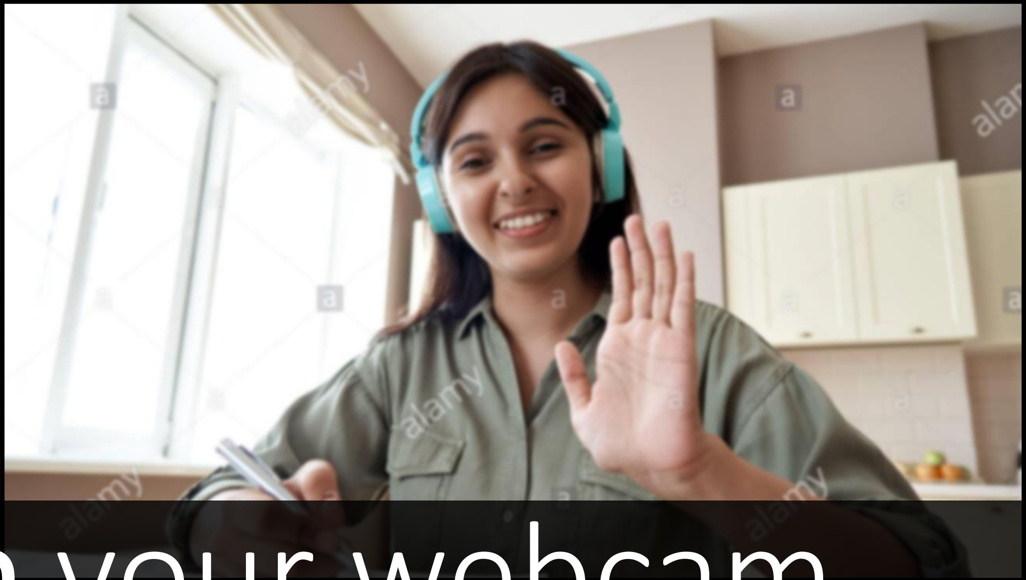
Deep Learning

10A

CS 3244
Machine Learning

NUS National University of Singapore | Computing

Please **turn on** your webcam



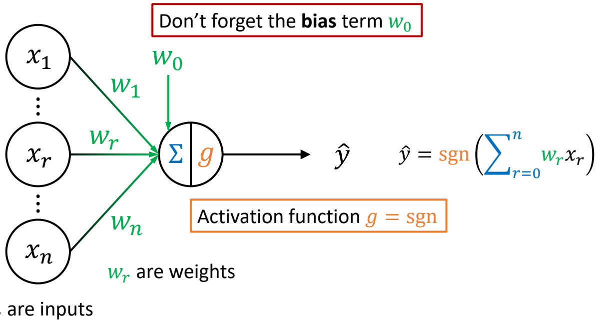
Mystery Student

Participate:
https://www.sustainabilityirl.synthesis.partners/

Mid-Semester Anonymous Survey

- What worked
  - Slack is good for comms & awareness
  - Exercises (pre-lecture, in-class) are engaging
  - Easier midterm (less stressful)!
- What to improve
  - Want more coding teaching: Bonus programming lectures to be conducted by Prof Min and TAs.
  - Want more examples: Included in lecture.
  - Somewhat out-of-scope tutorial questions: We will align tutorial questions more closely with lectures.

Perceptron



Perceptron Learning Algorithm

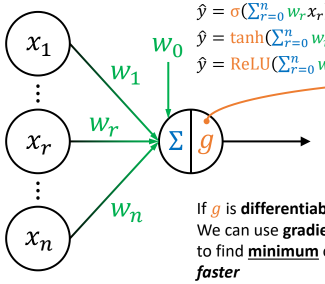
1. Initialize weights $\boldsymbol{w}$
   - Could be all zero, or random small values
2. For each instance i with features $\boldsymbol{x}^{(i)}$
   - Classify $\hat{y}^{(i)} = \mathrm{sgn}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)}\right)$
3. Select one misclassified instance
   - Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y - \hat{y})\boldsymbol{x}$
4. Iterate steps 2 to 3 until
   - Convergence (classification error < threshold), or
   - Maximum number of iterations

$$\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_r \\ \vdots \\ w_n \end{pmatrix} \leftarrow \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_r \\ \vdots \\ w_n \end{pmatrix} + \eta(y - \hat{y}) \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_r \\ \vdots \\ x_n \end{pmatrix}$$

$$w_r \leftarrow w_r + \eta(y - \hat{y})x_r$$

Differentiable Activation Functions

$\hat{y} = \mathrm{sgn}\left(\sum_{r=0}^{n} w_r x_r\right)$
$\hat{y} = \sigma\left(\sum_{r=0}^{n} w_r x_r\right)$
$\hat{y} = \tanh\left(\sum_{r=0}^{n} w_r x_r\right)$
$\hat{y} = \mathrm{ReLU}\left(\sum_{r=0}^{n} w_r x_r\right)$



If g is **differentiable**,
We can use **gradient descent** to find **minimum** of error $\varepsilon = y - \hat{y}$ ***faster***

Step
$\mathrm{sgn}(x) = \begin{cases} +1 & z > 0 \\ -1 & z \le 0 \end{cases}$

Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$

tanh
$\tanh(x)$

ReLU
$\max(0, x)$

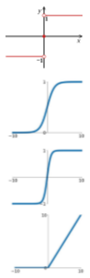
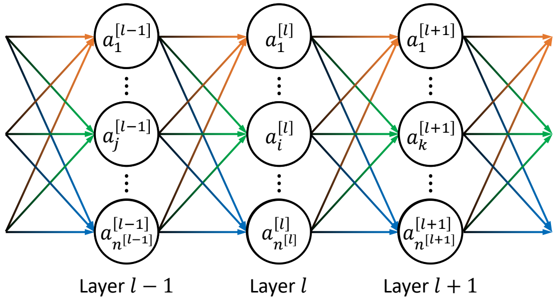


Image credit: https://miro.medium.com/max/1400/0*sIJ-gbjlz0zrz8lb.png

Multi-Layer Perceptron (Neural Network)



Chain Rule

Consider composite function

$$g(x) = g\big(f(x)\big)$$

$$g = g(f), f = f(x)$$

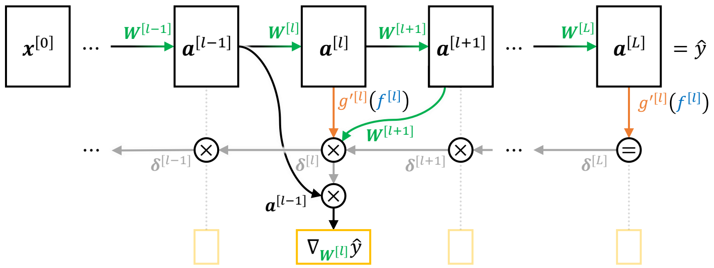
$$g'(x) = \frac{dg}{dx} = \frac{dg}{df}\frac{df}{dx}$$

Intuition
Rate of change of g relative to x is the product of
- rates of change of g relative to f and
- rates of change of f relative to x

"If
- *a car travels 2x fast as a bicycle and*
- *the bicycle is 4x as fast as a walking man,*

then the car travels 2 × 4 = 8 times as fast as the man."
– George F. Simmons, Calculus with Analytic Geometry (1985)

Backward Propagation



$$\hat{y}'\left(\boldsymbol{W}^{[l]}\right) = \boldsymbol{a}^{[l-1]}\left(\delta^{[l]}\right)^\top \qquad \delta^{[l]} = \left[g'^{[l]}\left(f^{[l]}\right)\right] \circ \left(\boldsymbol{W}^{[l+1]}\delta^{[l+1]}\right)$$

Notation

n = Number of features in $\boldsymbol{x}$
m = Number of instances in dataset

- **Scalar**: not bolded, lower case

$$x$$

- **Vector**: bolded, lower case

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

- **Matrix**: bolded, upper case

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix}$$

Functions with Vectors and Matrices

- Scalar-by-scalar:
  - $y(x) = wx$ for scaling input
- Scalar-by-vector:
  - $y(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} = \boldsymbol{w}^{\top}\boldsymbol{x} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = w_1 x_1 \cancel{+ w_1 x_2 + w_2 x_1} + w_2 x_2$ for **weighted sum**
- Vector-by-vector:
  - $\boldsymbol{y}(\boldsymbol{x}) = w\boldsymbol{x} = w\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} wx_1 \\ wx_2 \end{pmatrix}$ for **scaled** outputs (same weight)

Even more Chain Rule:

Gradient of Neural Network

$$\hat{y}(\boldsymbol{x}) = g^{[L]}(f^{[L]}(g^{[L-1]}(\cdots(g^{[l]}(f^{[l]}(g^{[l-1]}(\cdots(g^{[1]}(f^{[1]}(\boldsymbol{x}^{[0]})))))))))$$

Gradient relative to $\boldsymbol{W}$

$$\hat{y}'(\boldsymbol{W}^{[L\cancel{-1}]}) = \frac{\partial g^{[L]}}{\partial \boldsymbol{W}^{[L\cancel{-1}]}} = \frac{\partial f^{[L]}}{\partial \boldsymbol{W}^{[L]}} \boxed{\frac{\partial g^{[L]}}{\partial f^{[L]}}} \; \boldsymbol{\delta}^{[L\cancel{-1}]}$$

Reference

$$\boldsymbol{a}^{[l]} = g^{[l]}(f^{[l]})$$

$$f^{[l]} = \left(W^{[l]}\right)^{\top} \boldsymbol{a}^{[l-1]}$$

$$\hat{y}'(\boldsymbol{W}^{[l+1]}) = \frac{\partial g^{[L]}}{\partial \boldsymbol{W}^{[l+1]}} = \frac{\partial f^{[l+1]}}{\partial \boldsymbol{W}^{[l+1]}} \boxed{\frac{\partial g^{[l+1]}}{\partial f^{[l+1]}} \cdots \frac{\partial f^{[L]}}{\partial g^{[L-1]}} \frac{\partial g^{[L]}}{\partial f^{[L]}}} \; \boldsymbol{\delta}^{[l+1]}$$

$$\hat{y}'(\boldsymbol{W}^{[l]}) = \frac{\partial g^{[L]}}{\partial \boldsymbol{W}^{[l]}} = \frac{\partial f^{[l]}}{\partial \boldsymbol{W}^{[l]}} \frac{\partial g^{[l]}}{\partial f^{[l]}} \frac{\partial f^{[l+1]}}{\partial g^{[l]}} \underbrace{\frac{\partial g^{[l+1]}}{\partial f^{[l+1]}} \cdots \frac{\partial f^{[L]}}{\partial g^{[L-1]}} \frac{\partial g^{[L]}}{\partial f^{[L]}}}$$

