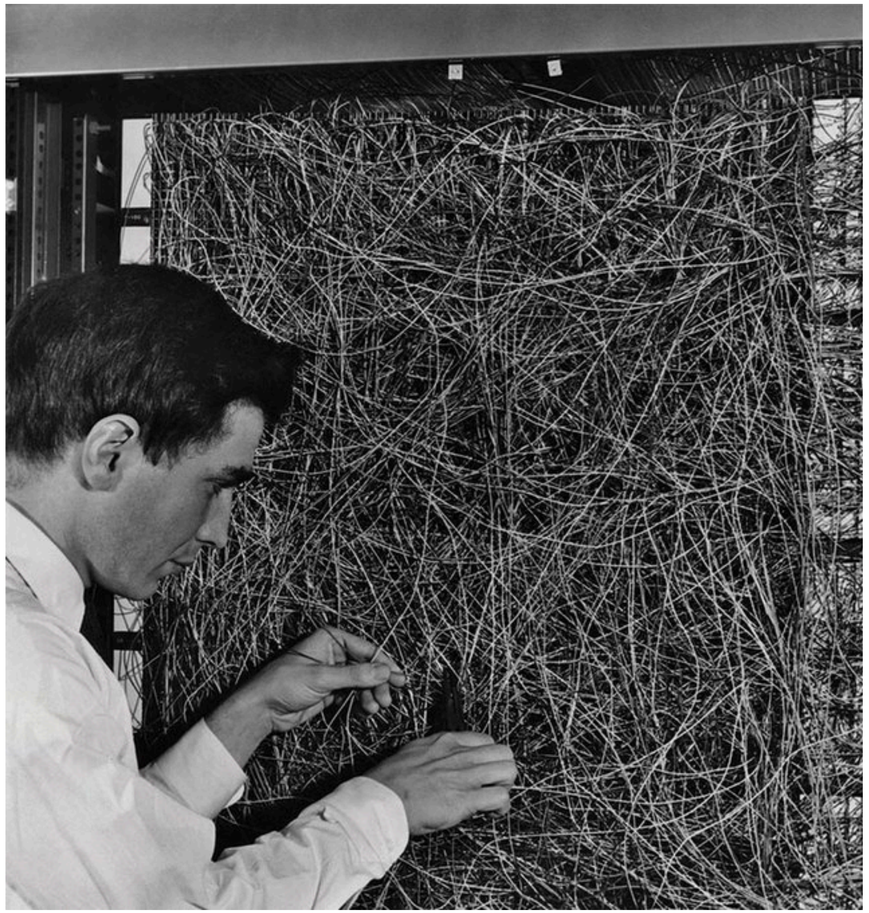
Background


SEQUENCE INDICATORS
22
23
24
31
32
33
34
MAIN SEQUENCE
START BUTTONS
STEP BACK
STEP FORWARD
MECHANISM NO. 2
STEP BACK
STEP FORWARD
MECHANISM NO. 3
MAIN SEQUENCE
STEP BUTTONS
22
23
24
31
32
33
34
MAIN SEQUENCE
SEQUENCE READ-OUT LIGHTS

Background

History

- 1957: Frank Rosenblatt designs the Mark I Perceptron, an early learning-based computer

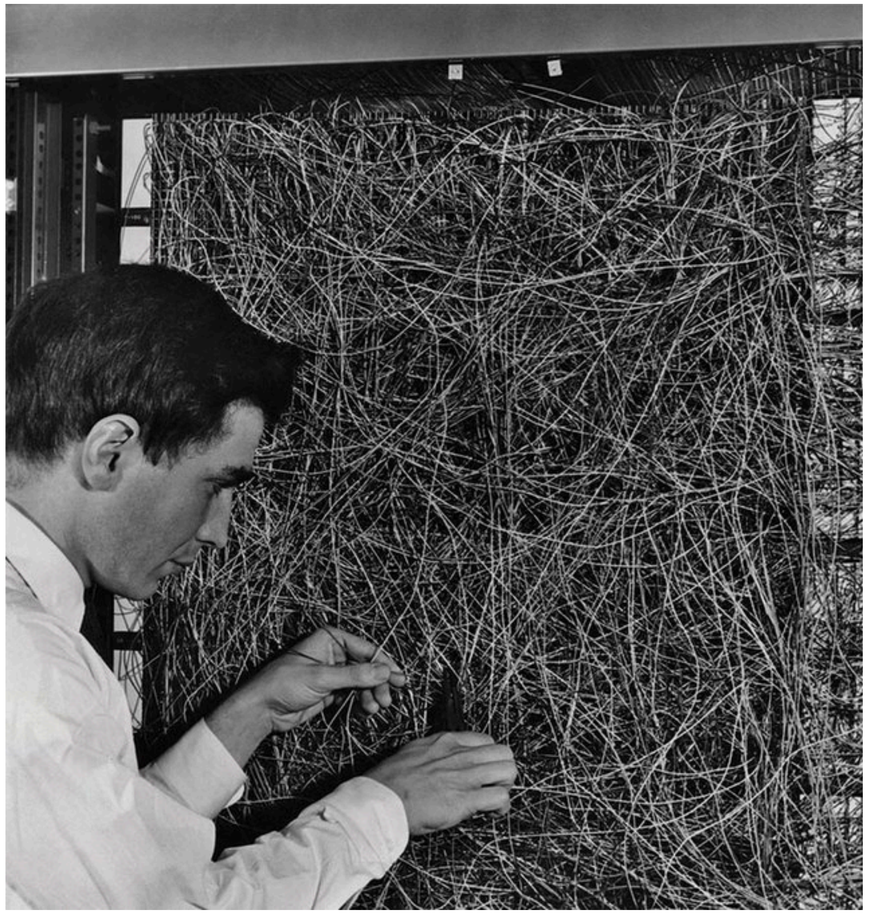


Tuning hyperparameters used to take **a lot longer** in Rosenblatt's day

Background

History

- 1957: Frank Rosenblatt designs the Mark I Perceptron, an early learning-based computer
- 1969: Multi-layer perceptron (early fully-connected neural networks) by Minksy and Papert

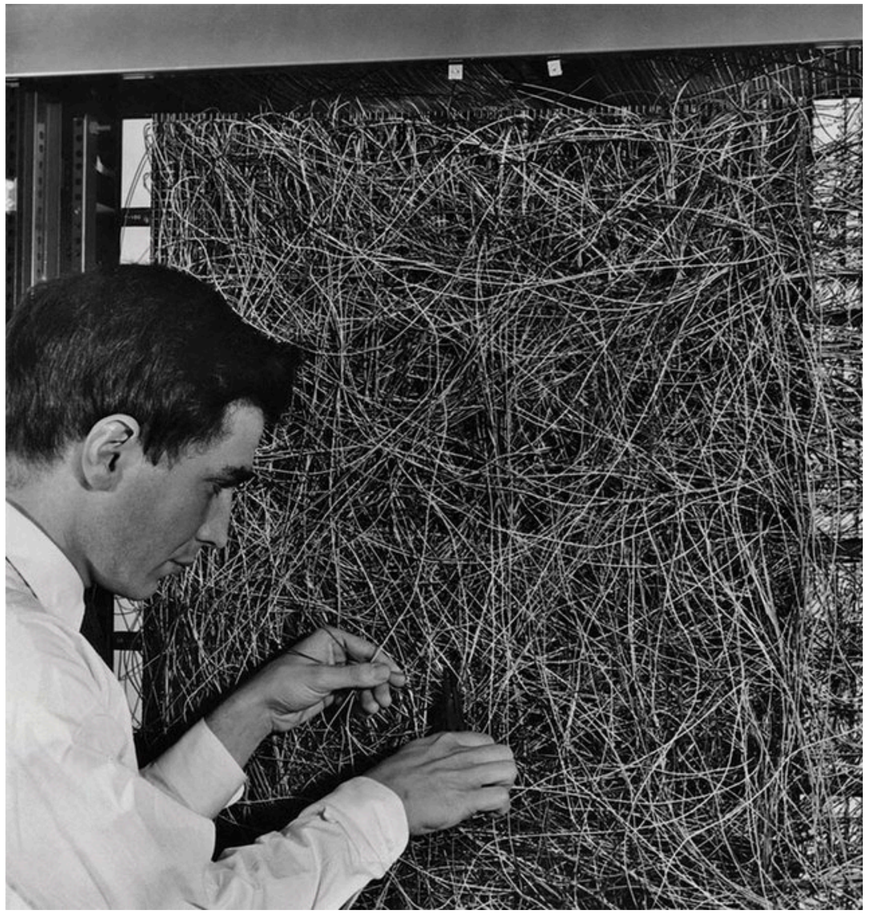


Tuning hyperparameters used to take **a lot longer** in Rosenblatt's day

Background

History

- 1957: Frank Rosenblatt designs the Mark I Perceptron, an early learning-based computer
- 1969: Multi-layer perceptron (early fully-connected neural networks) by Minksy and Papert
- 1986: Rumelhart, Hinton, and Williams (and others) develop the backpropagation algorithm (BP)

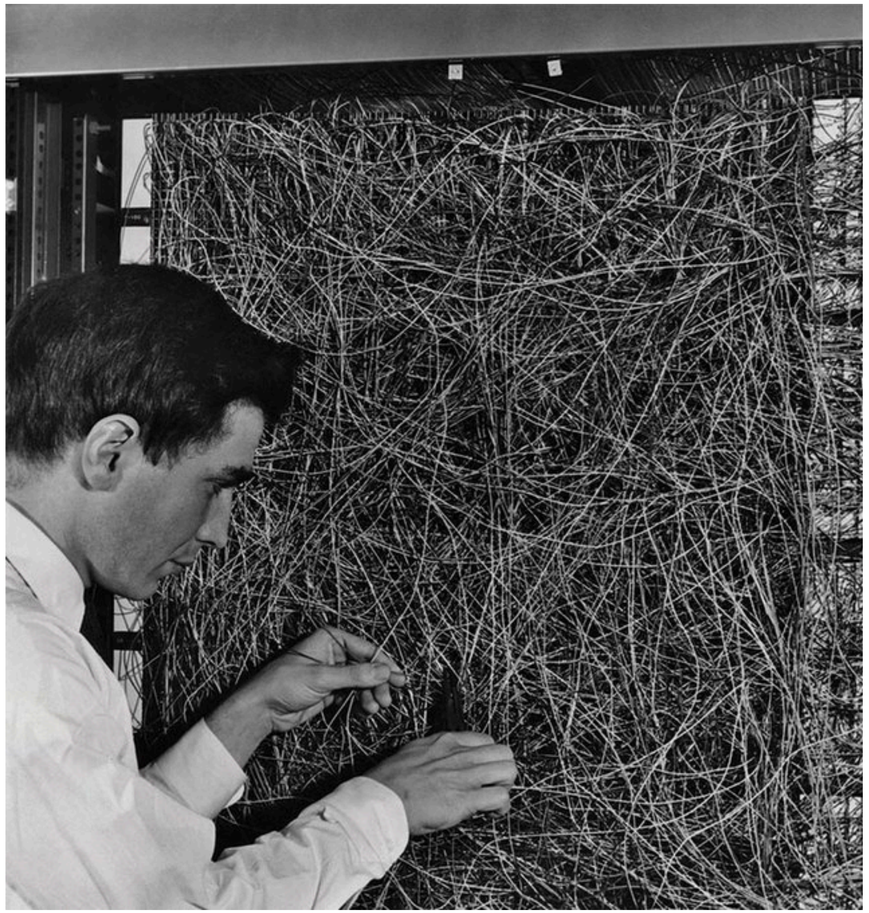


Tuning hyperparameters used to take **a lot longer** in Rosenblatt's day

Background

History

- 1957: Frank Rosenblatt designs the Mark I Perceptron, an early learning-based computer
- 1969: Multi-layer perceptron (early fully-connected neural networks) by Minksy and Papert
- 1986: Rumelhart, Hinton, and Williams (and others) develop the backpropagation algorithm (BP)
- 1989: LeCun et al. develop BP for Convolutional Neural Networks (CNNs), and introduce MNIST dataset

