A (Wild) Introduction to Deep Learning

Periklis Petridis & Vassilina Stoumpou

15.S60, 2024

Special Thanks to Leonard Boussioux and Angelos Koulouras!

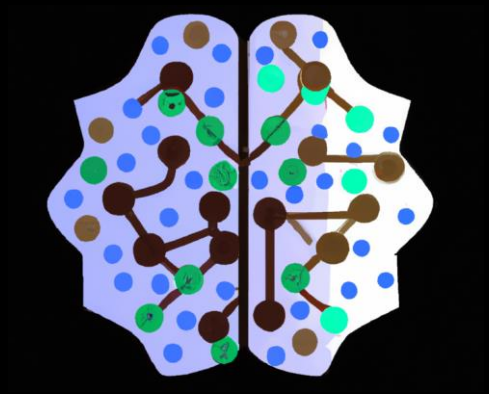


Image by DALLE

Warmup Activity

Warmup Activity

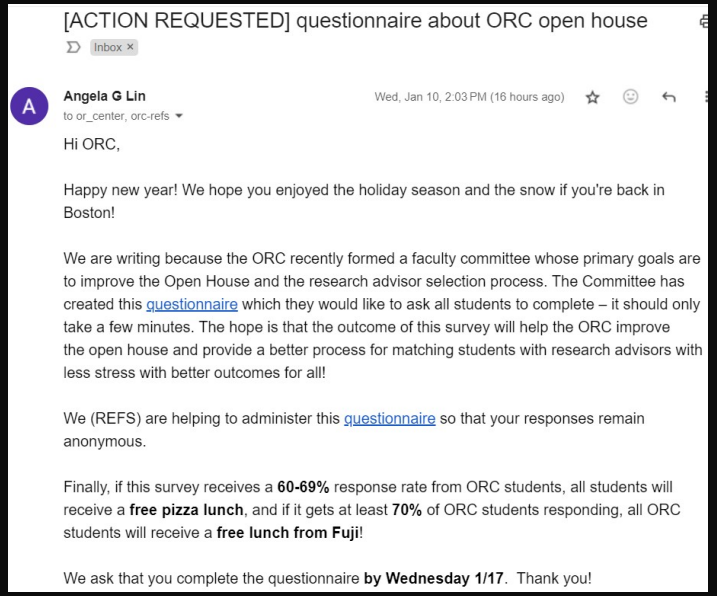
Who Likes Food?

Warmup Activity

Who Likes *Free* Food?

Warmup Activity

(3 minutes)



[ACTION REQUESTED] questionnaire about ORC open house

Inbox ×

Angela G Lin Wed, Jan 10, 2:03 PM (16 hours ago)

to or_center, orc-refs

Hi ORC,

Happy new year! We hope you enjoyed the holiday season and the snow if you're back in Boston!

We are writing because the ORC recently formed a faculty committee whose primary goals are to improve the Open House and the research advisor selection process. The Committee has created this questionnaire which they would like to ask all students to complete – it should only take a few minutes. The hope is that the outcome of this survey will help the ORC improve the open house and provide a better process for matching students with research advisors with less stress with better outcomes for all!

We (REFS) are helping to administer this questionnaire so that your responses remain anonymous.

Finally, if this survey receives a **60-69%** response rate from ORC students, all students will receive a **free pizza lunch**, and if it gets at least **70%** of ORC students responding, all ORC students will receive a **free lunch from Fuji**!

We ask that you complete the questionnaire **by Wednesday 1/17**. Thank you!

Warmup Activity

(3 minutes)

Finally, if this survey receives a **60-69%** response rate from ORC students, all st
receive a **free pizza lunch**, and if it gets at least **70%** of ORC students respondi
students will receive a **free lunch from Fuji!**

We ask that you complete the questionnaire **by Wednesday 1/17**. Thank you!