Recursive

$$\hat{y}'(\boldsymbol{W}^{[l]}) = \frac{\partial g^{[L]}}{\partial \boldsymbol{W}^{[l]}} = \frac{\partial f^{[l]}}{\partial \boldsymbol{W}^{[l]}} \frac{\partial g^{[l]}}{\partial f^{[l]}} \frac{\partial f^{[l+1]}}{\partial g^{[l]}} \boldsymbol{\delta}^{[l+1]}$$

$$\hat{y}'(\boldsymbol{W}^{[1]}) = \frac{\partial g^{[L]}}{\partial \boldsymbol{W}^{[1]}} = \frac{\partial f^{[1]}}{\partial \boldsymbol{W}^{[1]}} \frac{\partial g^{[1]}}{\partial f^{[1]}} \cdots \frac{\partial g^{[l]}}{\partial f^{[l]}} \frac{\partial f^{[l+1]}}{\partial g^{[l]}} \frac{\partial g^{[l+1]}}{\partial f^{[l+1]}} \cdots \frac{\partial f^{[L]}}{\partial g^{[L-1]}} \frac{\partial g^{[L]}}{\partial f^{[L]}}$$

Week 10A: Learning Outcomes

1. Understand how deep learning enables better model performance than shallow machine learning
2. Explain how CNNs and RNNs are different from feedforward neural networks
3. Appropriately choose and justify when to use each architecture
4. Explain how to mitigate training issues in deep learning

Week 10A: Lecture Outline

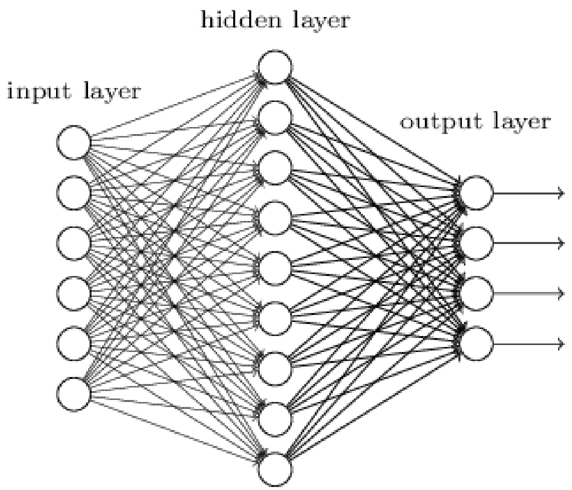
1. Deep learning motivation
2. Popular Architectures
    1. Convolutional Neural Networks
    2. Recurrent Neural Networks
3. Deep learning training issues

Deep Neural Network

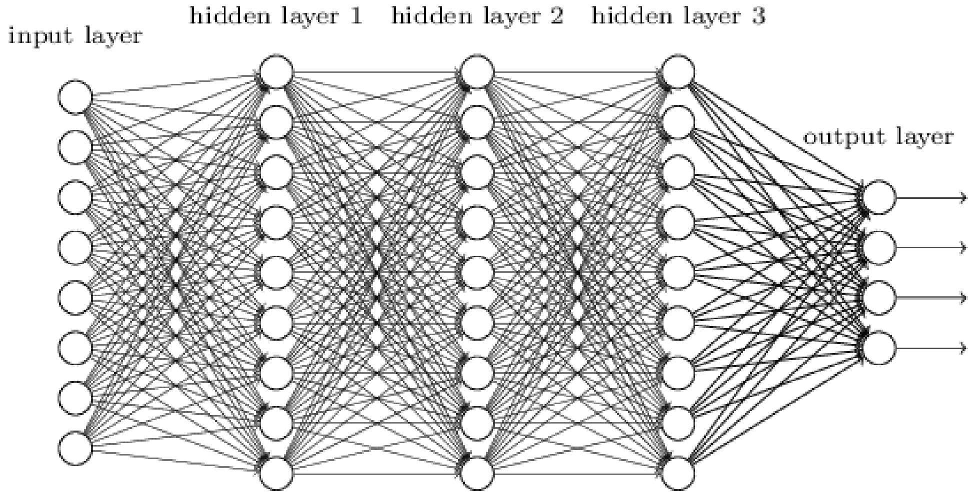
Department of Computer Science
School of Computing

Deep Neural Network = many hidden layers (≥3)

Shallow Network



Deep Network



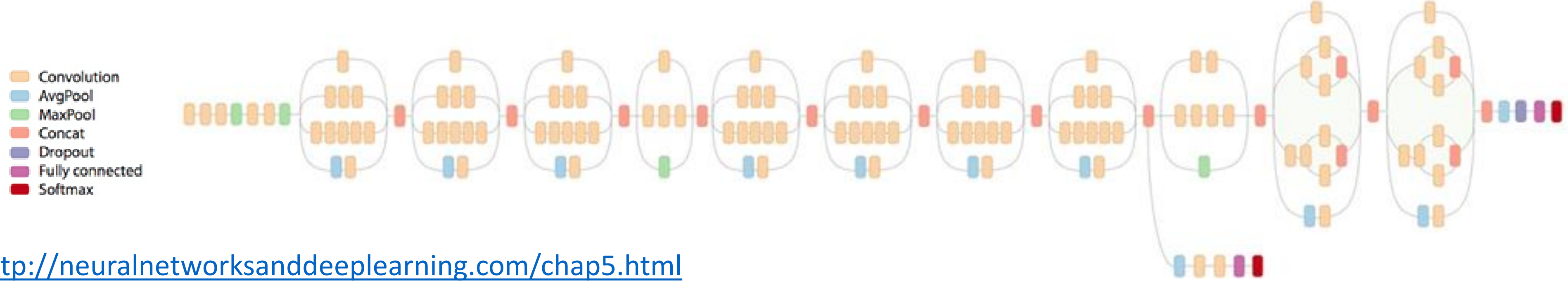
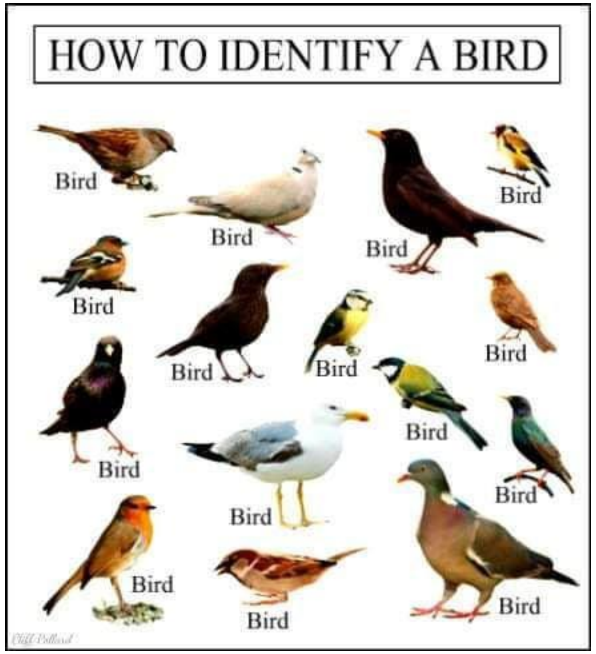


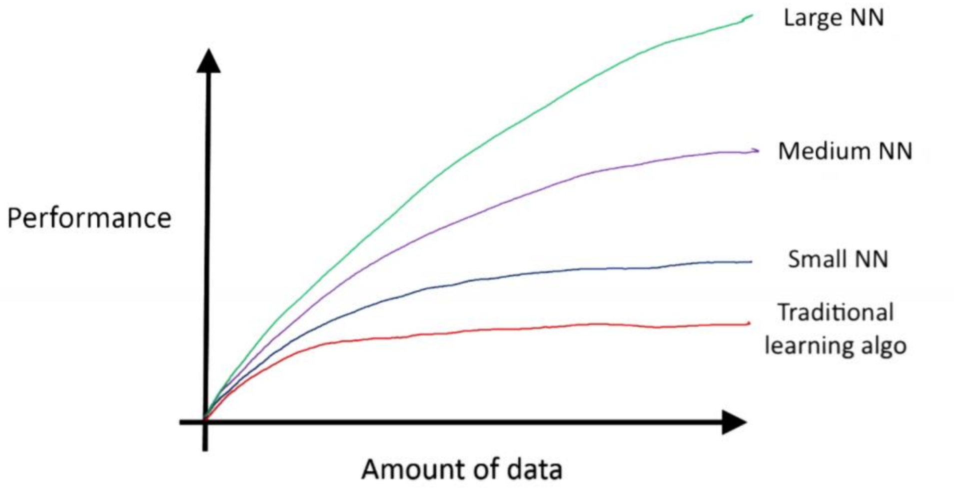
Image credit: http://neuralnetworksanddeeplearning.com/chap5.html
https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

Why Deep?

- Why need so many layers?
  - Need **many parameters**
  - Target functions of real-world tasks are **complex**
  - E.g., what is the function for recognizing birds or language?
- Why need so much training data?
  - Many parameters → Need more data
  - More data → Better performance