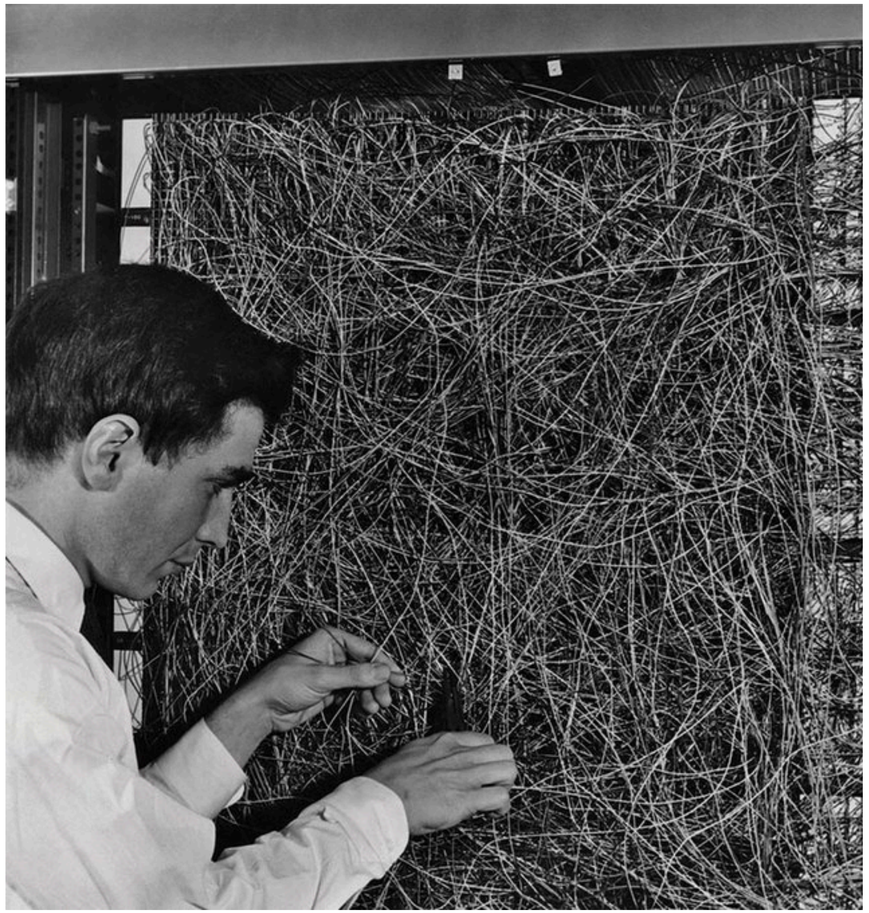


Tuning hyperparameters used to take **a lot longer** in Rosenblatt's day

Background

History

- 1957: Frank Rosenblatt designs the Mark I Perceptron, an early learning-based computer
- 1969: Multi-layer perceptron (early fully-connected neural networks) by Minksy and Papert
- 1986: Rumelhart, Hinton, and Williams (and others) develop the backpropagation algorithm (BP)
- 1989: LeCun et al. develop BP for Convolutional Neural Networks (CNNs), and introduce MNIST dataset
- 2012: AlexNet uses GPUs to train CNNs fast enough to be practical



Tuning hyperparameters used to take **a lot longer** in Rosenblatt's day

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

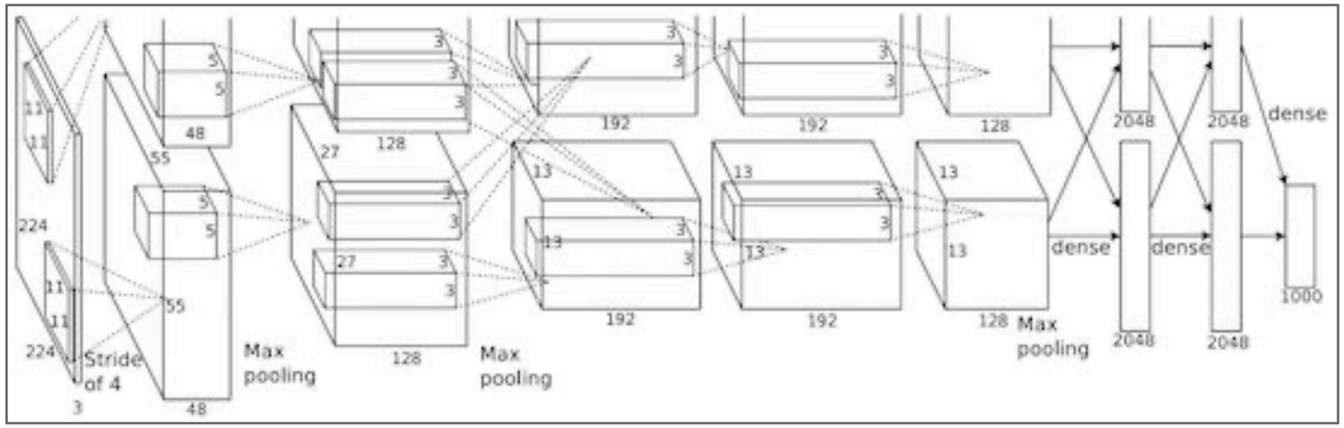


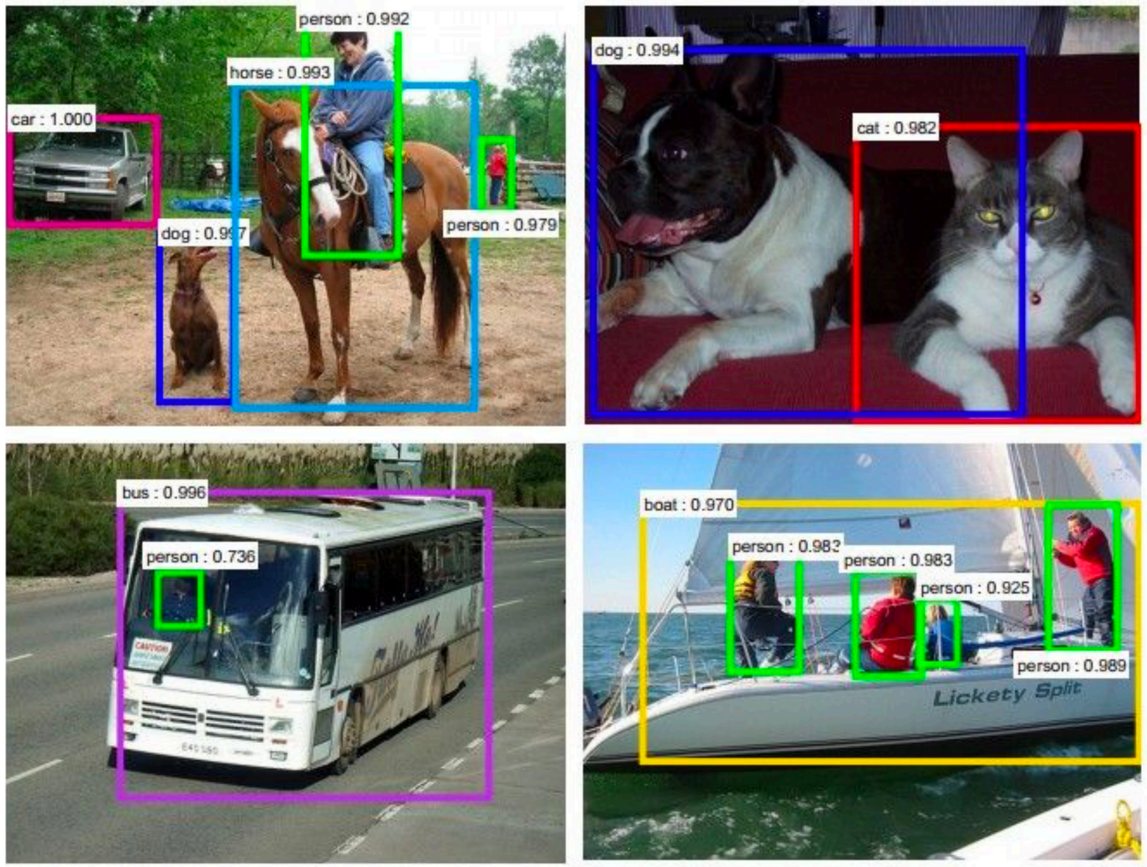
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

Following slides are borrowed from CS231N Lecture 5

Applications: Convolutional Networks

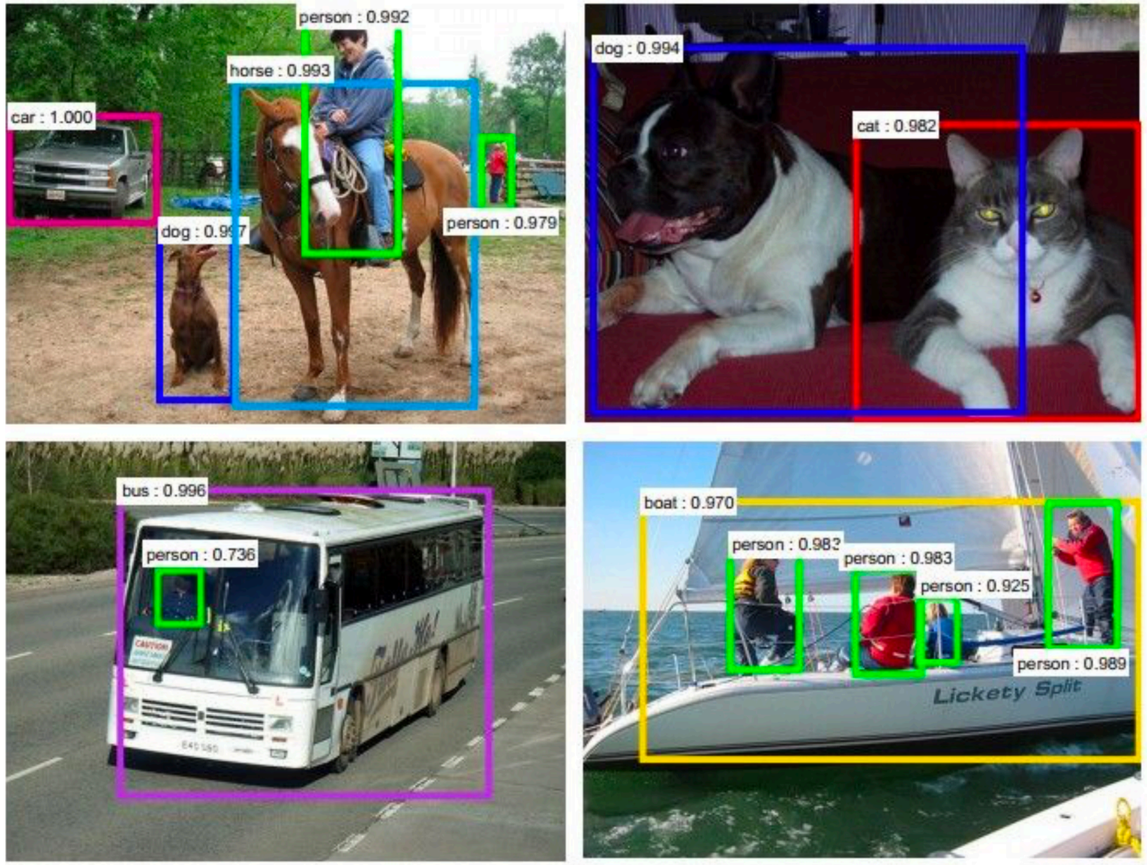
Detection



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Applications: Convolutional Networks