Plan

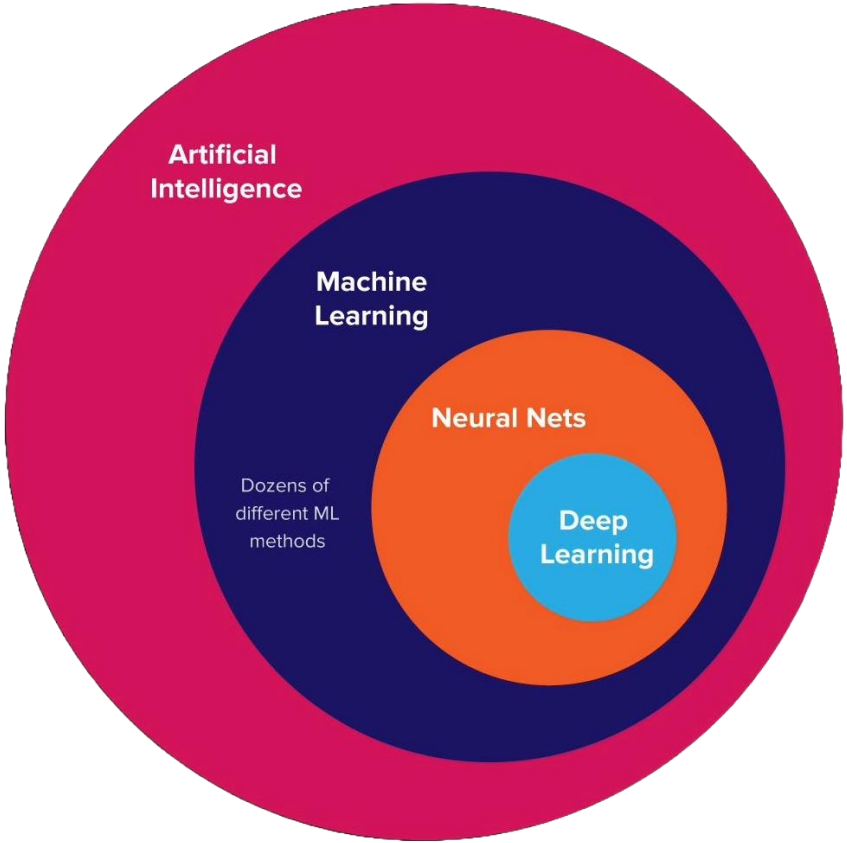
Concepts

- Perceptrons
- Gradient Descent & Training
- Convolutional Neural Networks
- Computer Vision