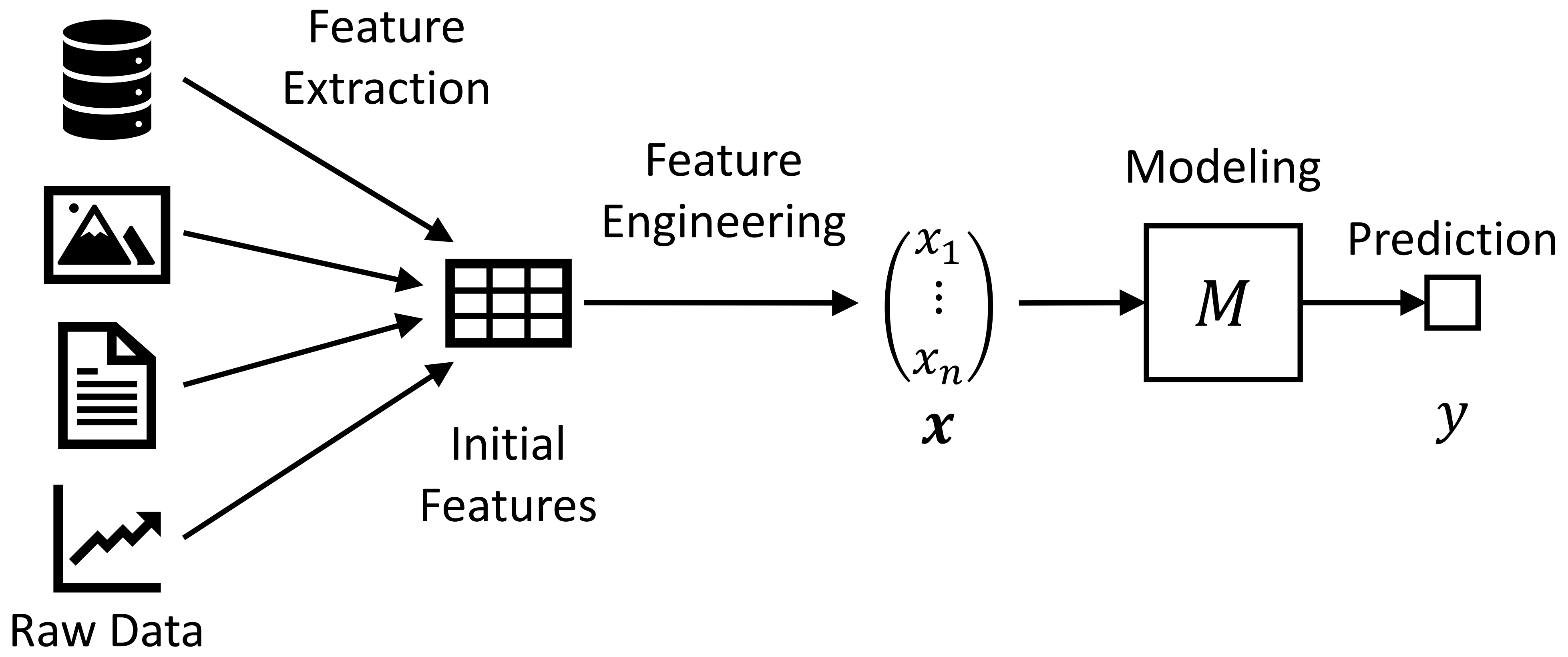


https://9gag.com/gag/ax9Roon

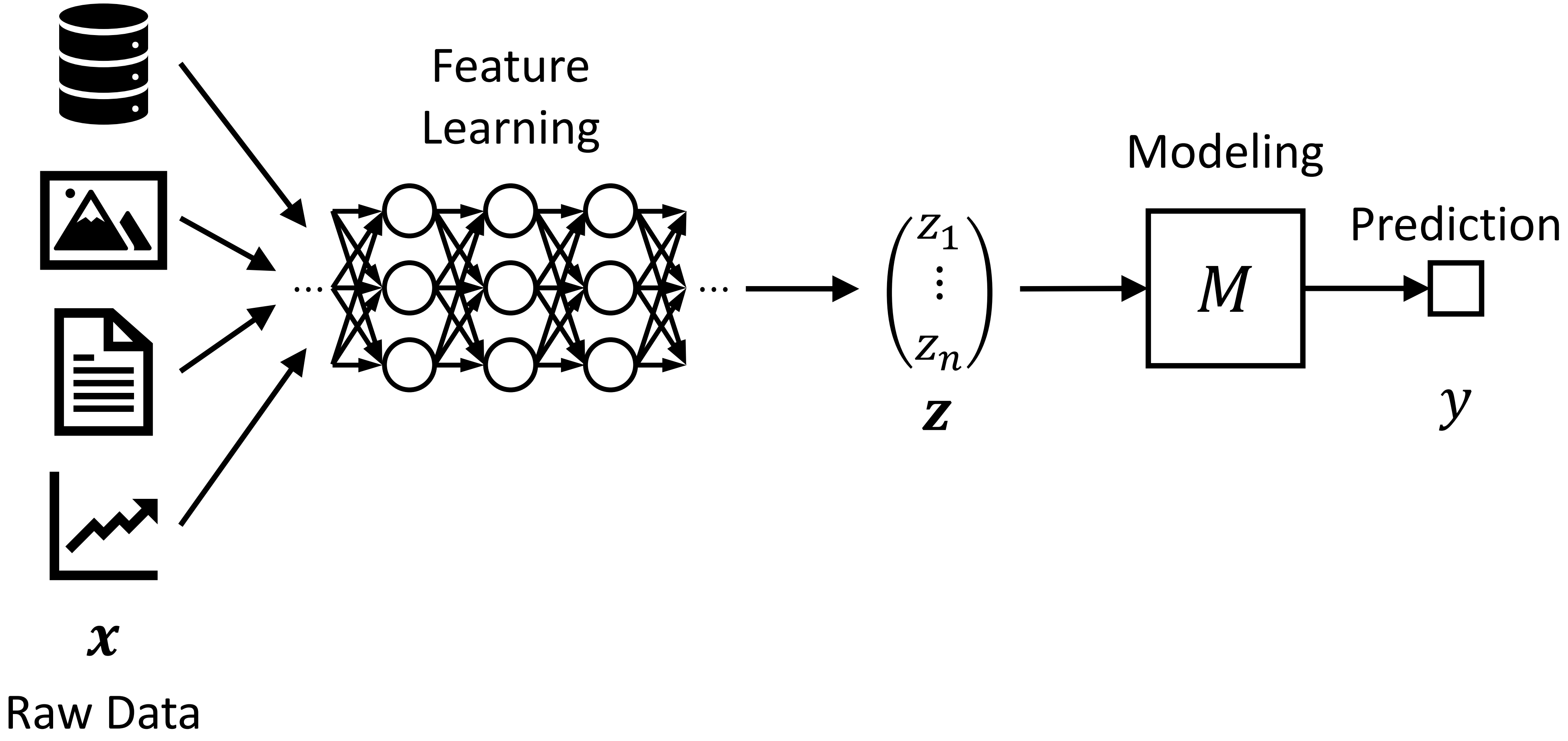


Andrew Ng https://youtu.be/LcfLo7YP8O4

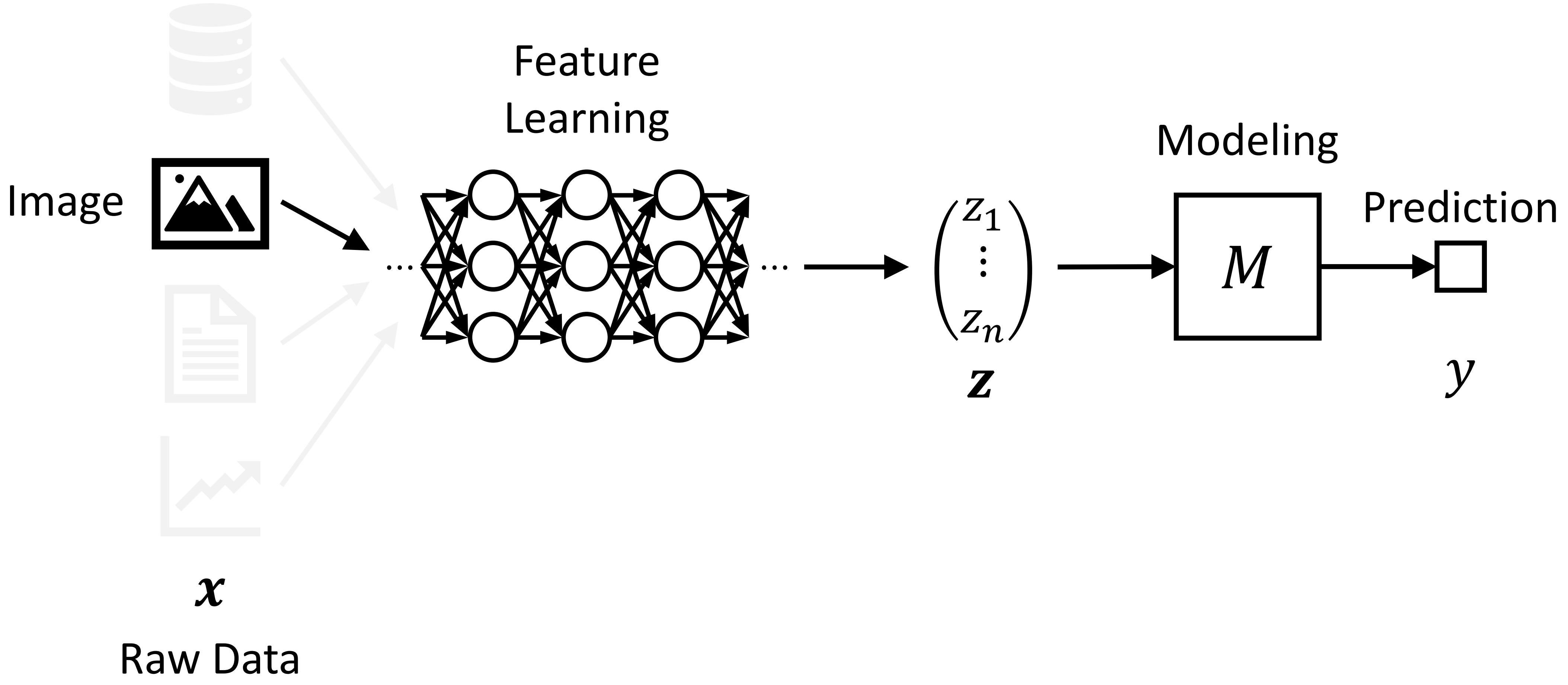
Feature Extraction/Engineering → Modeling



From Manual Feature Engineering To Automatic Feature Learning



From Manual Feature Engineering To Automatic Feature Learning



Convolutional Neural Networks (CNN)

NUS National University of Singapore | Department of Computer Science School of Computing

Applications of CNN

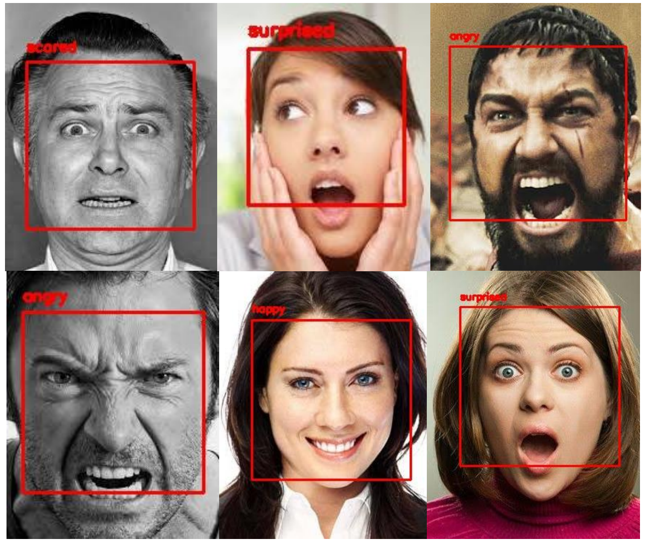
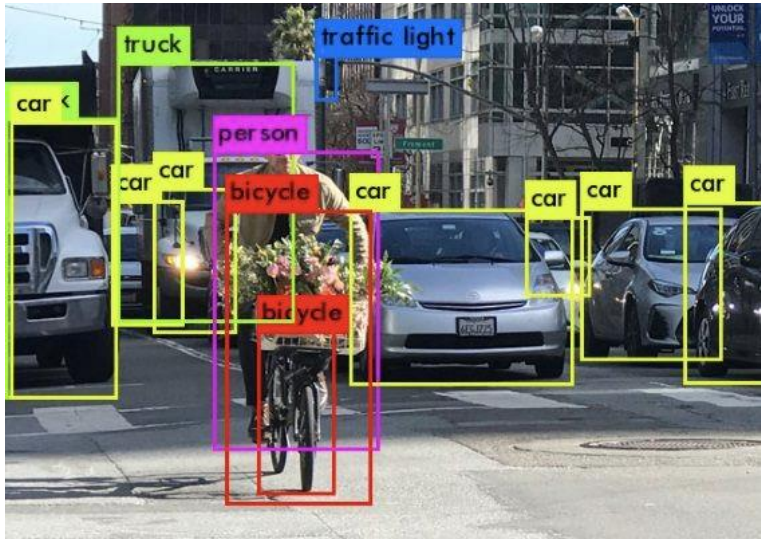