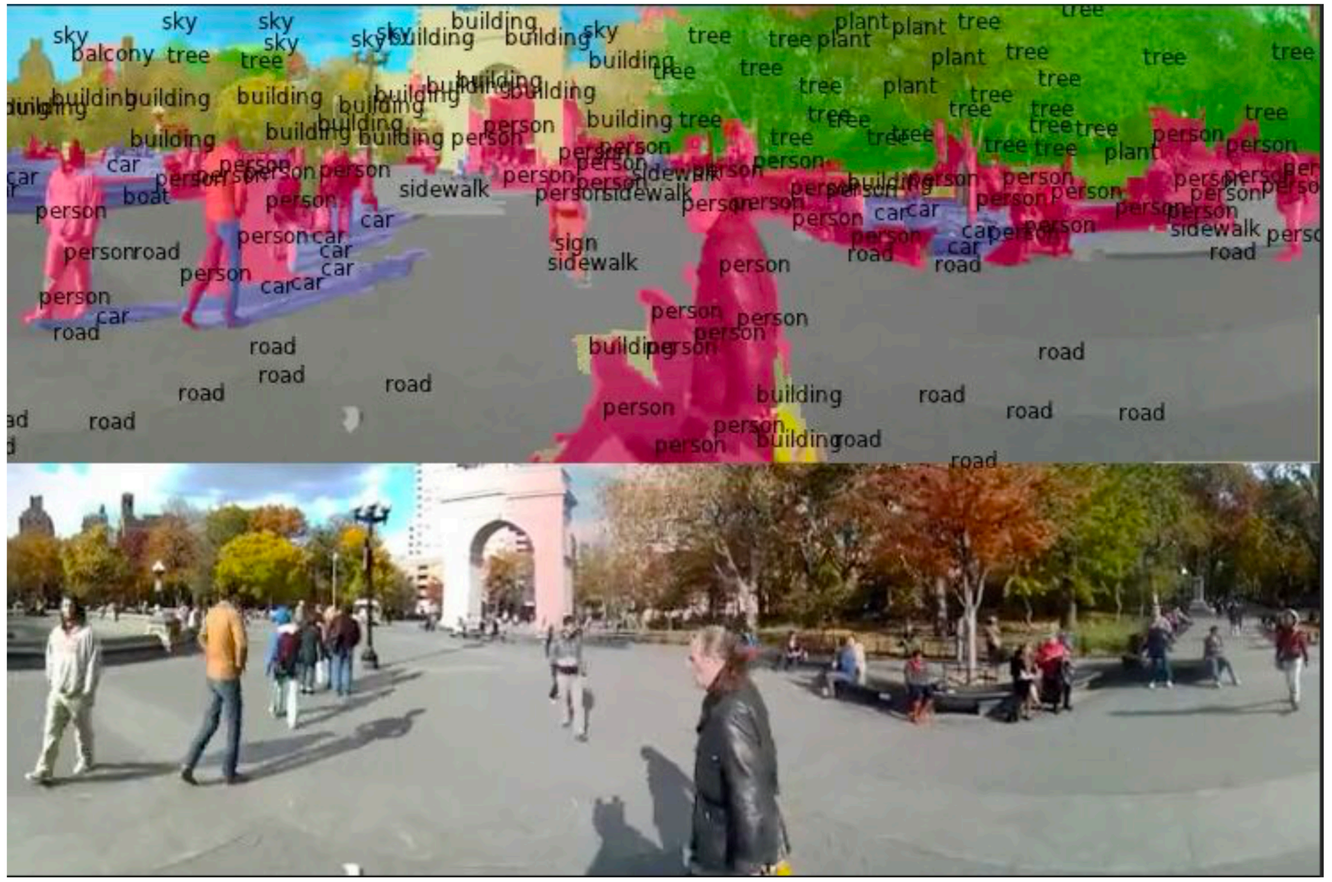
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



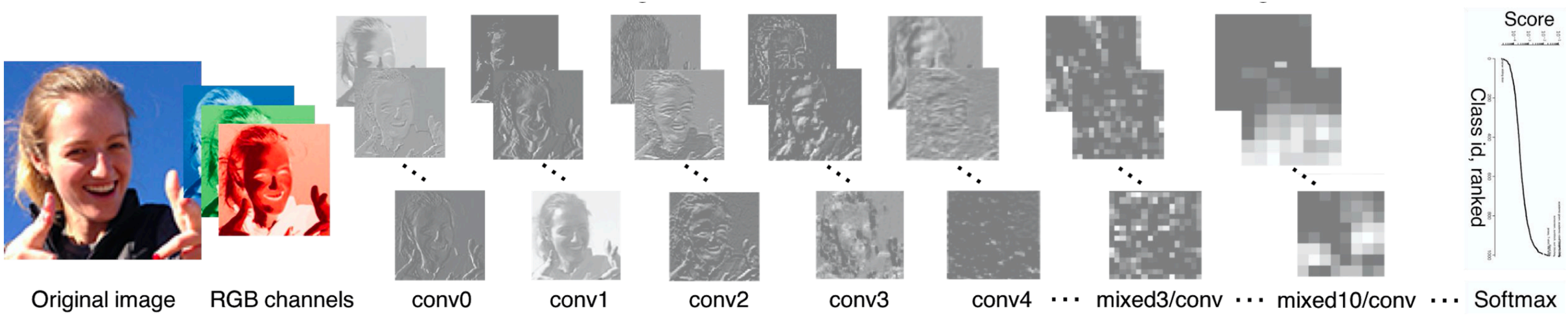
Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Classification

Retrieval

DeepFace (Face Verification)

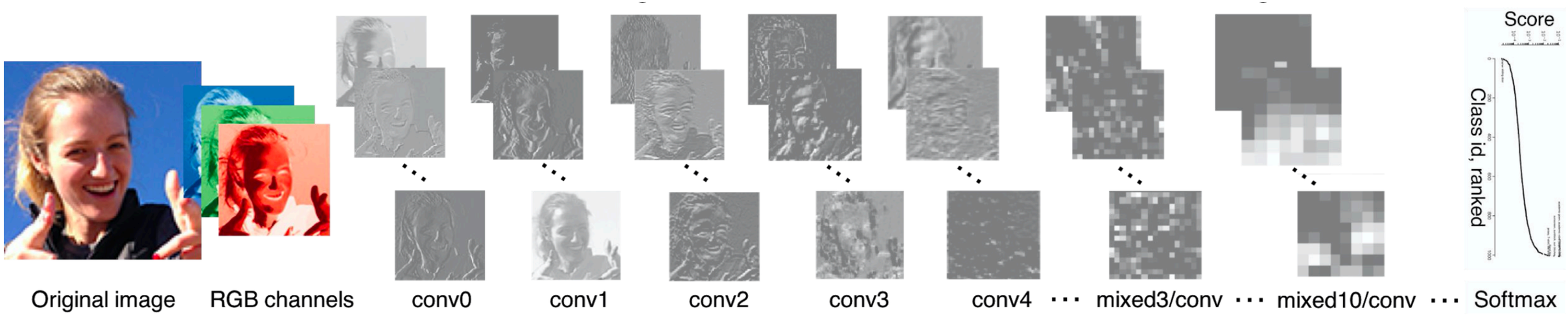


[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.

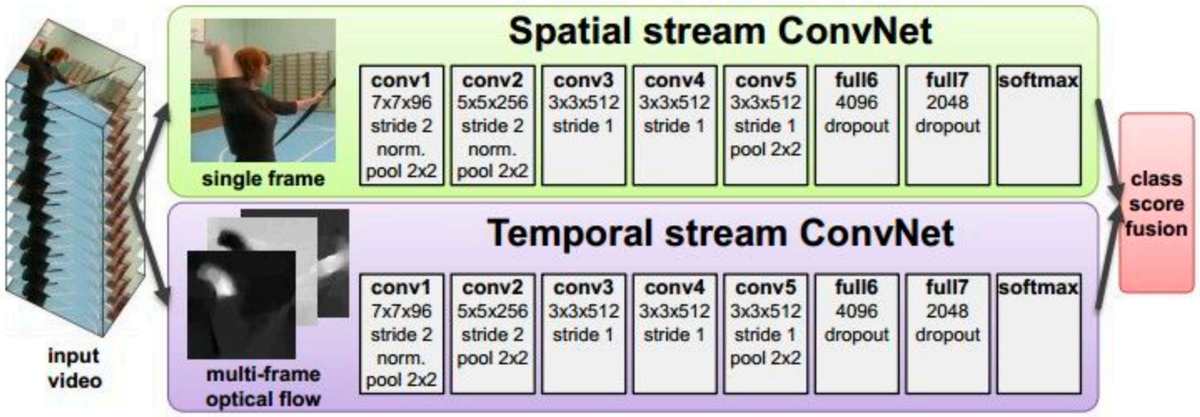
Two-Stream Convolutional Networks for Action Recognition in Videos

DeepFace (Face Verification)



[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

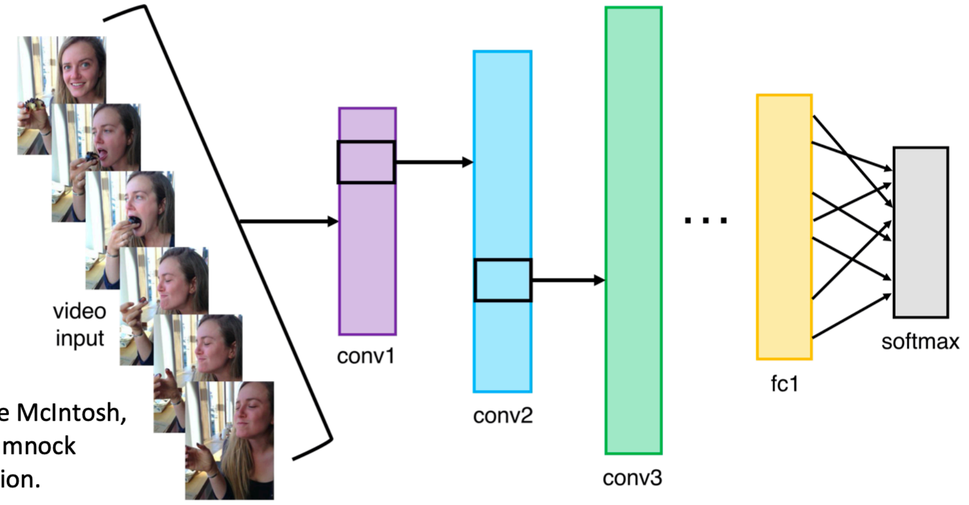
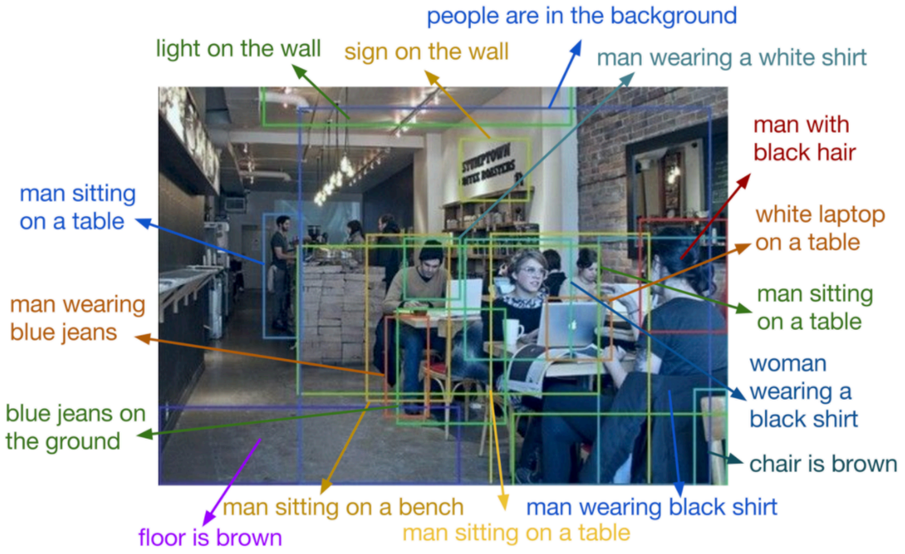
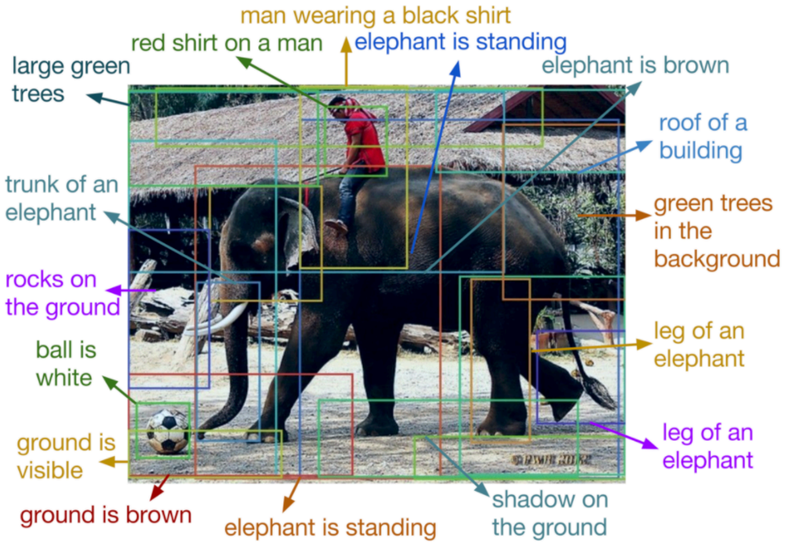


Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

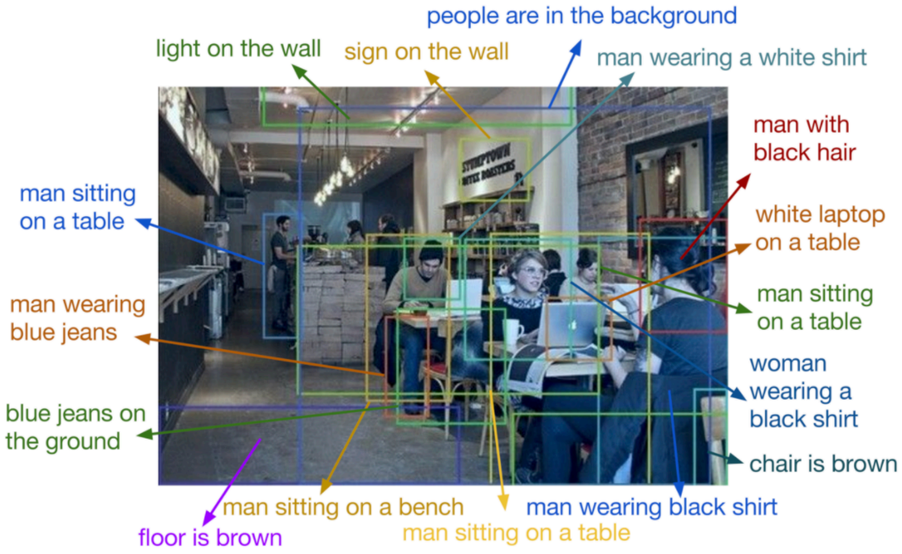
Two-Stream Convolutional Networks for Action Recognition in Videos

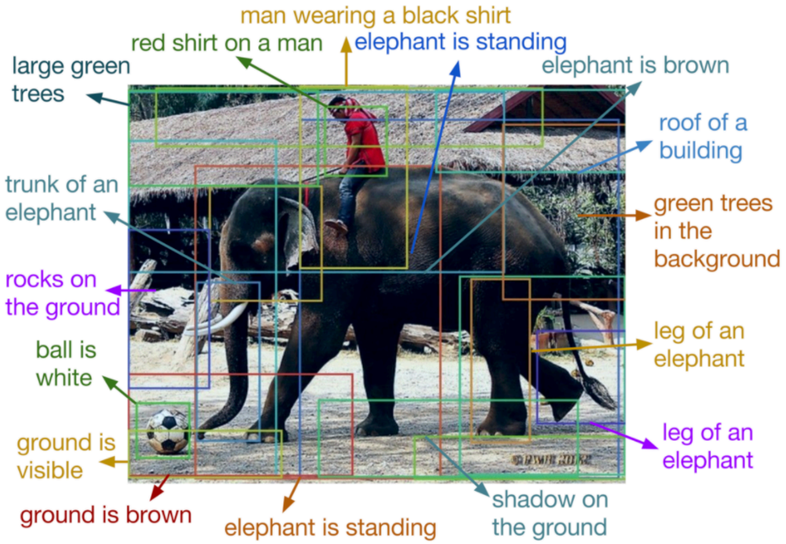
Dense Captioning [Johnson et al. 2016]





Dense Captioning [Johnson et al. 2016]





Visualizing Circuits [Voss et al. 2021]

Without context, these weights aren't very interesting.

With context, they show us how a head detector gets attached to a body.

FIGURE 3: Contextualizing weights.

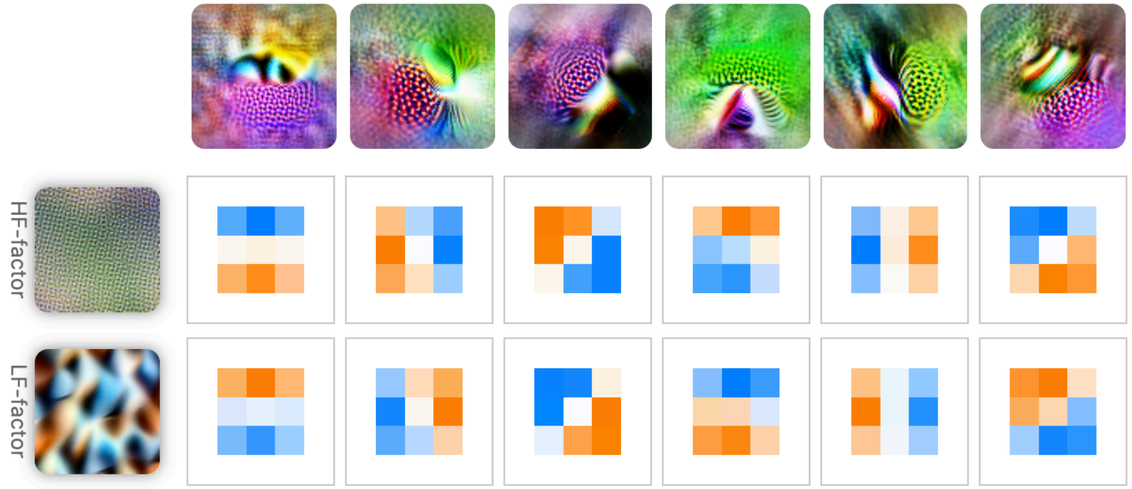
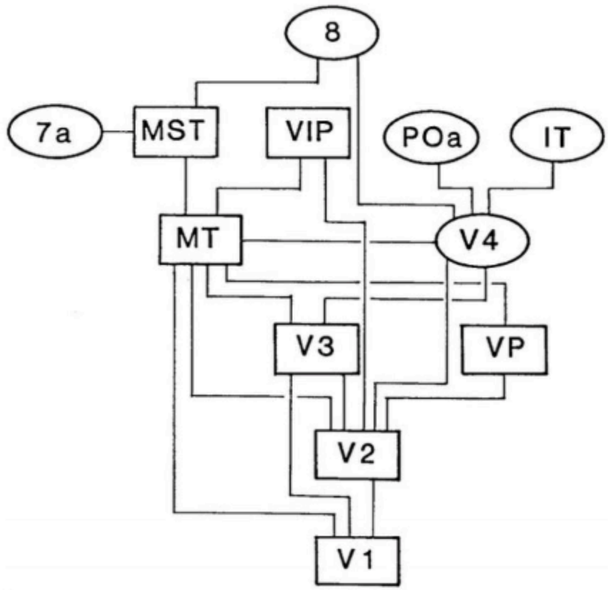
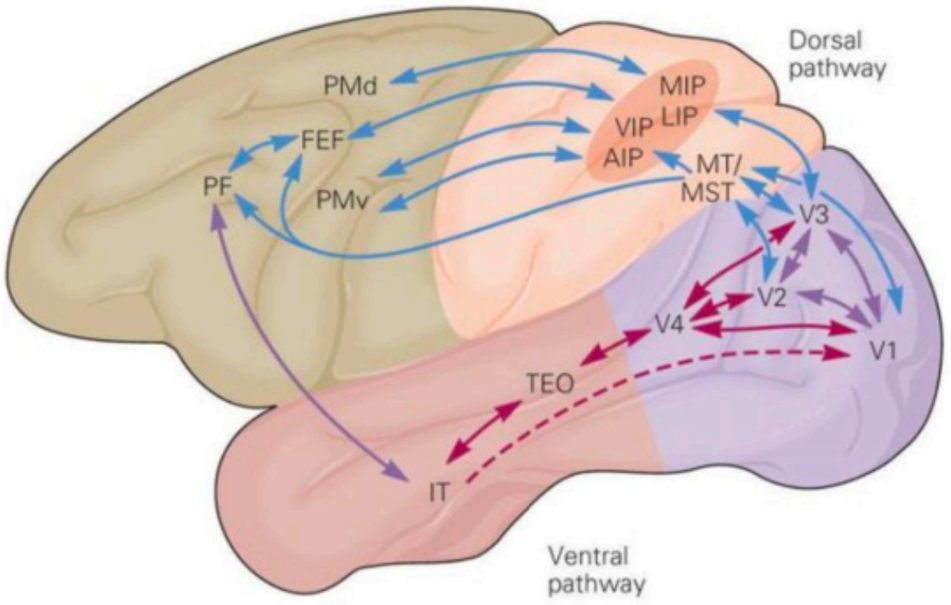


FIGURE 7: NMF factorization on the weights (excitatory and inhibitory) connecting six high-low frequency detectors in InceptionV1 to the layer `conv2d2`.