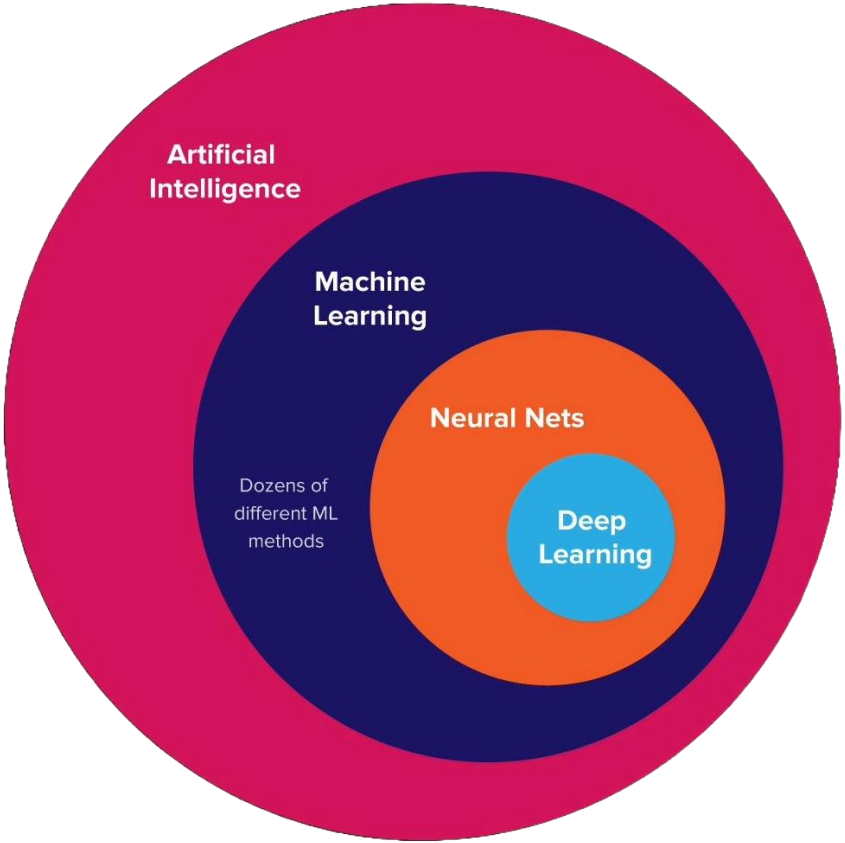
Code

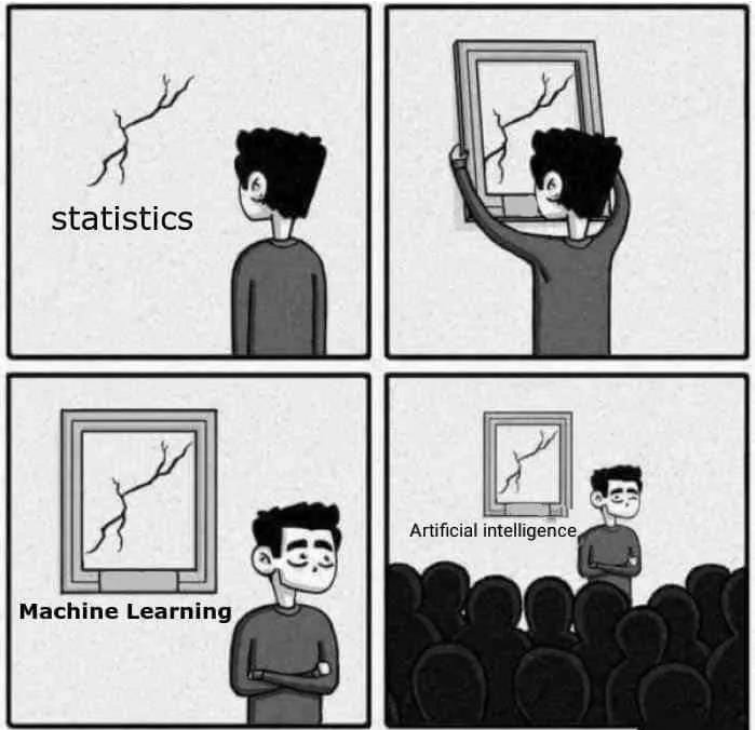
- TensorFlow

What is Deep Learning?



What is Deep Learning?





Pics:
https://serokell.io/blog/ai-ml-dl-difference
https://www.instagram.com/sandserifcomics/

What is the history behind the current AI boom?

1952 Stochastic Gradient Descent

1958 Perceptron

Learnable weights

1986 Backpropagation

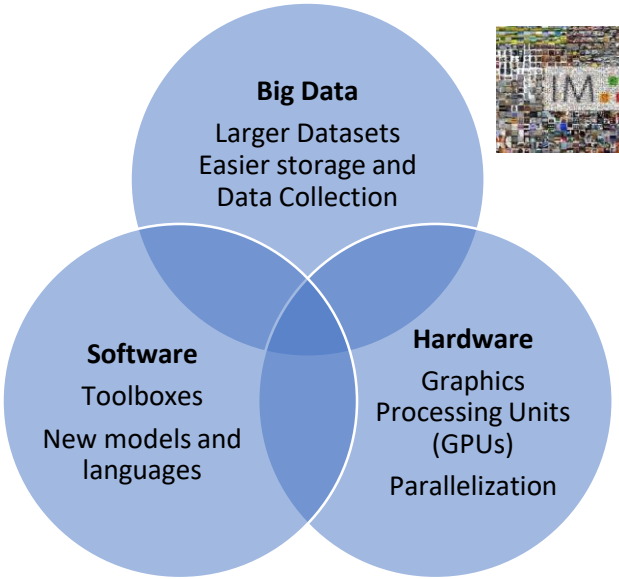
Multi-Layer Perceptron

1995 Deep Convolutional NN

Digit Recognition

2012 ImageNet competition

2019 Turing Award for Deep Learning



TensorFlow

PyTorch

What is the history behind the current AI boom?