Image Classification
e.g., face emotions



Object Detection
e.g., self-driving cars

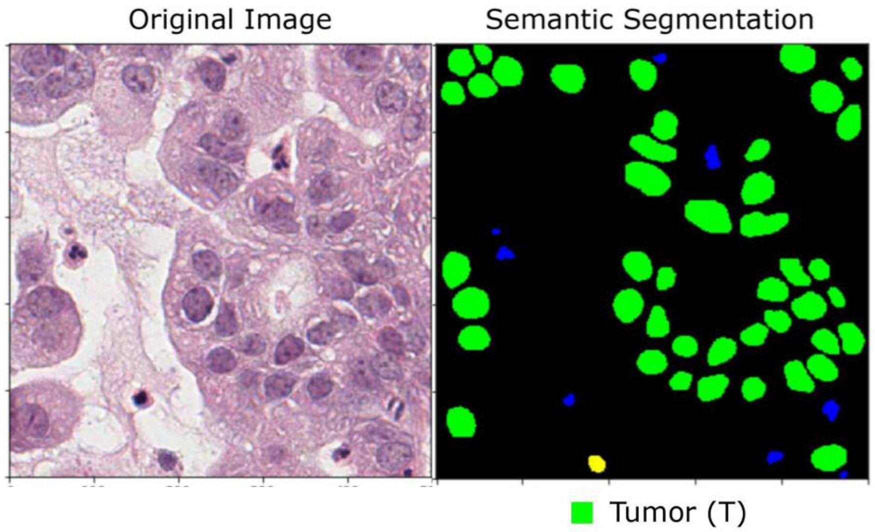


Image Segmentation
e.g., cancer cell detection

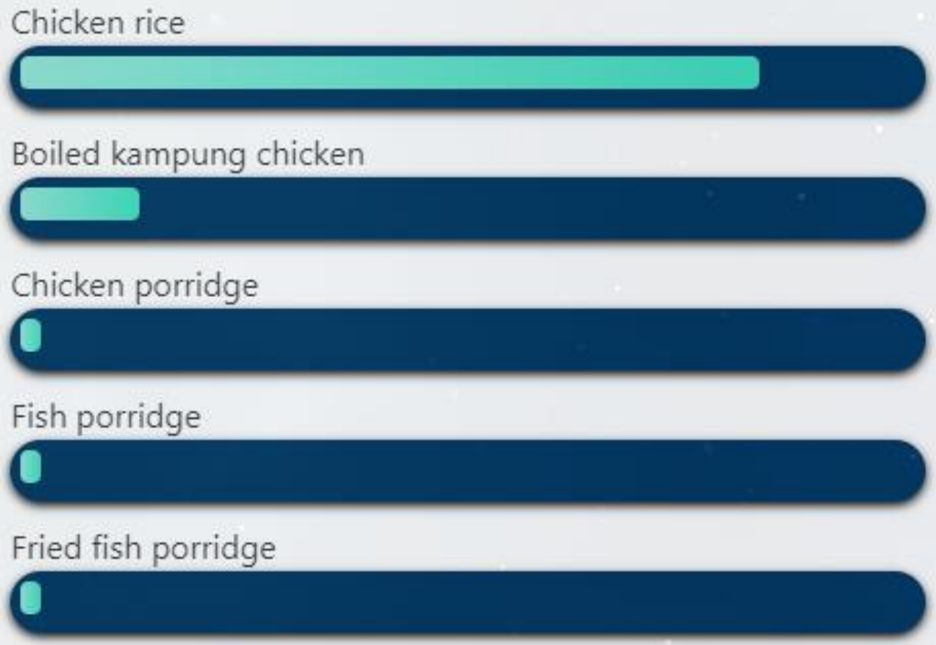
Image credit:
https://monica-dommaraju.medium.com/analysis-of-deep-learning-based-object-detection-f14d5138148
https://ajp.amjpathol.org/article/S0002-9440(18)31121-0/fulltext
https://appliedmachinelearning.blog/2018/11/28/demonstration-of-facial-emotion-recognition-on-real-time-video-using-cnn-python-keras/

FOODAi Demo

https://foodai.org

Try out our demo below or visit our developer portal for our API services.

To try our demo, you can click the upload icon to choose the image, or copy and paste the image or drag and drop the image from desktop or internet to the upload area.



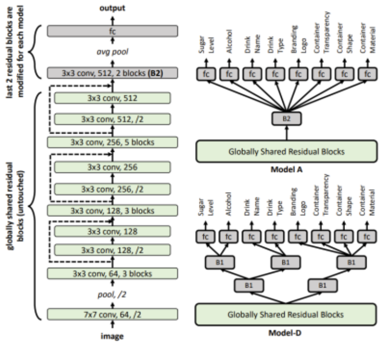
Register for FoodAI API Free Trial

Research (not in exam)

tastehealthy

Backend Models

Food & Drink Recognition



Food Recommendation

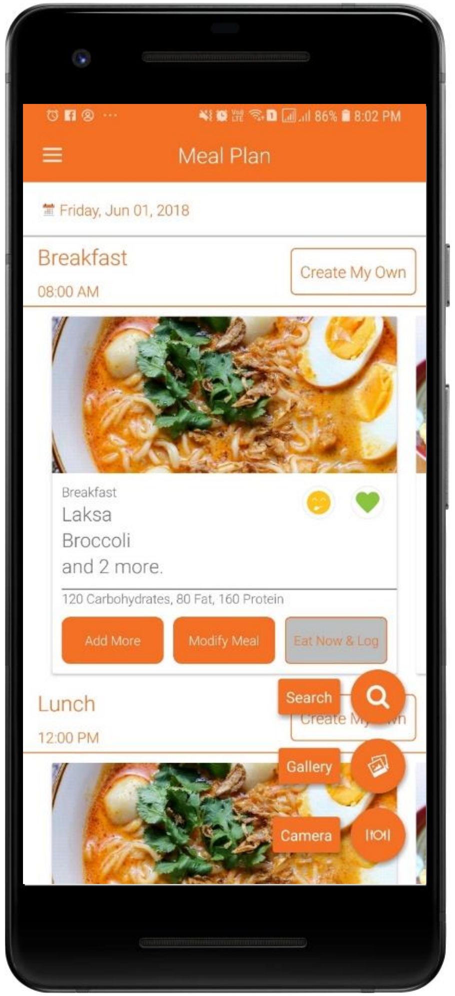
RecGAN

Generator

$$
\begin{aligned}
x_t^u &= RELU(W_{xh}^u h_{t-1}^u + W_{xk}^u y_t^u) \\
r_t^u &= \sigma(W_{rh}^u h_{t-1}^u + W_{rk}^u y_t^u) \\
m_t^u &= tanh(W_h(r_t^u \cdot h_{t-1}^u) + W_{input} y_t^u) \\
h_t^u &= (1 - x_t^u) \cdot h_{t-1}^u + x_t^u \cdot m_t^u
\end{aligned}
$$

Discriminator

$$
\begin{aligned}
\hat{x}_t^u &= RELU(V_{xh}^u \hat{h}_{t-1}^u + V_{xk}^u y_t^u) \\
\hat{r}_t^u &= \sigma(V_{rh}^u \hat{h}_{t-1}^u + V_{rk}^u y_t^u) \\
\hat{m}_t^u &= tanh(V_h(\hat{r}_t^u \cdot \hat{h}_{t-1}^u) + V_{input} y_t^u) \\
\hat{h}_t^u &= (1 - \hat{x}_t^u) \cdot \hat{h}_{t-1}^u + \hat{x}_t^u \cdot \hat{m}_t^u
\end{aligned}
$$

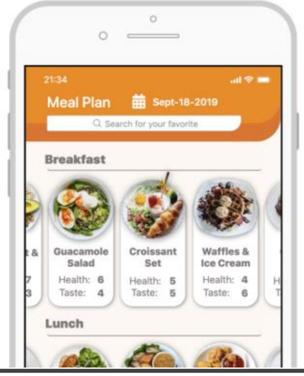


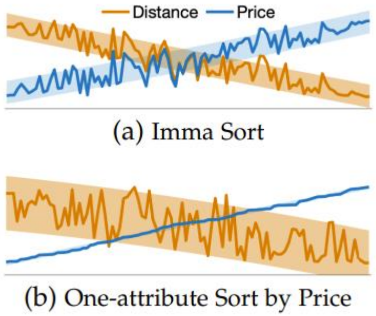
Frontend UI/UX

Family Food Logging

TableChat

Multi-Attribute Sorting





Lyu, Y., Gao, F., Wu, I. S., & Lim, B. Y. 2020. Imma Sort by Two or More Attributes With Interpretable Monotonic Multi-Attribute Sorting. TVCG.
Park, H., Bharadhwaj, H., and Lim, B. Y. 2019. Hierarchical Multi-Task Learning for Healthy Drink Classification. *IJCNN.*
Bharadhwaj, H., Park, H., Lim, B. Y.. 2018. RecGAN: Recurrent Generative Adversarial Networks for Recommendation Systems. *RecSys '18.*
Lukoff, K., Li, T., Zhuang, Y., & Lim, B. Y. 2018. TableChat: Mobile Food Journaling to Facilitate Family Support for Healthy Eating. *CSCW '18.*