Background

Signal Relay

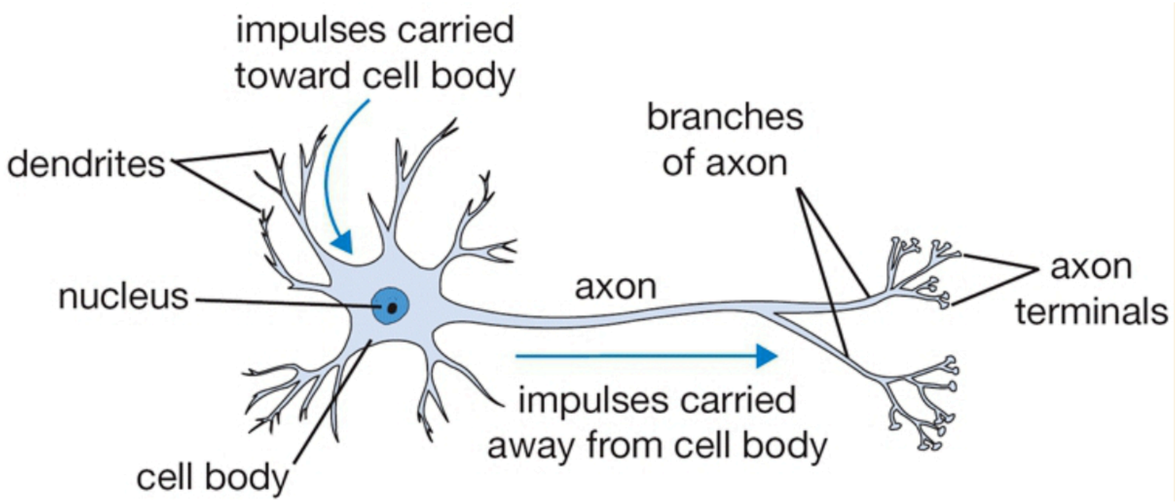


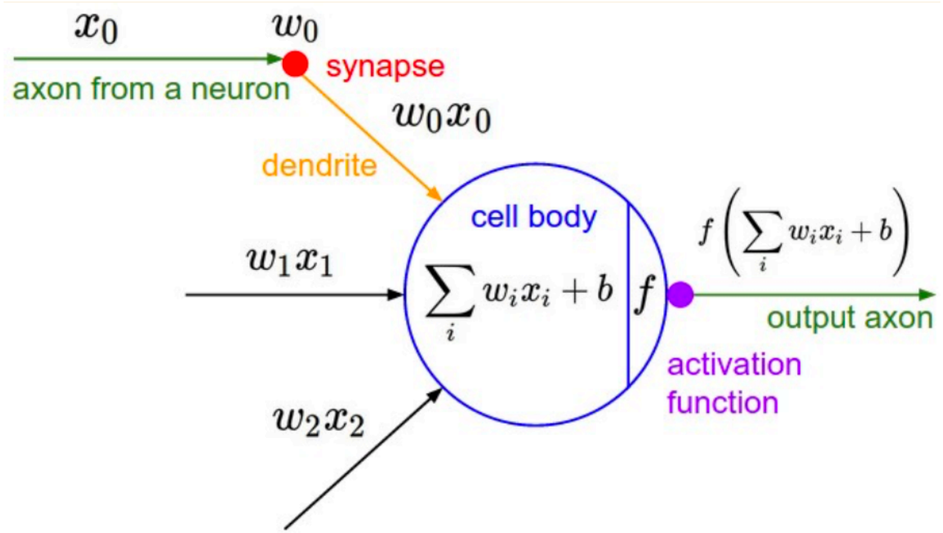


Starting from V1 primary visual cortex, visual signal is transmitted upwards, forming a more complex and abstract representation at every level

Foundations of Vision, Brian A. Wandell (1995)

Fully-Connected Neural Networks





Fully-Connected Neural Networks

Components

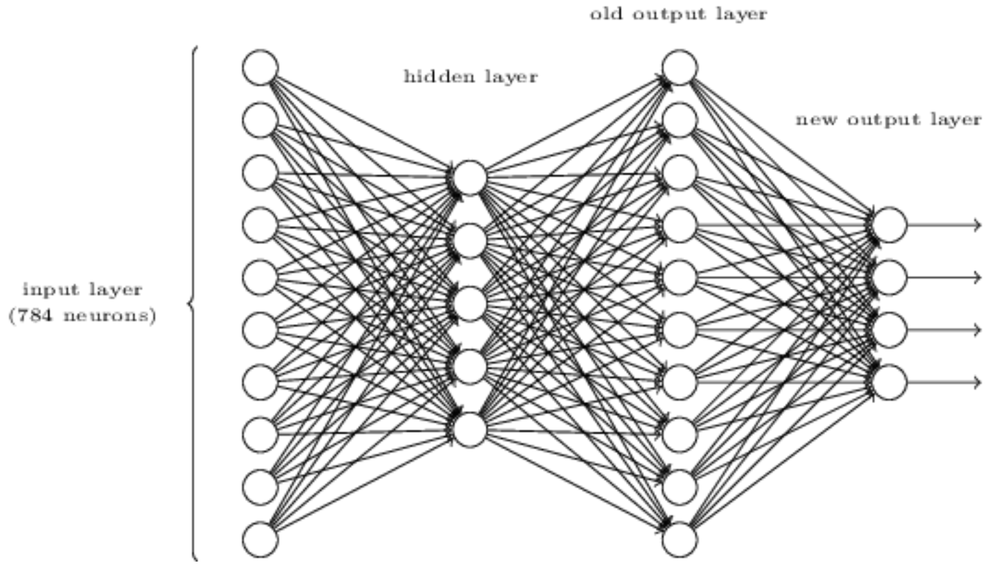


Image source

Fully-Connected Neural Networks

Components

- A single input layer, $h_0 \in \mathbb{R}^n$

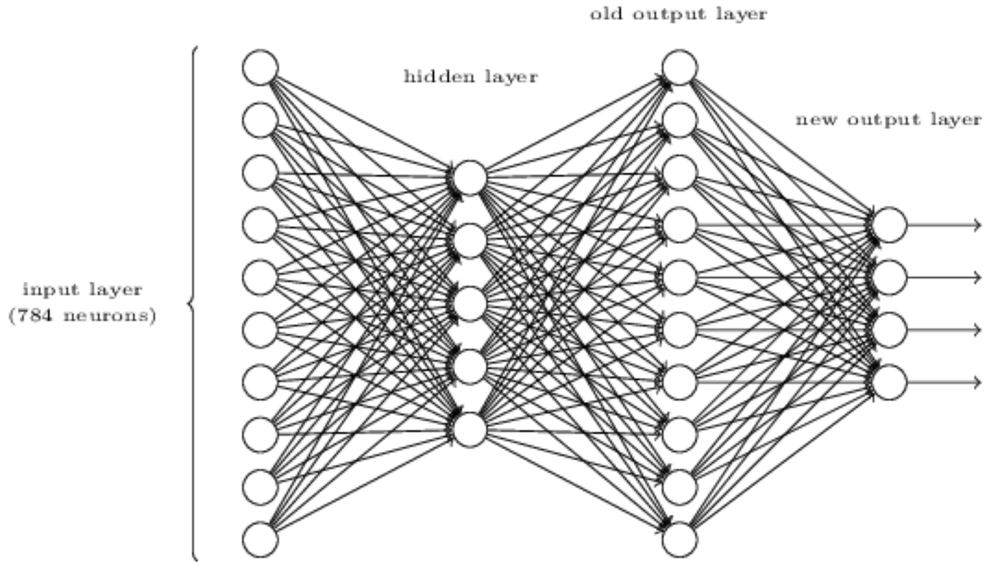


Image source

Fully-Connected Neural Networks

Components

- A single input layer, $h_0 \in \mathbb{R}^n$
- k- hidden layers, $a_i \in \mathbb{R}^{d_i}$
  - Weight matrices, $W_i \in \mathbb{R}^{d_{i-1} \times d_i}$
  - Bias vectors, $b_i \in \mathbb{R}^{d_i}$

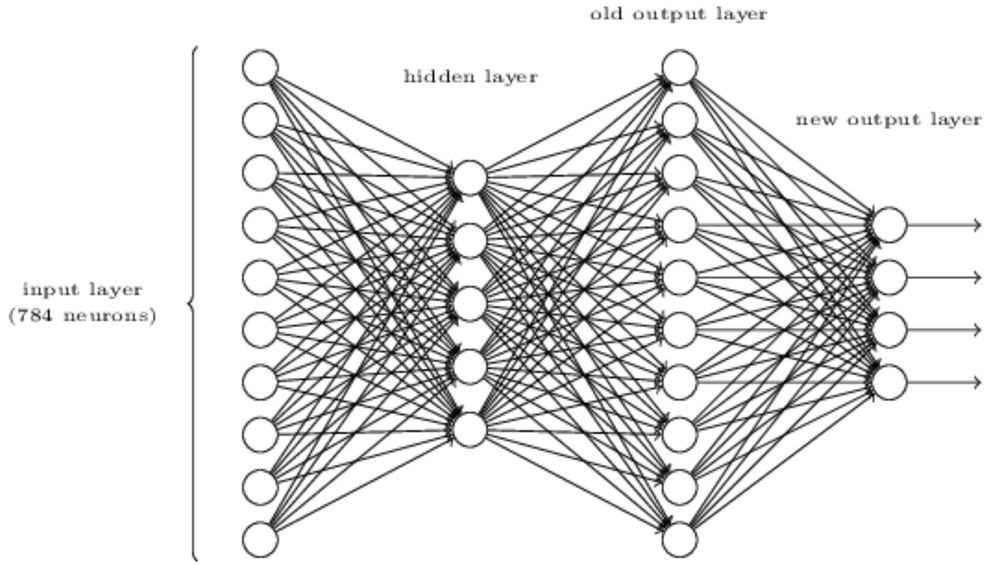


Image source

Fully-Connected Neural Networks

Components

- A single input layer, $h_0 \in \mathbb{R}^n$
- k- hidden layers, $a_i \in \mathbb{R}^{d_i}$
  - Weight matrices, $W_i \in \mathbb{R}^{d_{i-1} \times d_i}$
  - Bias vectors, $b_i \in \mathbb{R}^{d_i}$
- Output layer, $\hat{y} \in \mathbb{R}^m$

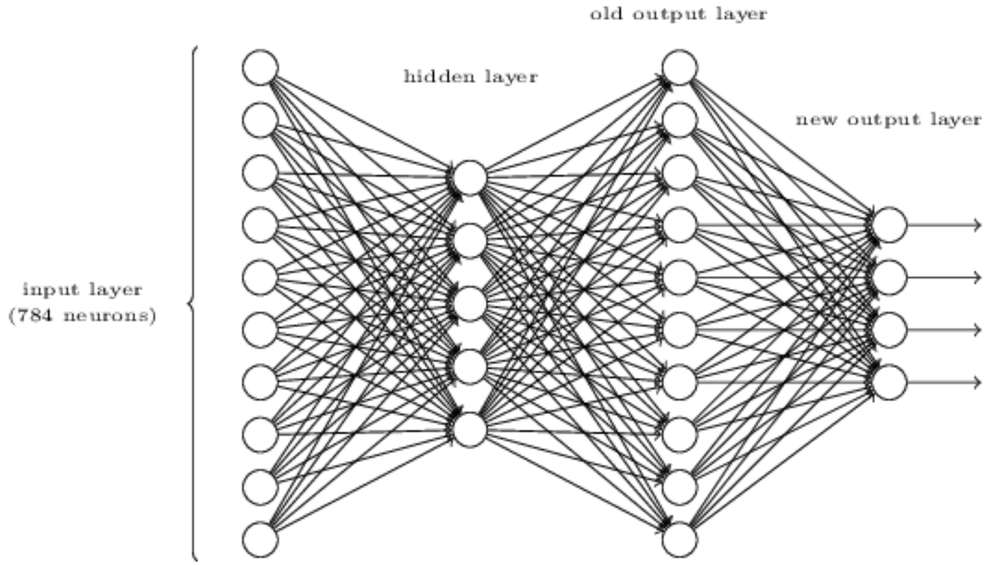


Image source

Fully-Connected Neural Networks

Components

- A single input layer, $h_0 \in \mathbb{R}^n$
- k- hidden layers, $a_i \in \mathbb{R}^{d_i}$
  - Weight matrices, $W_i \in \mathbb{R}^{d_{i-1} \times d_i}$
  - Bias vectors, $b_i \in \mathbb{R}^{d_i}$
- Output layer, $\hat{y} \in \mathbb{R}^m$
- For each layer, $a_i = f(z_i) = f(W_i h_i + b_i)$, where f is an activation function