1952 Stochastic Gradient Descent

1958 Perceptron

1986 Backpropagation

1995 Deep Convolutional NN

2012 ImageNet competition

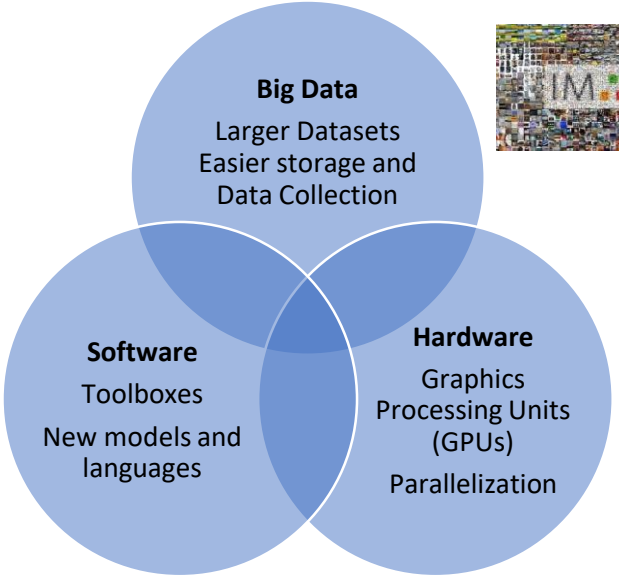
2019 Turing Award for Deep Learning

2023 ChatGPT, "Sparks of AGI"

Learnable weights

Multi-Layer Perceptron

Digit Recognition



TensorFlow

PyTorch

What is the history behind the current AI boom?

1952 Stochastic Gradient Descent

1958 Perceptron

1986 Backpropagation

1995 Deep Convolutional NN

2012 ImageNet competition

2019 Turing Award for Deep Learning

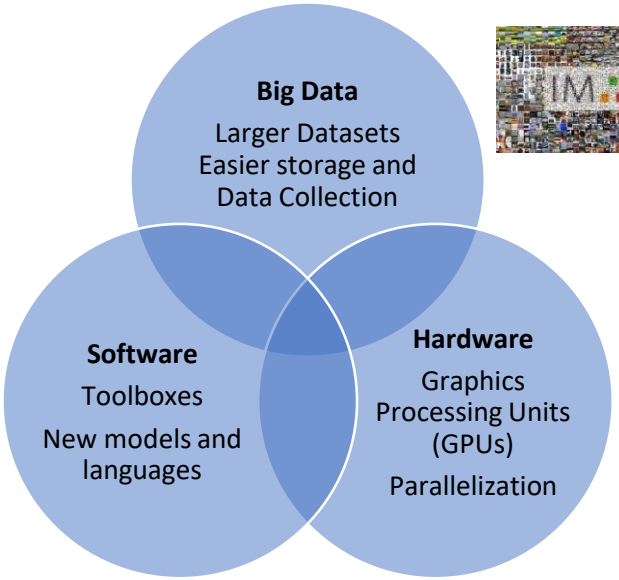
2023 ChatGPT, "Sparks of AGI"

2030+ Takover by our AI overlords?