NUS Ubicomp Lab
Apps and Analytics for Smart Cities and Health

Images as 2D matrices

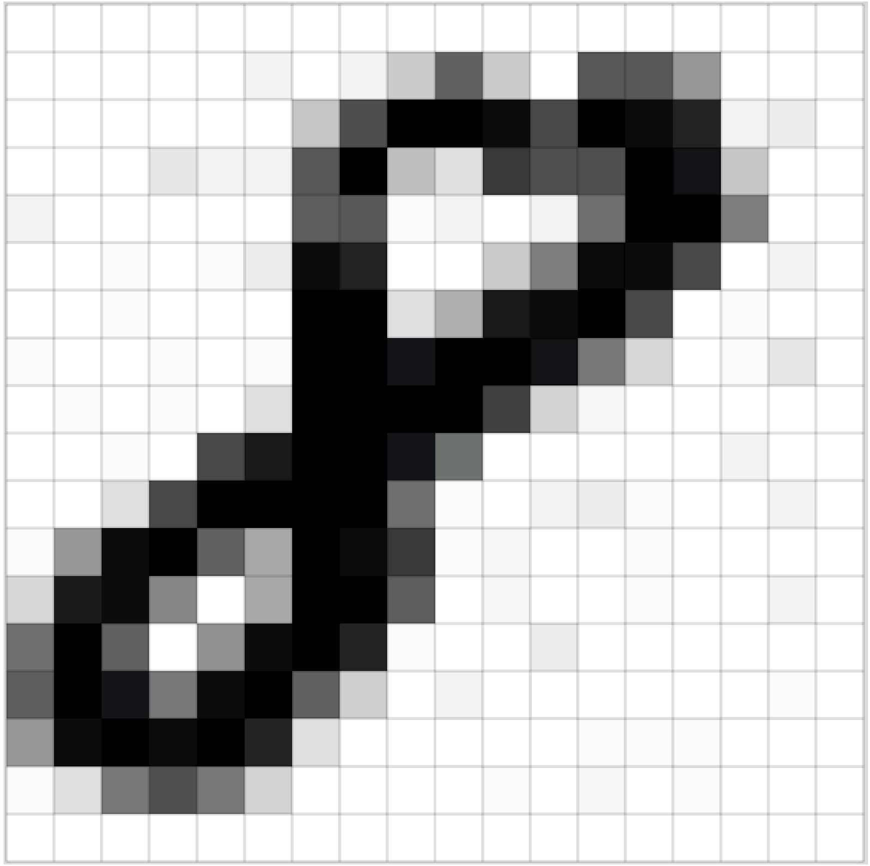
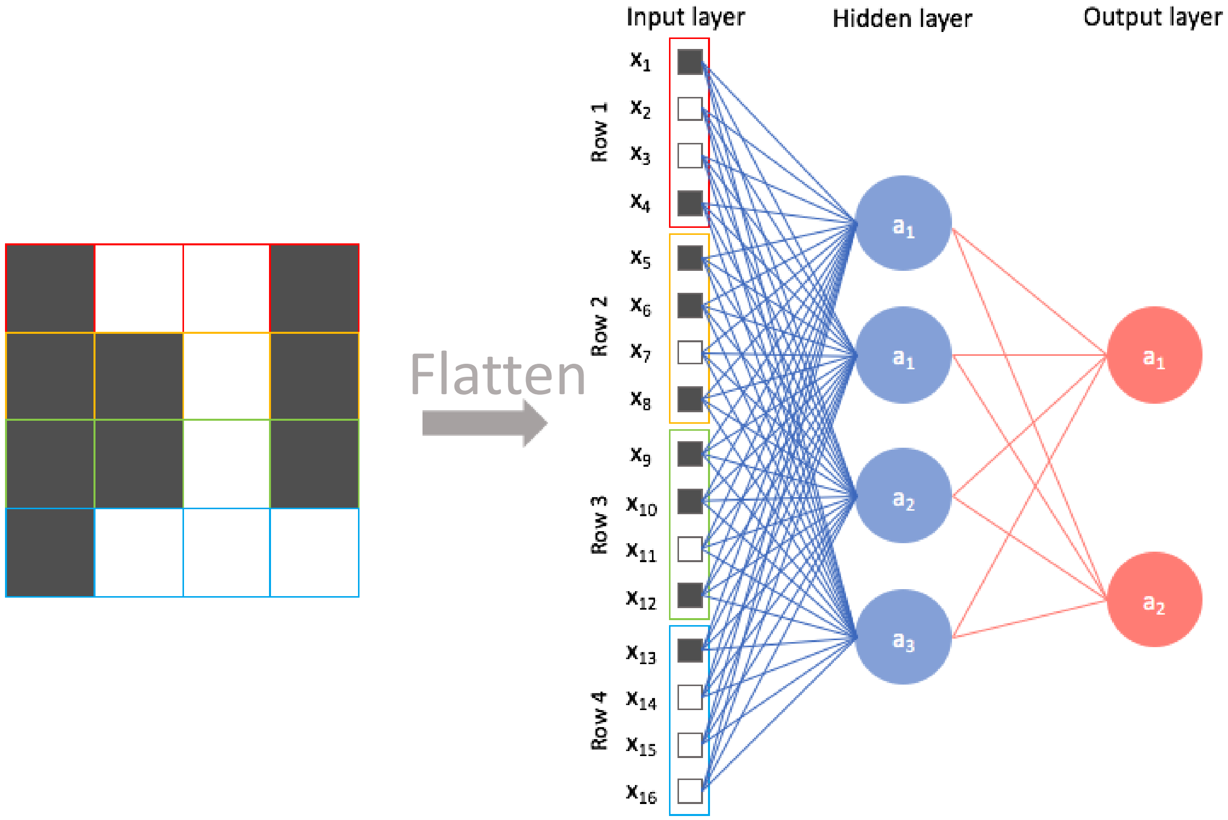
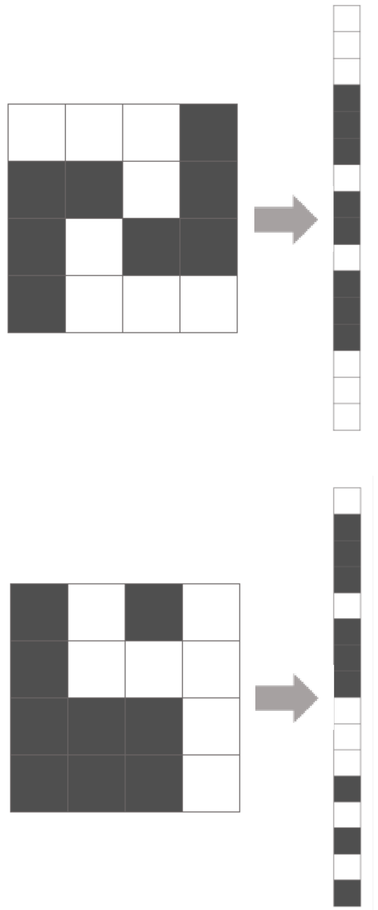


Image credit

Image Feature Extraction with
Fully Connected Neural Networks (Multi-Layer Perceptron)





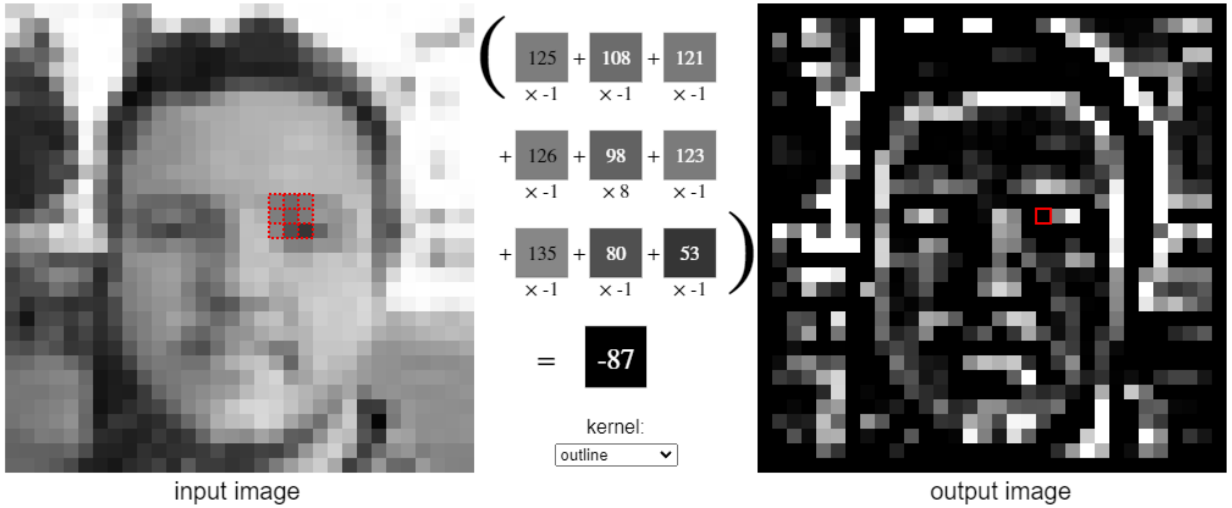
Reduce parameters for images:
Exploit **Spatial** Relations with **Convolutions**

Let's walk through applying the following 3x3 **outline** kernel to the image of a face from above.

outline

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

Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.



Manually finding good filters is tedious

Further study:
https://setosa.io/ev/image-kernels/

High-level Feature Detection

Eyes, Nose, Mouth
Facial Hair

Wheels, Headlights,
Bonnet/Hood

Fish, Rice,
Vegetables

How to **automatically** learn these features?

Feature Detectors: Intuition of Neuron Kernels in Layers

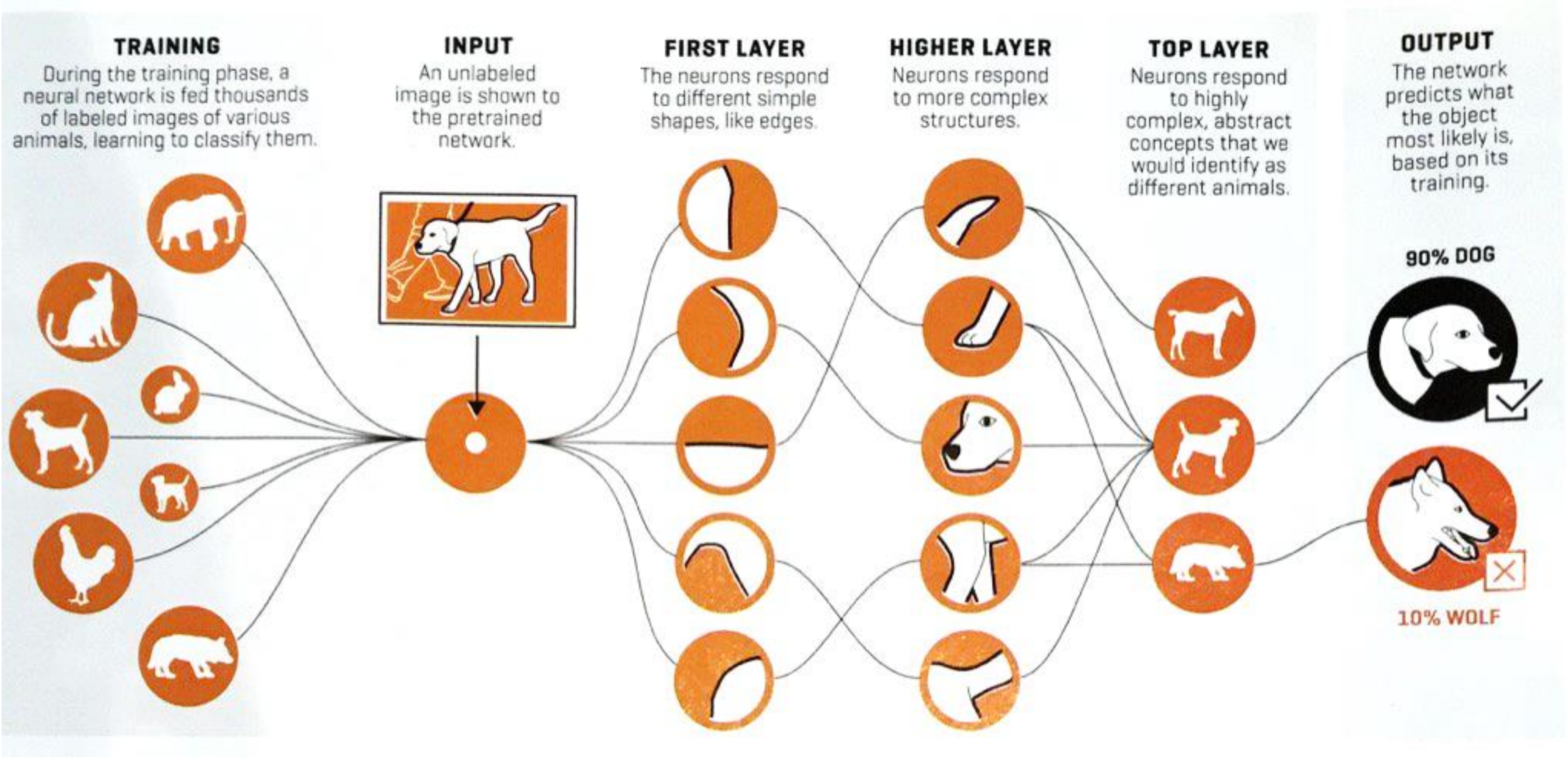


Image credit: https://fortune.com/longform/ai-artificial-intelligence-deep-machine-learning/