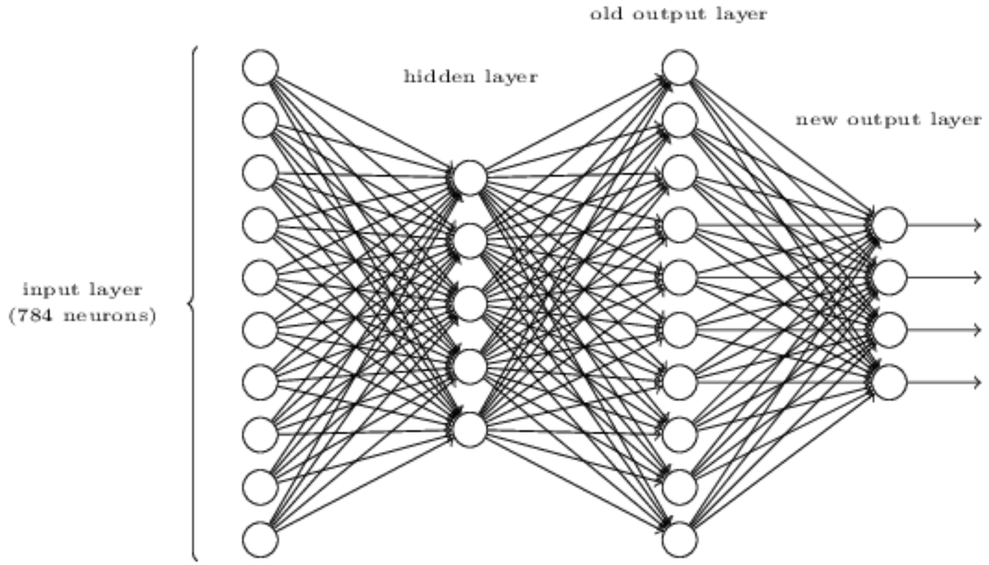


Image source

Fully-Connected Neural Networks

Components

- A single input layer, $h_0 \in \mathbb{R}^n$
- k- hidden layers, $a_i \in \mathbb{R}^{d_i}$
  - Weight matrices, $W_i \in \mathbb{R}^{d_{i-1} \times d_i}$
  - Bias vectors, $b_i \in \mathbb{R}^{d_i}$
- Output layer, $\hat{y} \in \mathbb{R}^m$
- For each layer, $a_i = f(z_i) = f(W_i h_i + b_i)$, where f is an activation function
- Series of stacked layers compose multiple function together (e.g. $(f \circ g)(x)$)

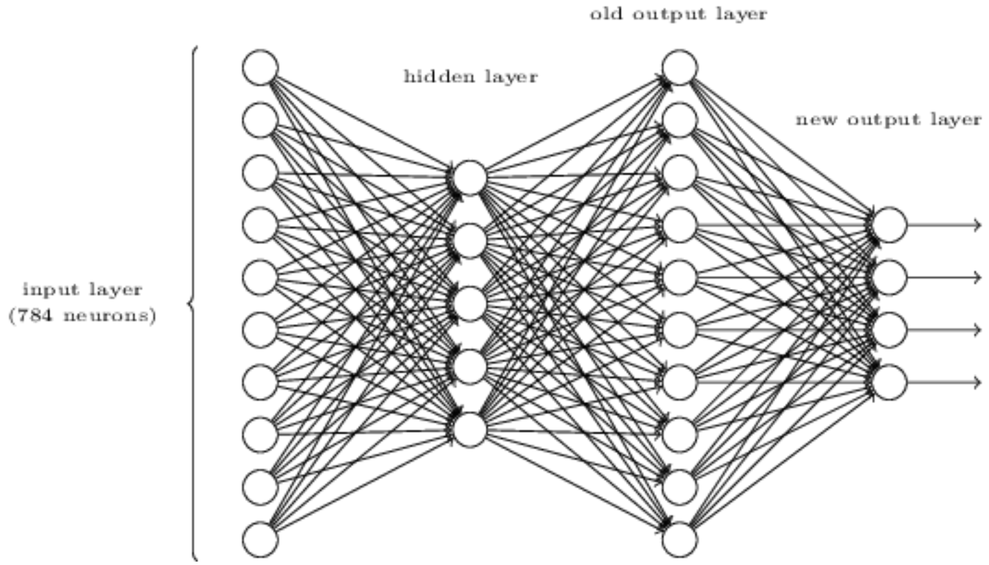


Image source

Fully-Connected Neural Networks

Cost Function

Fully-Connected Neural Networks

Cost Function

- To train parameters, compute a cost associated with every predicted/labeled output pair, $y, \hat{y}$.

Fully-Connected Neural Networks

Cost Function

- To train parameters, compute a cost associated with every predicted/labeled output pair, $y, \hat{y}$.
- Requirements: can be averaged over a batch, can be computed with outputs from network

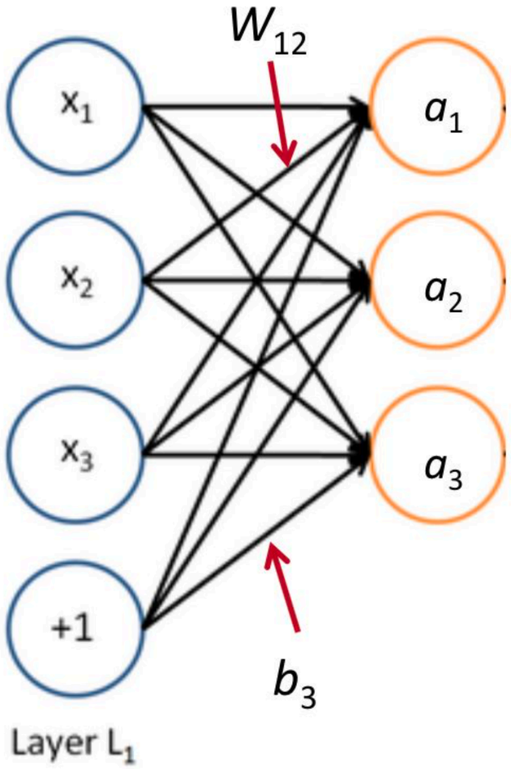
Fully-Connected Neural Networks

Cost Function

- To train parameters, compute a cost associated with every predicted/labeled output pair, $y, \hat{y}$.
- Requirements: can be averaged over a batch, can be computed with outputs from network
- Common loss functions:
  - Least squares (quadratic): $\frac{1}{2m}\sum_{i=1}^{m} \|y_i - \hat{y}_i\|^2$
  - Binary Cross-Entropy: $y\log(\hat{y}) + (1-y)\log(1-\hat{y})$
  - Cross entropy (classification, y_j is one-hot encoding at j): $\sum_{i=1}^{m} y_i \log(\hat{y}_i)$

Fully-Connected Neural Network

Example

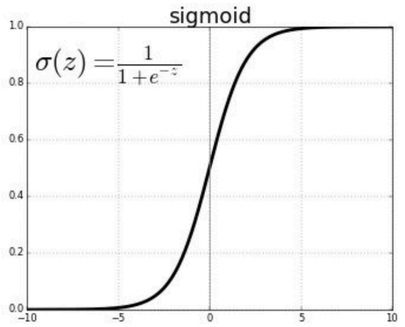


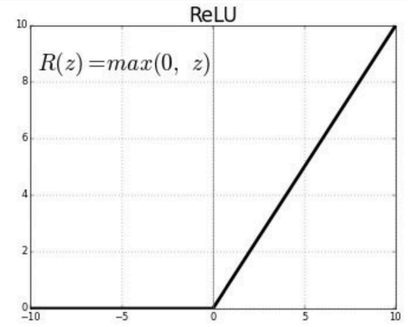
$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$
$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$
$$a_3 = f(W_{31}x_1 + W_{32}x_2 + W_{33}x_3 + b_3)$$

Sigmoid (logit) transform. $\sigma(z) = \frac{1}{1+e^{-z}}$

Hyperbolic tangent (tanh). $\tanh(z) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Rectified Linear Unit (ReLU). $\mathrm{ReLU}(z) = \max(0, z)$





Convolutional Neural Networks

Introduction

- For computer vision applications, convolutional networks are used to learn feature detectors from images
- Advantages:
  - Images are high-dimensional data, fully connected layers would require too many parameters to tune
  - Convolution operations preserve spatial structure of data
  - Convolution operation can be computed efficiently on GPUs (using CUDA)
- Analogues:
  - Inputs/activations are “what” the network “sees”
  - Weights are “how” the network computes one layer from the previous one (feature-detection)
- As architectures become more complex, interpretability of these learned features becomes more difficult

Convolutional Neural Networks

Components

Convolutional Neural Networks

Components

- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$

Convolutional Neural Networks

Components

- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape `[relative x-position, relative y-position, input channels, output channels]`

Convolutional Neural Networks

Components

- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:

Convolutional Neural Networks

Components

- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
    - Number of filters, or *depth* K (each channel also called an “activation map”)

Convolutional Neural Networks

Components

- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
    - Number of filters, or *depth* K (each channel also called an “activation map”)
    - *Spatial extent*, or *receptive field* F

Convolutional Neural Networks

Components

- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
  - Number of filters, or *depth* K (each channel also called an “activation map”)
  - *Spatial extent*, or *receptive field* F
  - The stride S

Convolutional Neural Networks

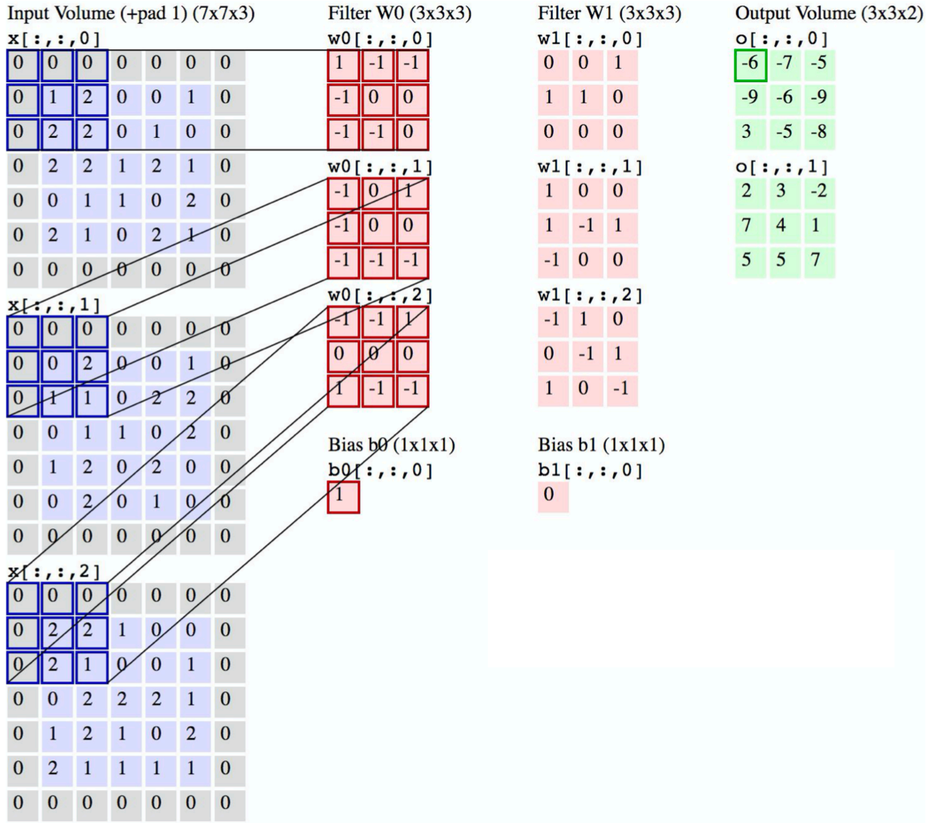
Components

- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
  - Number of filters, or *depth* K (each channel also called an “activation map”)
  - *Spatial extent*, or *receptive field* F
  - The stride S
  - Amount of zero-padding P