Learnable weights

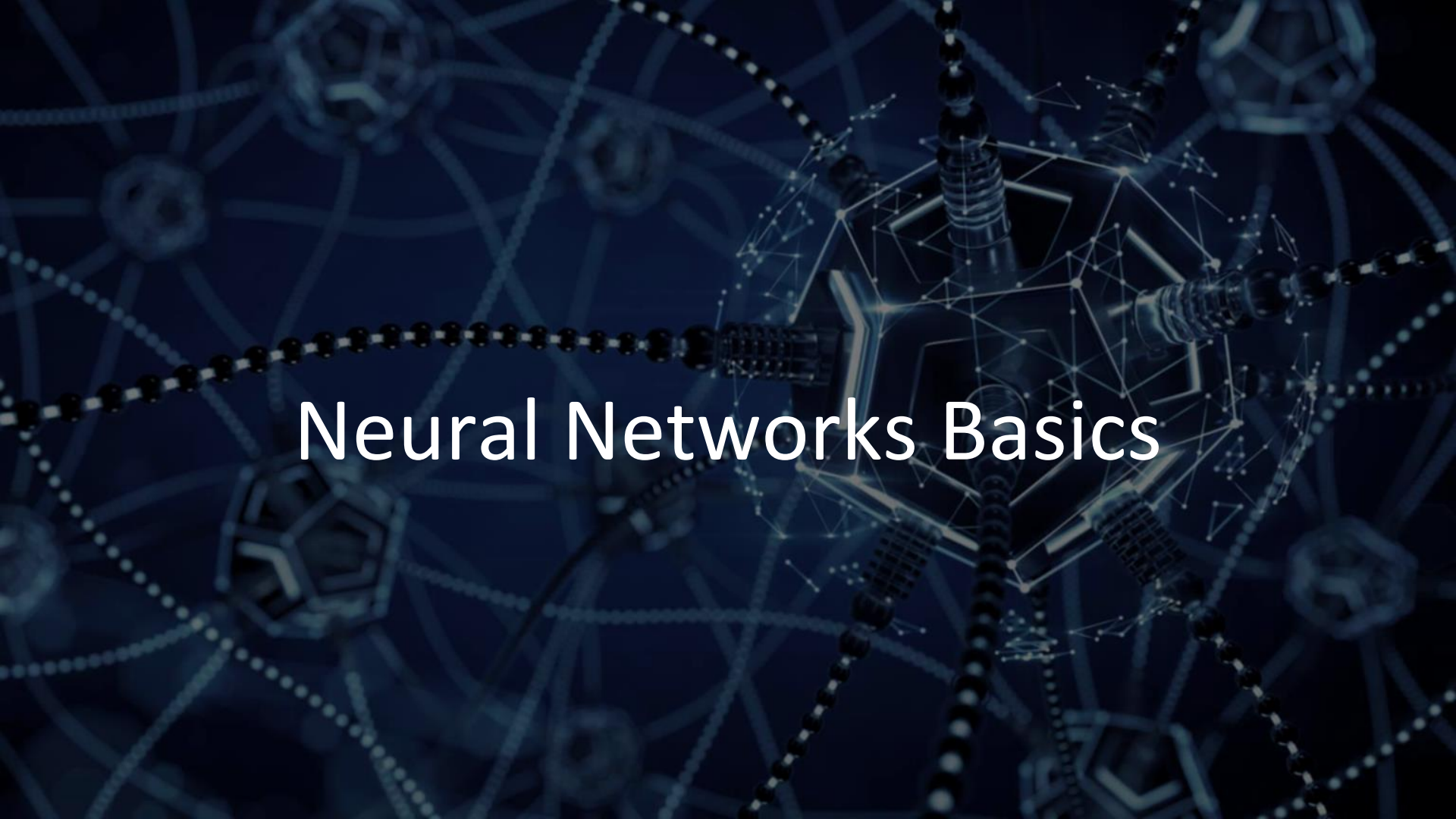
Multi-Layer Perceptron

Digit Recognition

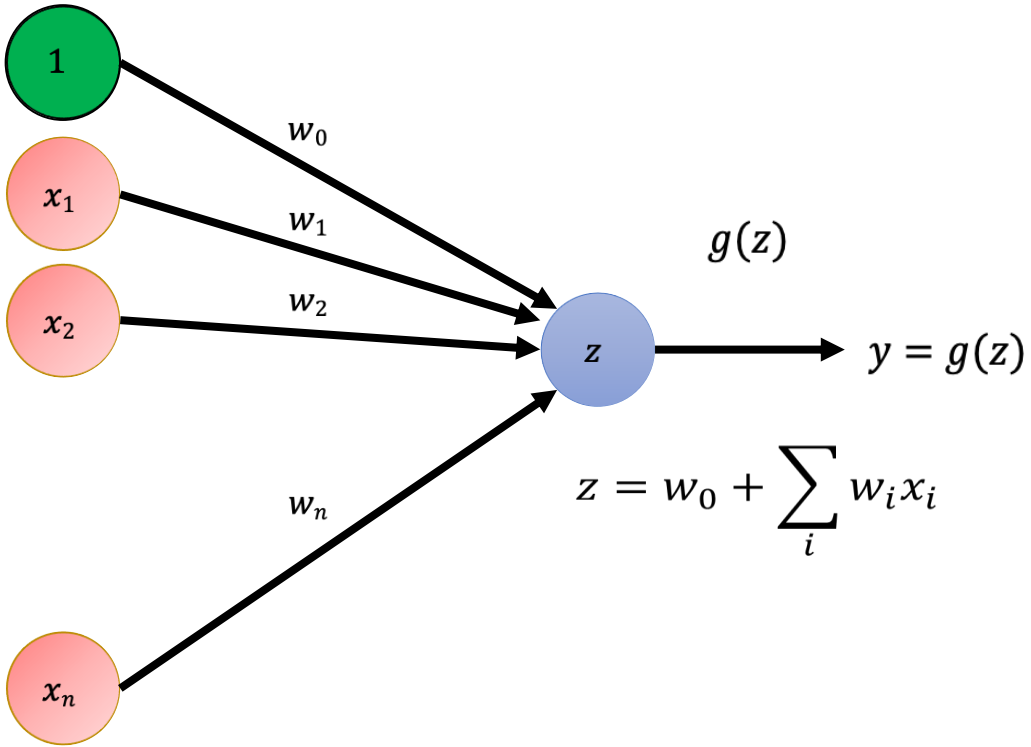


TensorFlow