Analogy: activations of different <u>filters</u> learned by CNNs is like seeing the image through different lens filters

Image credit: https://www.amazon.com/Godefa-Samsung-Andriod-Smartphone-Universal/dp/B07RQRLQYH
https://www.yankodesign.com/2020/02/17/this-retro-inspired-camera-records-dreamy-looking-gifs-that-replicate-vintage-8mm-film/

Convolutions: Kernel Size, Stride, Padding

$$W = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad x = \begin{pmatrix} 9 & 9 & 3 & 3 & 4 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 3 & 3 & 5 & 5 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 9 & 3 & 3 & 4 \end{pmatrix} \quad W * x = \begin{pmatrix} -6-6-6 & -6+1+2 & 1+2+2 \\ -6-6-6 & 1+2+1 & 2+2+2 \\ -6-6-6 & 2+1-6 & 2+2+1 \end{pmatrix}$$

<u>Stride $s \neq 1$</u>

$$W = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad x = \begin{pmatrix} 9 & 9 & 3 & 3 & 4 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 3 & 3 & 5 & 5 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 9 & 3 & 3 & 4 \end{pmatrix} \quad W * x = \begin{pmatrix} -6-6-6 & 1+2+2 \\ -6-6-6 & 2+2+1 \end{pmatrix}$$

<u>Padding $p \neq 0$</u>

$$W = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad x = \begin{pmatrix} 3 & 3 & 4 \\ 3 & 3 & 5 \\ 3 & 3 & 4 \end{pmatrix} \quad W * x = \begin{pmatrix} -6-6-6 & 1+2+2 \\ -6-6-6 & 2+2+1 \end{pmatrix}$$

What are the Kernel Size, Stride, Padding?

$$\boldsymbol{W} = \begin{pmatrix} -1 & 0 \\ -1 & 1 \\ 0 & 1 \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} 9 & 9 & 9 & 3 & 3 & 4 \\ 9 & 9 & 3 & 5 & 5 & 8 \end{pmatrix} \quad \boldsymbol{y} = \boldsymbol{W} * \boldsymbol{x} = \begin{pmatrix} 0+0+9 & 0+0+3 & 0+0+4 \\ 0+0+9 & 0-6+5 & 0+1+8 \\ -9+0+0 & -9+2+0 & -3+3+0 \\ -9+0+0 & -3+0+0 & -5+0+0 \end{pmatrix}$$

In Slack #general
1. **Write** answer to thread
    1. Kernel Size = ?
    2. Stride = ?
    3. Padding = ?
2. **Emote** (👍 :+1:) to vote for answer

What are the Kernel Size, Stride, Padding?

$\{height \times width\}$ $\quad \dim \boldsymbol{x} = \{2 \times 6\}$ $\quad \dim \boldsymbol{y} = \{4 \times 3\}$

$$\boldsymbol{W} = \begin{pmatrix} -1 & 0 \\ -1 & 1 \\ 0 & 1 \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} 9 & 9 & 9 & 3 & 3 & 4 \\ 9 & 9 & 3 & 5 & 5 & 8 \end{pmatrix}$$

$h_p/2$

$h_p/2$

w_s

$$\boldsymbol{y} = \boldsymbol{W} * \boldsymbol{x} = \begin{pmatrix} 0+0+9 & 0+0+3 & 0+0+4 \\ 0+0+9 & 0-6+5 & 0+1+8 \\ -9+0+0 & -9+2+0 & -3+3+0 \\ -9+0+0 & -3+0+0 & -5+0+0 \end{pmatrix}$$

Hyperparameters

- Kernel size $\boldsymbol{\kappa} = \{3 \times 2\}$
- Padding $\boldsymbol{p} = \{(2+2) \times 0\}$
- Stride $\boldsymbol{s} = \{1 \times 2\}$

Chosen manually, or automatically with [hyperparameter tuning]

$$\dim \boldsymbol{y} = \left\{\left(\frac{h_x + h_p - h_\kappa + h_s}{h_s}\right) \times \left(\frac{w_x + w_p - w_\kappa + w_s}{w_s}\right)\right\}$$

$$= \left\{\left(\frac{2+4-3+1}{1}\right) \times \left(\frac{6+0-2+2}{2}\right)\right\}$$

Convolutional Layer

What's the differences between the left and right expressions?

Convolution of each **Feature Kernel** $\boldsymbol{W}^{[l]}$ produces a **Feature Map** $= \boldsymbol{W}^{[l]} * \boldsymbol{A}^{[l-1]}$

Weights = **Kernels** (3D matrix)

2D Convolution (height × width)

$$\boldsymbol{a}^{[l]} = g^{[l]}\left(\left(\boldsymbol{W}^{[l]}\right)^{\top}\boldsymbol{a}^{[l-1]}\right) \quad \text{vs.} \quad \boldsymbol{A}^{[l]} = g^{[l]}\left(\boldsymbol{W}^{[l]} * \boldsymbol{A}^{[l-1]}\right)$$

Activation function can be the same

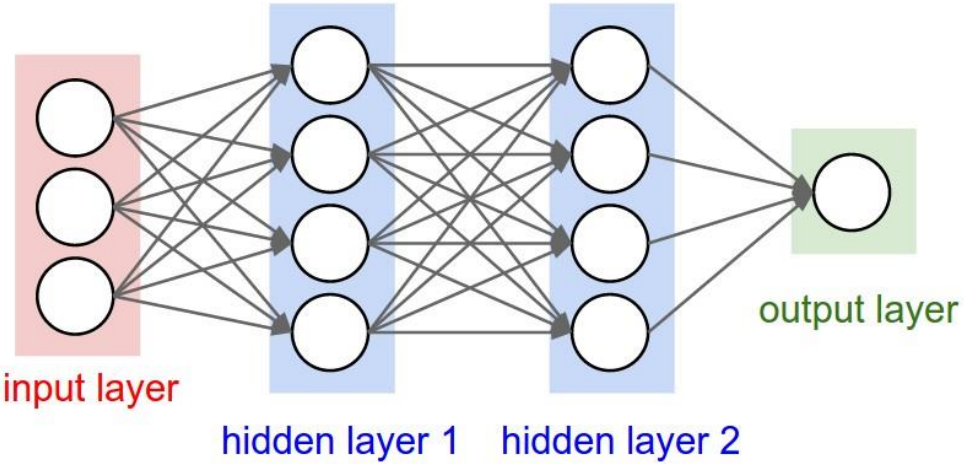
Activation is a 3D Matrix!
Concatenation of Feature Maps

Convolutional Layers

Remember: each kernel is like a different lens filter

Fully Connected Layers

- Each layer has multiple **neurons**
- Neuron output: **0D scalar** activation
- Neuron input: **1D vector** of activations
    - Each *element* is a different neuron
- Each layer is a **1D vector**



Convolutional Layers

- Each layer has multiple **kernels**
- Kernel output: **2D matrix** feature map
- Kernel input: **3D matrix** of feature maps
    - Each *depth position* is a different kernel
    - Analogy: filters are "stacked" together
- Each layer is a **3D matrix**

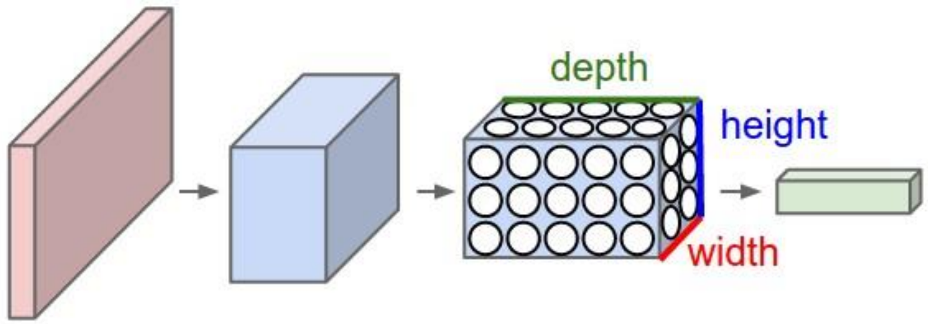
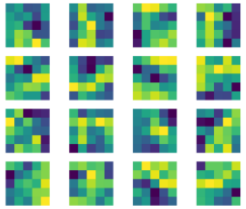


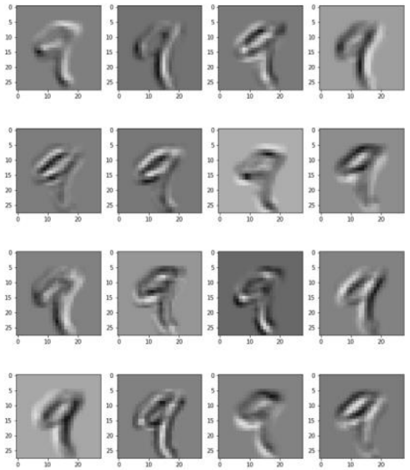
Image credit: https://cs231n.github.io/convolutional-networks/

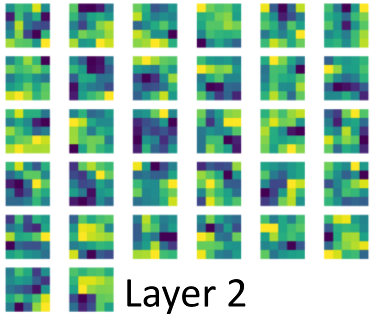
Convolutional Layer:
Feature Kernels & Feature Maps

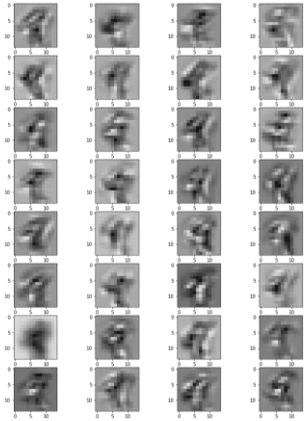
$$\boldsymbol{X}^{[0]} \rightarrow g^{[1]}\left(\boldsymbol{W}^{[1]} * \boldsymbol{X}^{[0]}\right) = \boldsymbol{A}^{[1]} \xrightarrow{\text{Pooling}} g^{[2]}\left(\boldsymbol{W}^{[2]} * \boldsymbol{X}^{[1]}\right) = \boldsymbol{A}^{[2]}$$


Input


Layer 1 Feature Kernels


Layer 1
Feature Maps


Layer 2
Feature Kernels


Layer 2
Feature Maps

Hyperparameters
1. Number of kernels k
2. Kernel size $\boldsymbol{\kappa}$
3. Padding $\boldsymbol{p}$
4. Stride $\boldsymbol{s}$

Chosen manually, or automatically with hyperparameter tuning

Kernels are learned *automatically* through **weight updates**.