Convolutional Neural Networks

Components

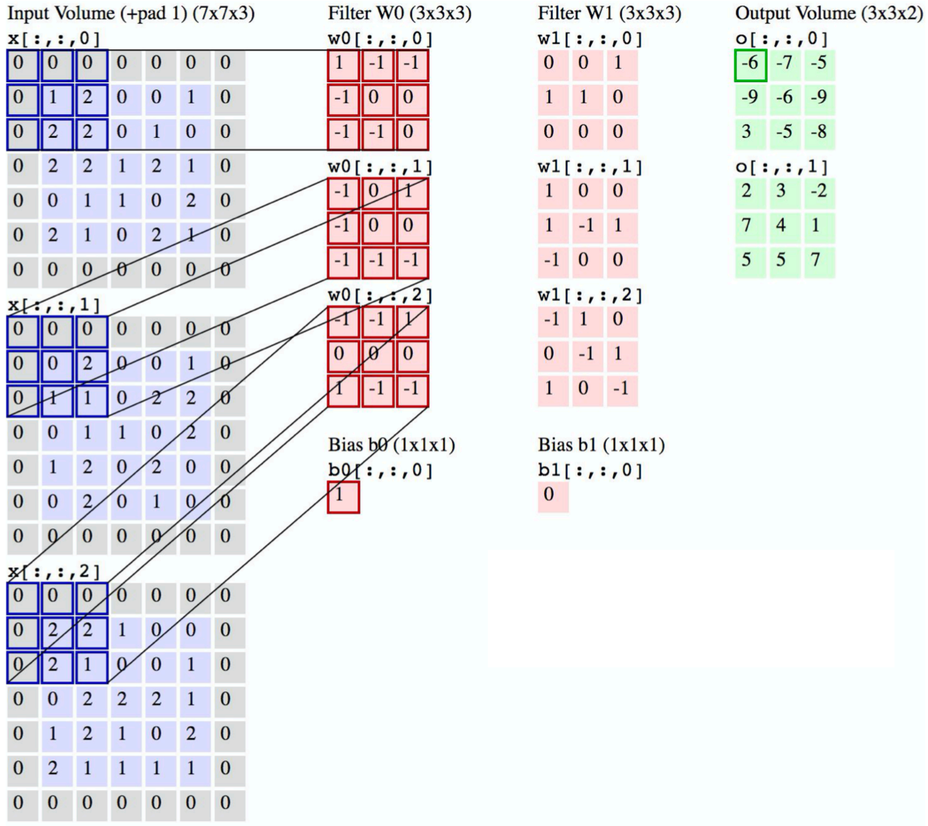
- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
  - Number of filters, or *depth* K (each channel also called an “activation map”)
  - *Spatial extent*, or *receptive field* F
  - The stride S
  - Amount of zero-padding P



Convolutional Neural Networks

Components

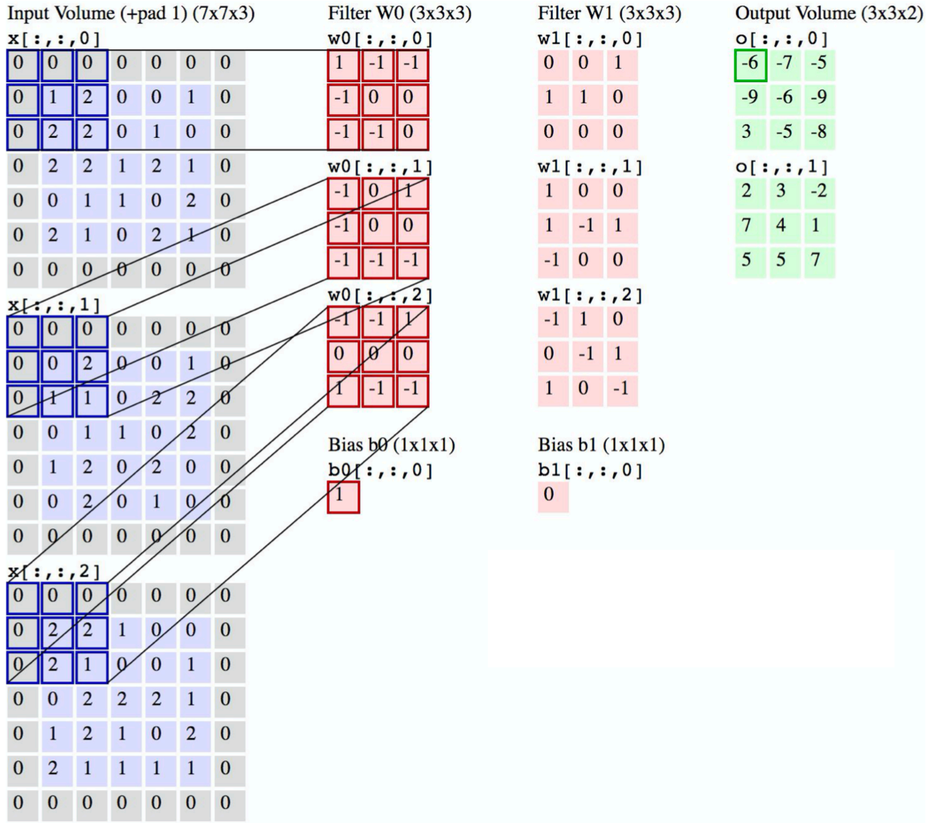
- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
  - Number of filters, or *depth* K (each channel also called an “activation map”)
  - *Spatial extent*, or *receptive field* F
  - The stride S
  - Amount of zero-padding P
- With this, the shape of layer l convolved from layer $l-1$ is:



Convolutional Neural Networks

Components

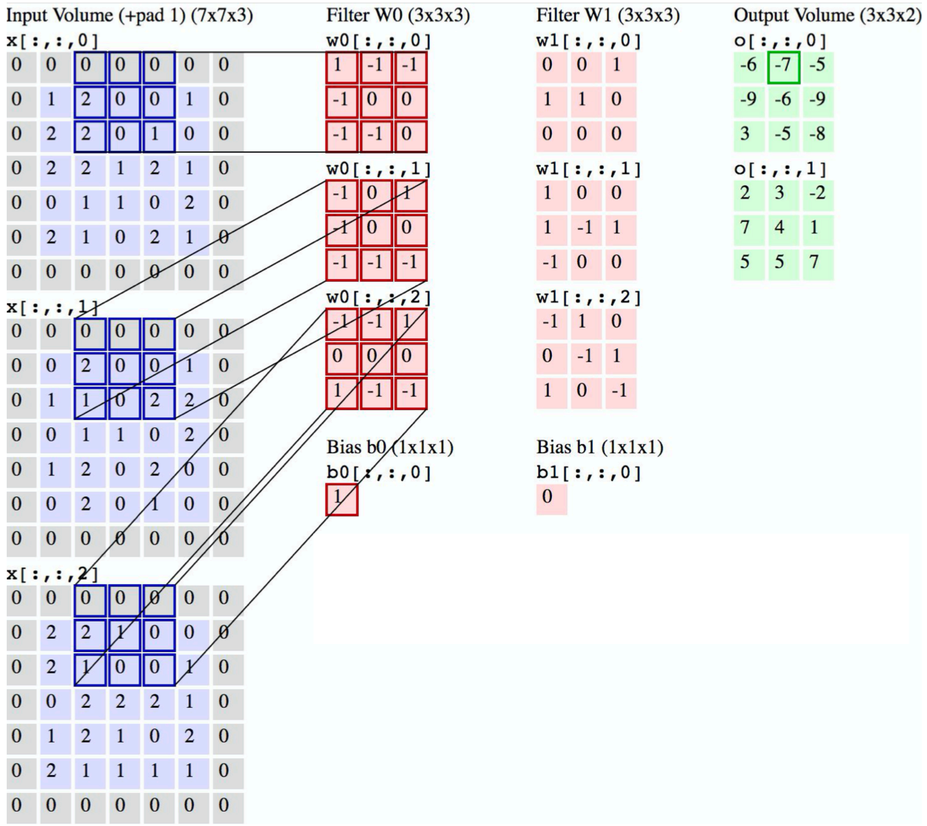
- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
  - Number of filters, or *depth* K (each channel also called an “activation map”)
  - *Spatial extent*, or *receptive field* F
  - The stride S
  - Amount of zero-padding P
- With this, the shape of layer l convolved from layer $l-1$ is:
  - $[\,(W-F+2P)/S+1, (H-F+2P)/S+1, K\,]$



Convolutional Neural Networks

Components

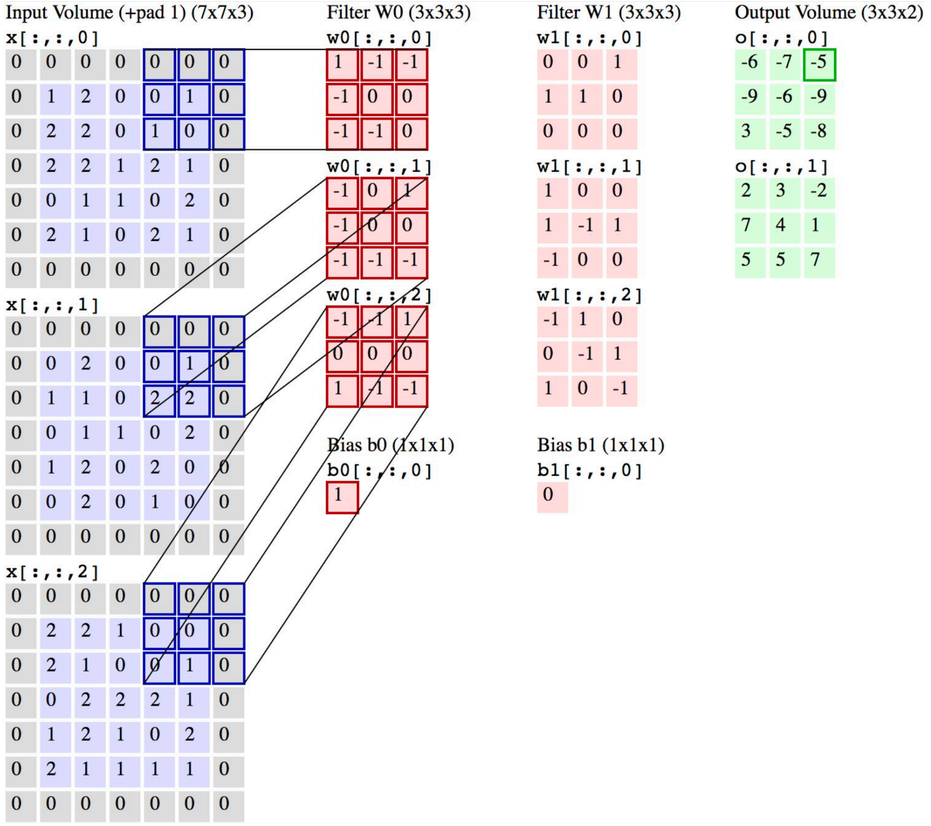
- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
  - Number of filters, or *depth* K (each channel also called an “activation map”)
  - *Spatial extent*, or *receptive field* F
  - The stride S
  - Amount of zero-padding P
- With this, the shape of layer l convolved from layer $l-1$ is:
  - $[\,(W-F+2P)/S+1, (H-F+2P)/S+1, K\,]$