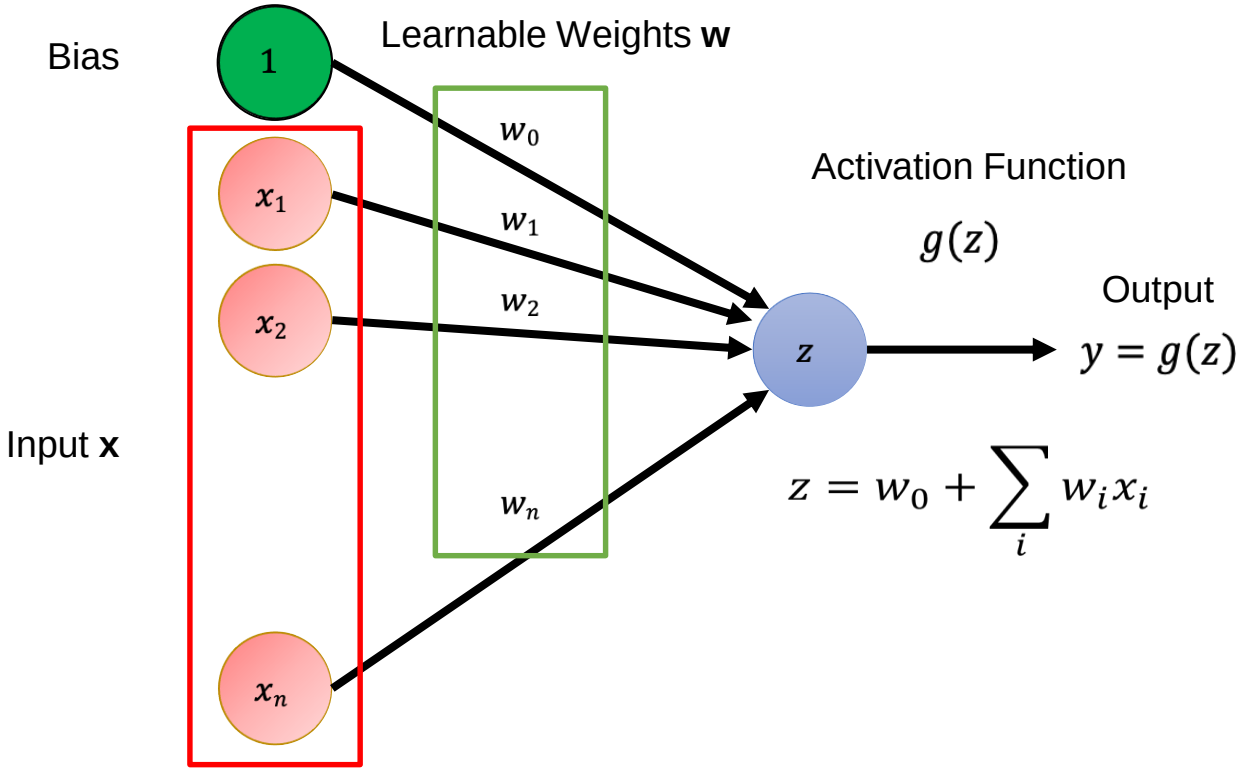
PyTorch


Neural Networks Basics

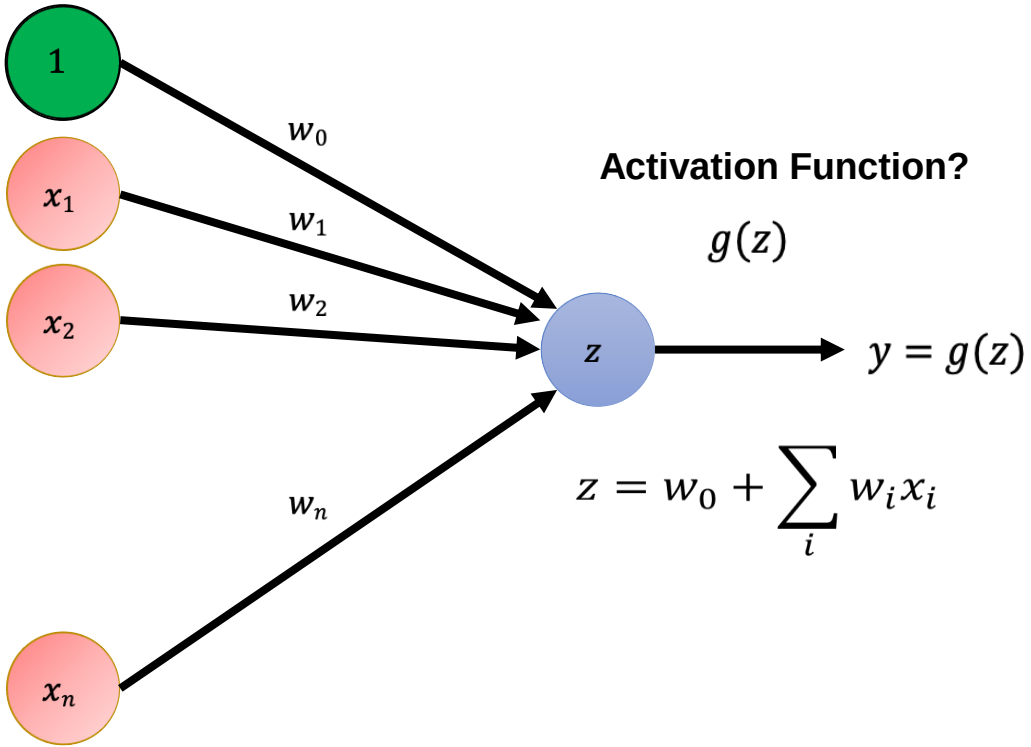
A perceptron



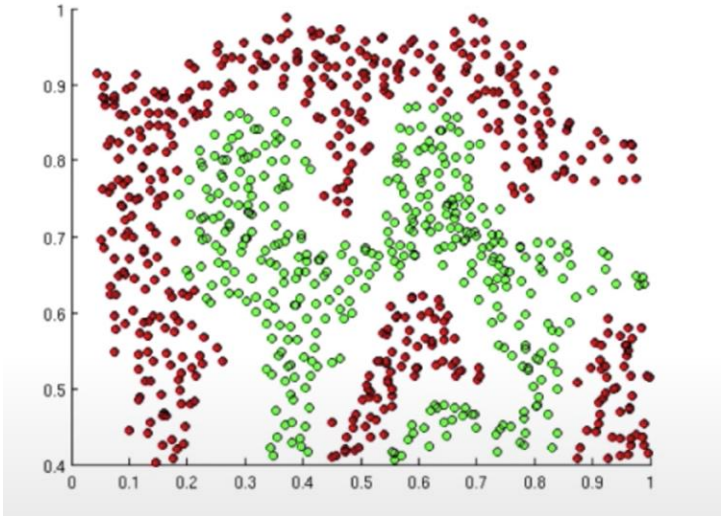