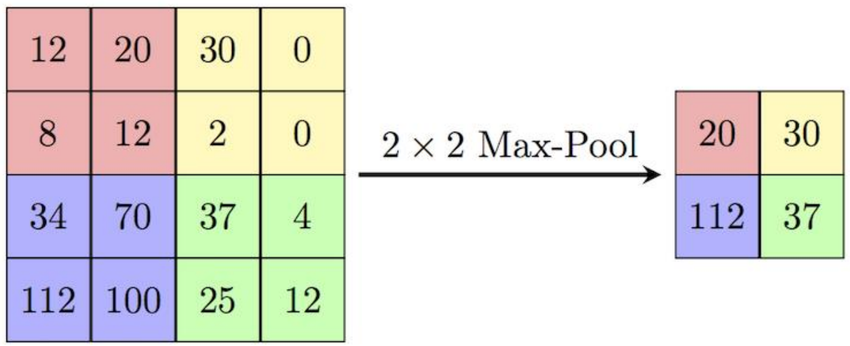
Interpretability: do you know what these kernel mean?

Image credit: https://medium.com/dataseries/visualizing-the-feature-maps-and-filters-by-convolutional-neural-networks-e1462340518e

Pooling Layer

- **Downsamples** Feature Maps
- Helps to train later kernels to detect **higher-level** features
- Reduces **dimensionality**
- Aggregation methods
  - Max-Pool (most used)
  - Average-Pool
  - Sum-Pool

Calculation



Example

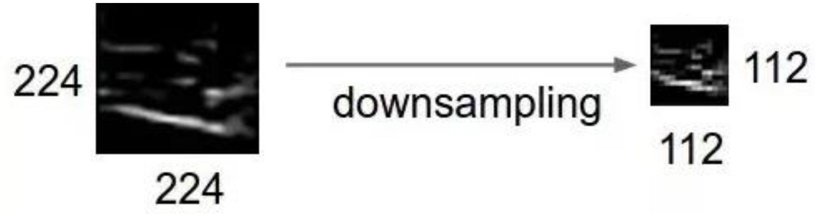
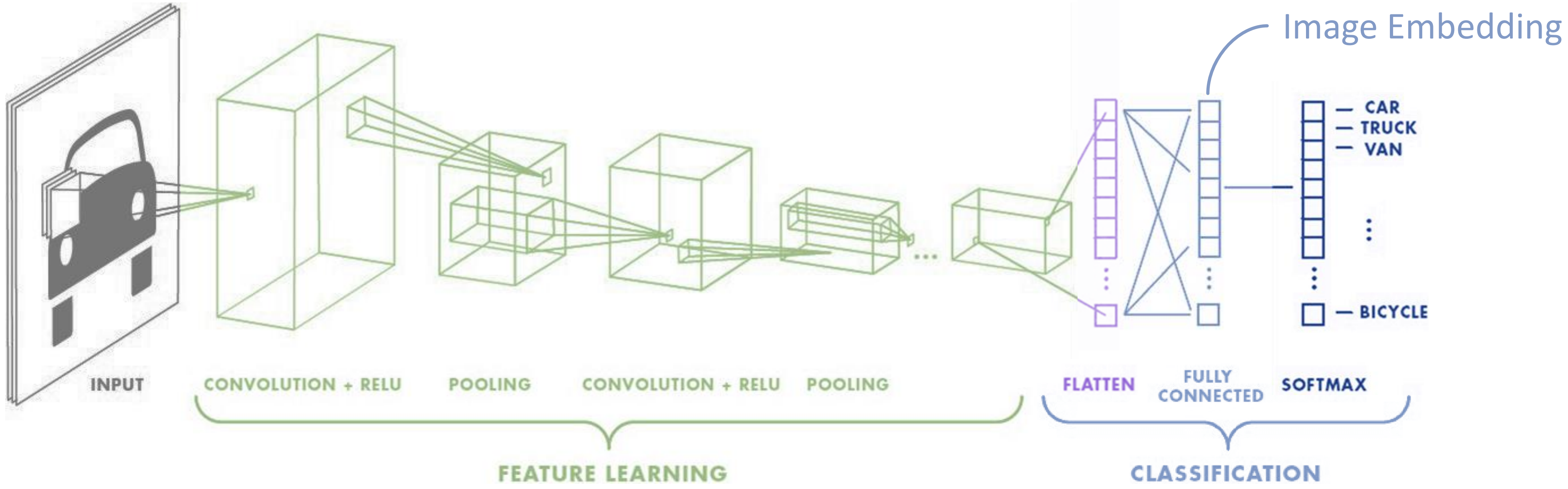


Image credit: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

Convolutional Neural Network



Key concepts

❶ Learn Spatial Feature
- Series of multiple convolution + pooling layers
- Progressively learn more diverse and higher-level features

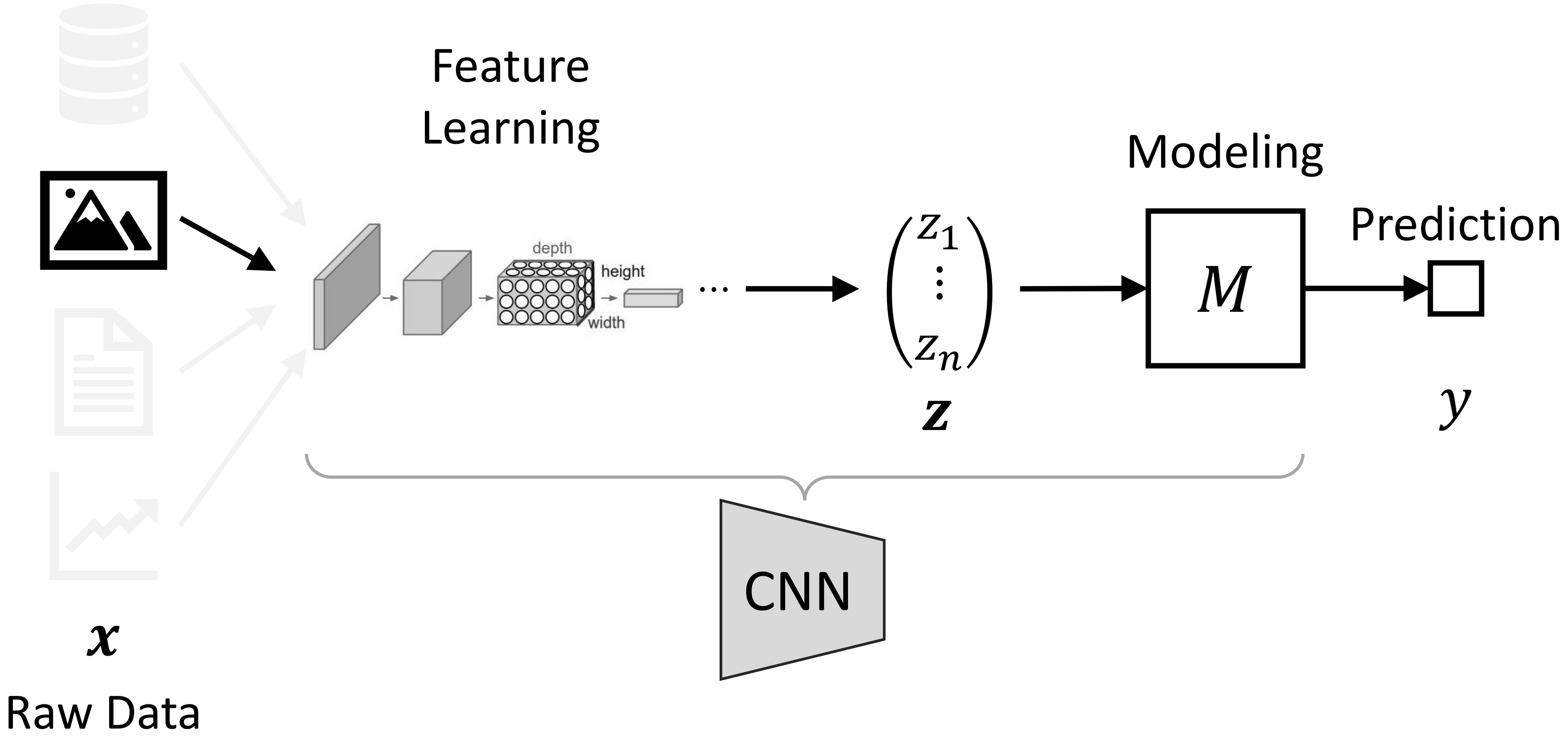
❷ Flattening
- Convert to fixed-length 1D vector

❸ Learn Nonlinear Features
- With fully connected layers (regular neurons)
- Learns nonlinear relations with multiple layers

❹ Classification
- Softmax ≔ Multiclass Logistic Regression
- Feature input = image embedding vector (typically large vector)

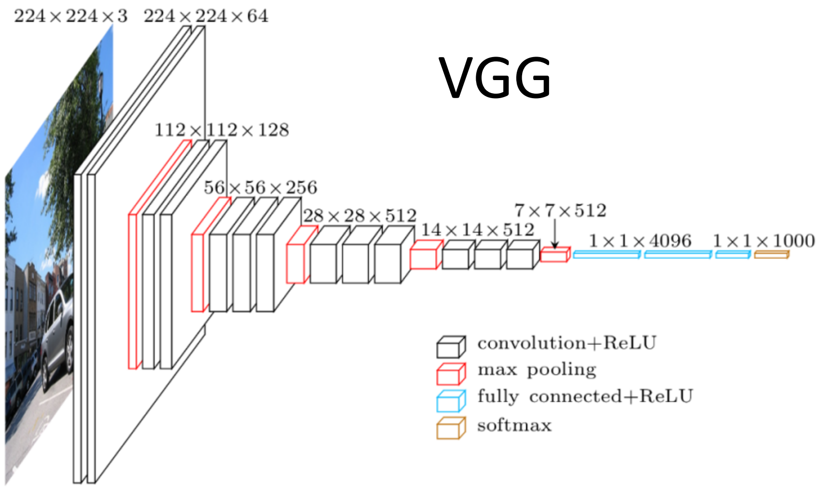
Image credit: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