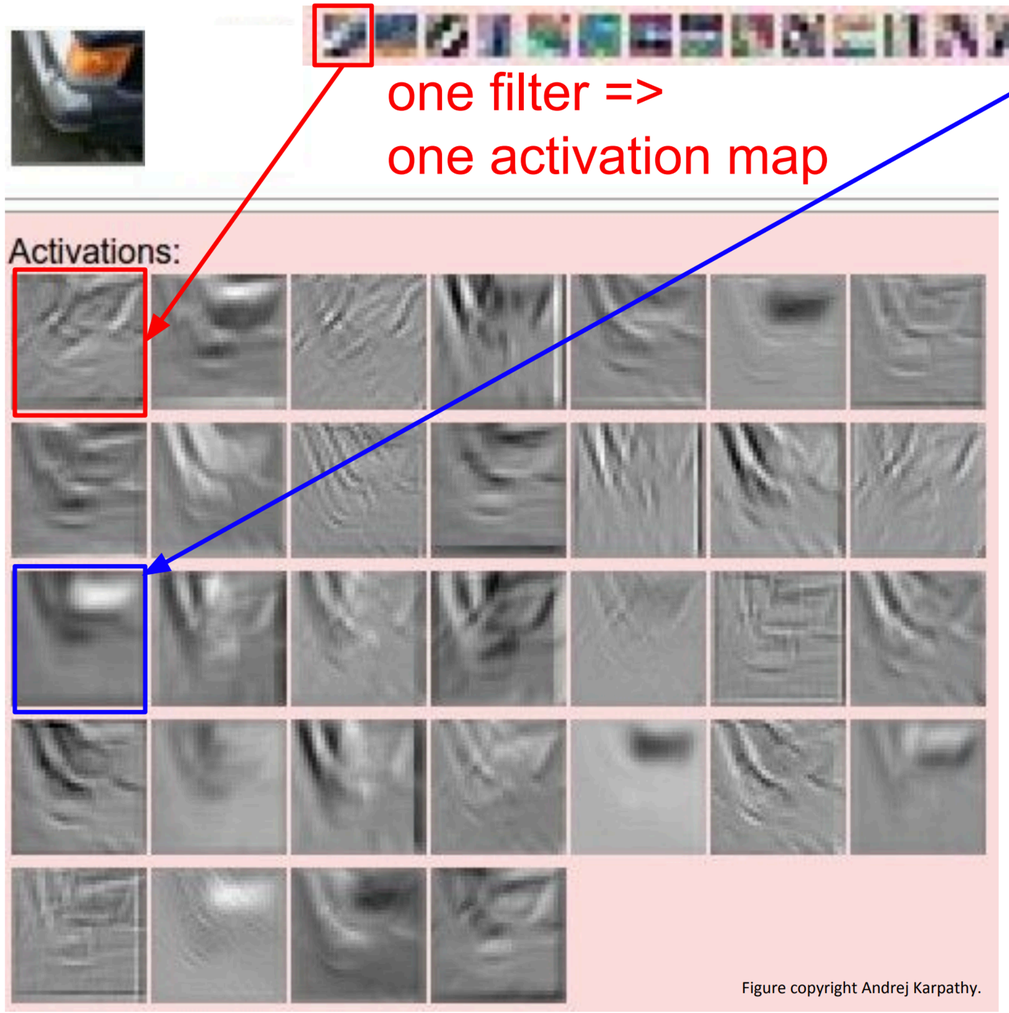
Convolutional Neural Networks

Components

- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
  - Number of filters, or *depth* K (each channel also called an “activation map”)
  - *Spatial extent*, or *receptive field* F
  - The stride S
  - Amount of zero-padding P
- With this, the shape of layer l convolved from layer $l-1$ is:
  - $[\,(W-F+2P)/S+1, (H-F+2P)/S+1, K\,]$



Convolutional Neural Networks



example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

Slide borrowed from CS231N Lecture 5

Convolutional Neural Network

Cuda document

Mathematically, a convolution measures the amount of overlap between two functions [1]. It can be thought of as a blending operation that integrates the point-wise multiplication of one dataset with another.

$$r(i) = (s * k)(i) = \int s(i-n)k(n)dn$$

In discrete terms this can be written as:

$$r(i) = (s * k)(i) = \sum_{n} s(i-n)k(n)\,.$$

Convolution can be extended into two dimensions by adding indices for the second dimension:

$$r(i) = (s * k)(i, j) = \sum_{n}\sum_{m}\ s(i-n, j-m)k(n, m)$$

[Source]

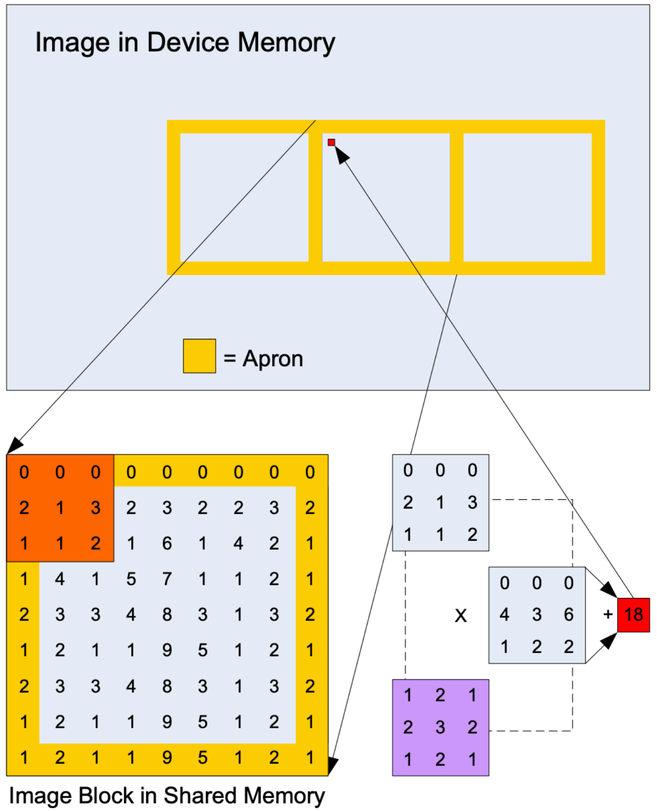
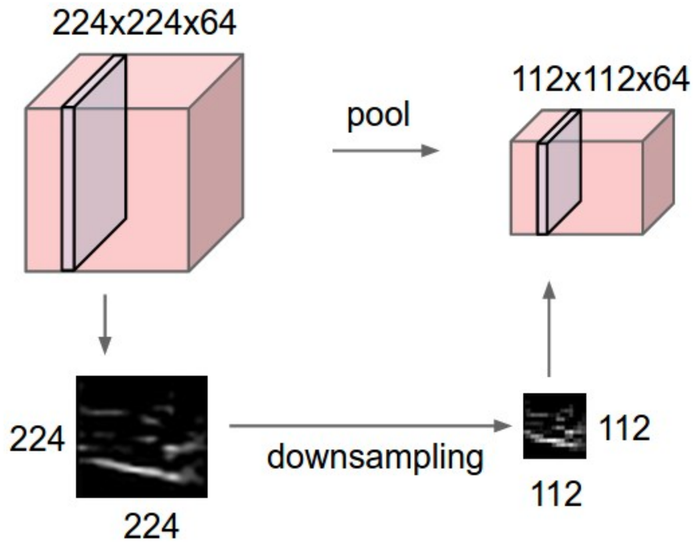


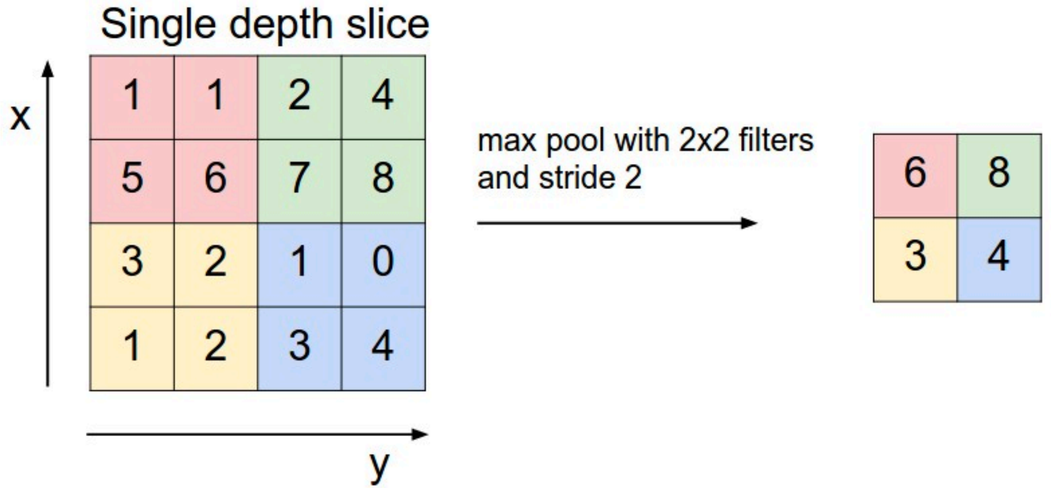
Figure 4: Modification of the naive algorithm of Figure 3 to include the image block apron region.

Convolutional Neural Networks

Pooling and FC layers

- Max and Average (L2-norm) pooling:
  - Downsampling operation to reduce width x height (but not depth) of a layer
- Fully-connected (FC) layers:
  - Flattens entire input volume to a vector, and treats like a normal FC network layer
- FC -> Conv layer:
  - A way to take advantage of GPU conv operations to perform large FC computations quickly
  - represent them as $[1 \times 1 \times h]$ convolutions





Backpropagation

Backpropagation: Preliminaries

- A way of adjusting the weights in a neural network to minimize (or maximize) loss (or reward) function.
- Typically viewed as a convex optimization problem, finding the best setting of parameters W_i and b_i (i being the ith layer) to minimize the loss $\mathcal{L}$.
- Many optimizers to choose from, the most common is SGD, which performs batched updates to parameters.

Gradient descent parameter update:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} \mathcal{L}$$

Backpropagation: Notation

- Let $w^l_{j,k}$ denote the weight of node k in layer $l-1$ applied by node j in layer l
- b^l_j denote the bias of node j of layer l
- $z^l_j = \sum_k (x^{l-1}_k \cdot w^l_{j,k}) + b^l_j$
- a^l_j denotes the activation from applying a nonlinear function to node j in layer l.
- Sigmoid activation function: σ, where $a^l_j = \sigma(z^l_j) = \frac{1}{1+e^{-z^l_j}}$
- M: minibatch (subset) of input data samples
- $\delta^l_j = \frac{\partial \mathcal{L}}{\partial z^l_j}$: emperical error partial derivative at node j in layer l (which we use to update the parameters)
- L: Final layer

Backpropagation: Step

The loss changes with respect to the final layer's output, so we calculate the updates to the final layer using the chain rule:

$$
\begin{aligned}
\delta_j^L &= \frac{\partial \mathcal{L}}{\partial z_j^L} \\
&= \sum_k \frac{\partial \mathcal{L}}{\partial a_k^L} \cdot \frac{\partial a_k^L}{\partial z_j^L} \\
&= \frac{\partial \mathcal{L}}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} \\
&= \frac{\partial \mathcal{L}}{\partial a_j^L} \cdot \sigma'(z_j^L)
\end{aligned}
$$

Backpropagation: Step

Lastly, note that the error δ_j^l for any previous layer $l < L$ is

$$\begin{aligned}\delta_j^l &= \frac{\partial \mathcal{L}}{\partial z_j^l} \\ &= \sum_k \frac{\partial \mathcal{L}}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \delta_k^{l+1} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l}\end{aligned}$$

Finally, the general gradients for weights and biases are as follows:

- for weights: $\frac{\partial \mathcal{L}}{\partial w_{j,k}^l} = a_k^{l-1} \delta_j^l$
- for biases: $\frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l$

From this, we can calculate the gradients for all of the parameters in the model by propogating the loss signal from the final layer's parameters all the way to the parameters in the first hidden layer.

Backpropagation: Algorithm

1. $l_i = \|y^{(i)} - \tilde{x}^{(i)}\|_2^2$ is the emperical loss for element $x^{(i)}$ in minibatch M **Result of constructing forward pass computation graph**
2. Calculate errors δ for each layer, working backwards from last layer.
3. Repeat calculating all gradients for every element $x^{(i)}$.
4. Calculate the minibatch gradient update by $\frac{\sum_{x^{(i)} \in M} \nabla_\theta l_i}{|M|} \approx \nabla_\theta \mathcal{L}$ for all parameters θ (weights and biases).
5. Update all parameters θ at tth minibatch using gradient descent step:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} \mathcal{L}$$