A perceptron



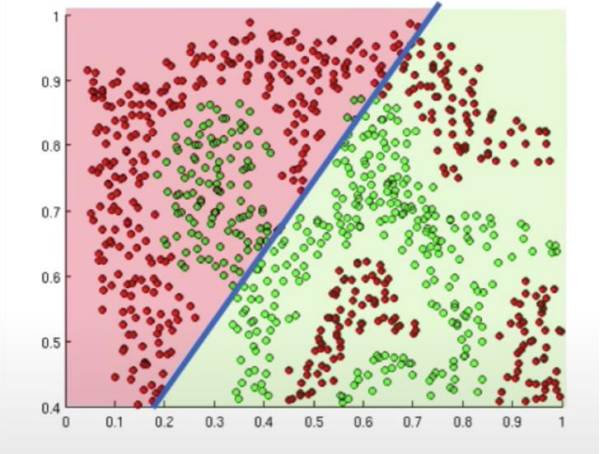
A perceptron



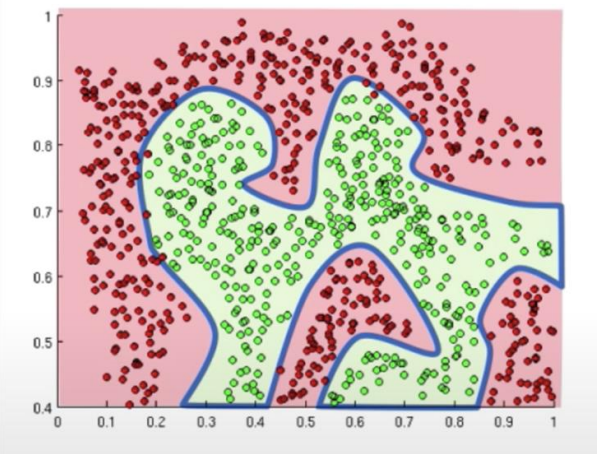
Classification Task



Activation functions