From Manual Feature Engineering To Automatic Feature Learning

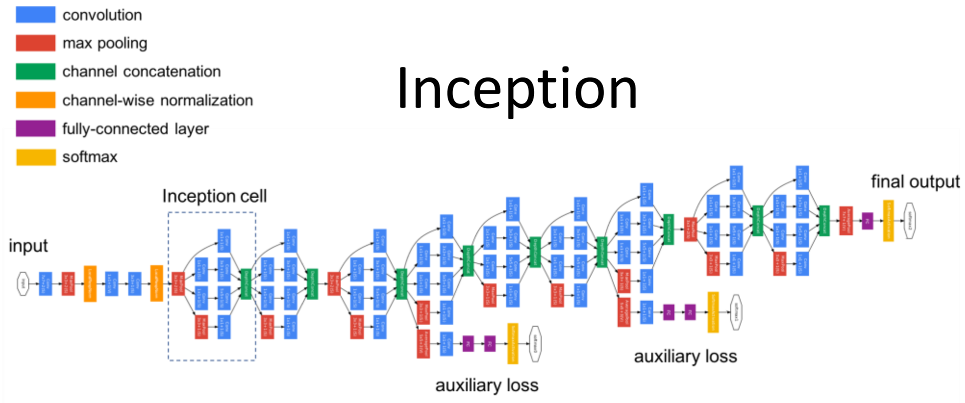


Other popular CNN architectures

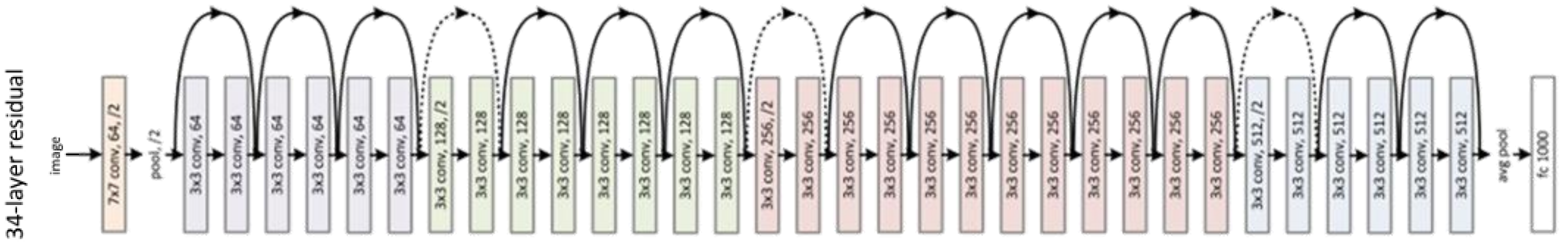
VGG



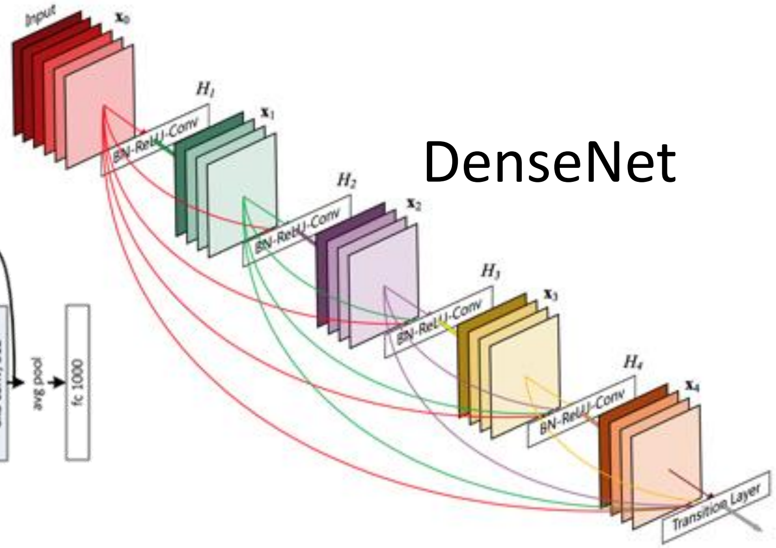
Inception



ResNet

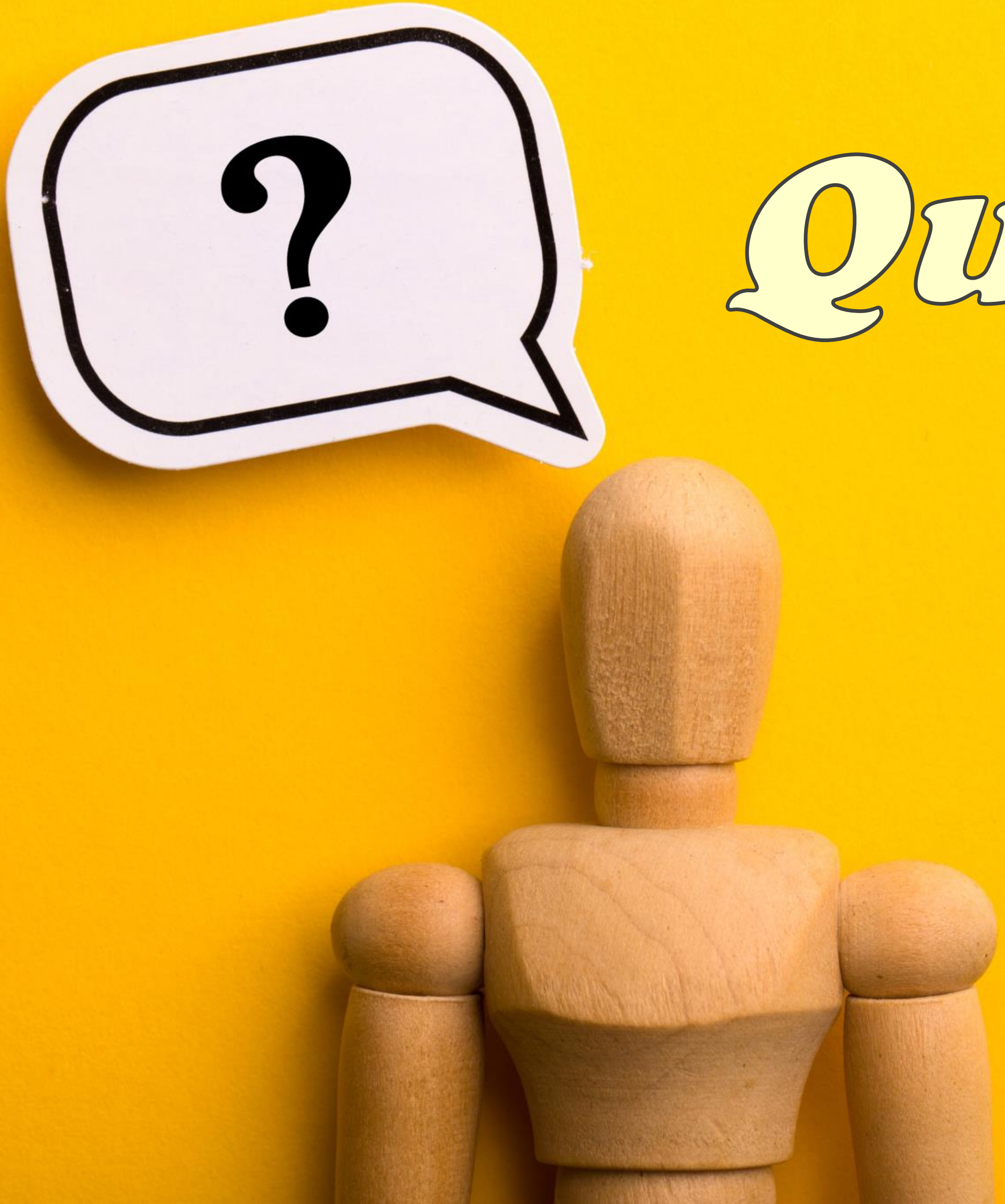


DenseNet

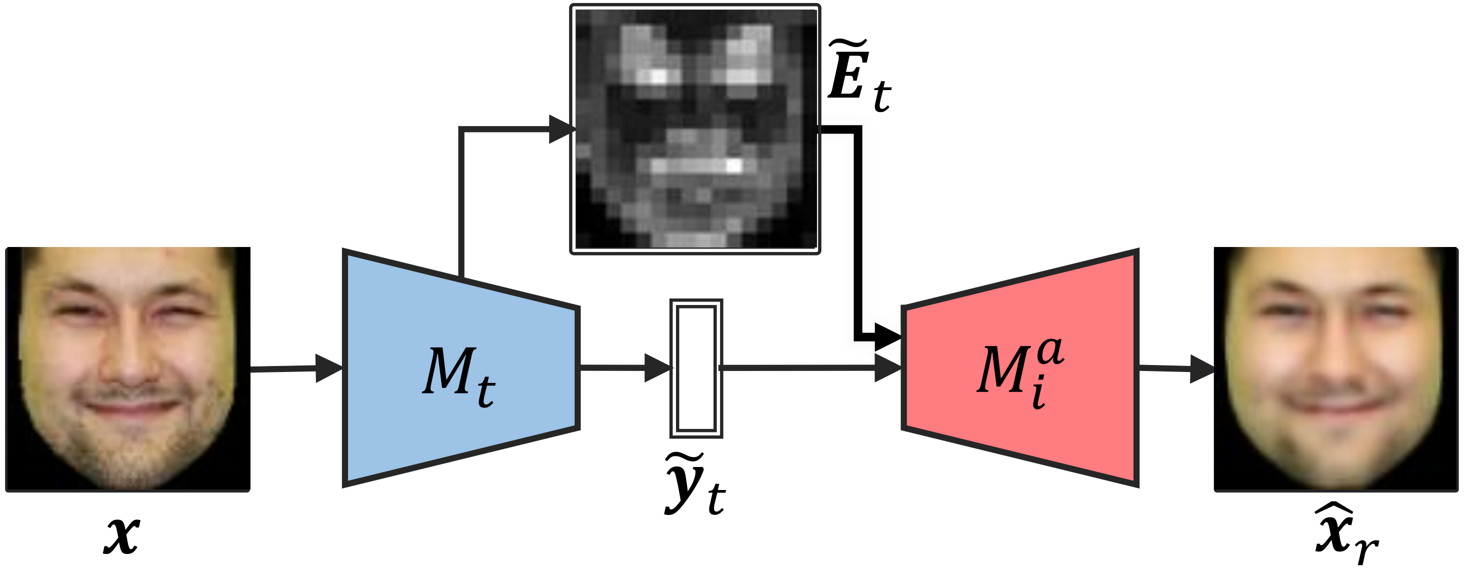


Further reading: https://www.jeremyjordan.me/convnet-architectures/

Questions!



Model Inversion Attack:
Predicting Images from Classification Vector



Model inversion attacks can reconstruct private face photos from prediction vectors only.

Model explanations can **worsen** model inversion attacks.

Zhao, X., Zhang, W., Xiao, X., & Lim, B. Y. (2021). Exploiting Explanations for Model Inversion Attacks. *ICCV 2021*.