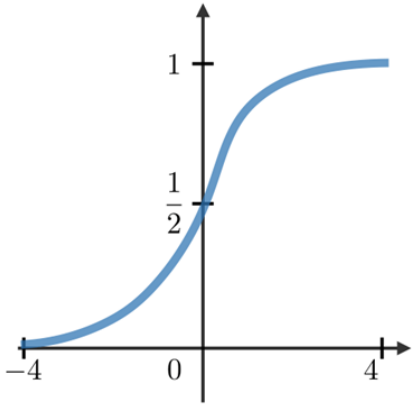
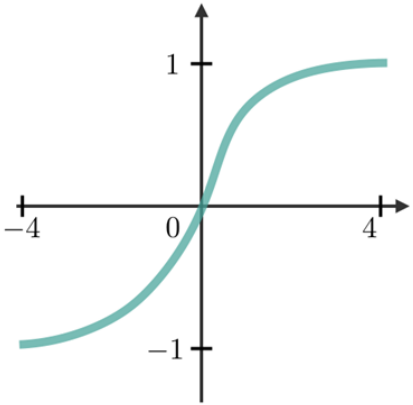
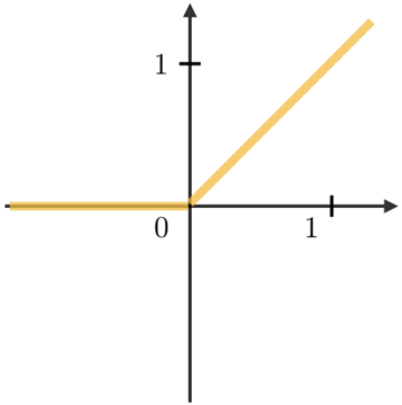


Linear activation functions -> linear decisions

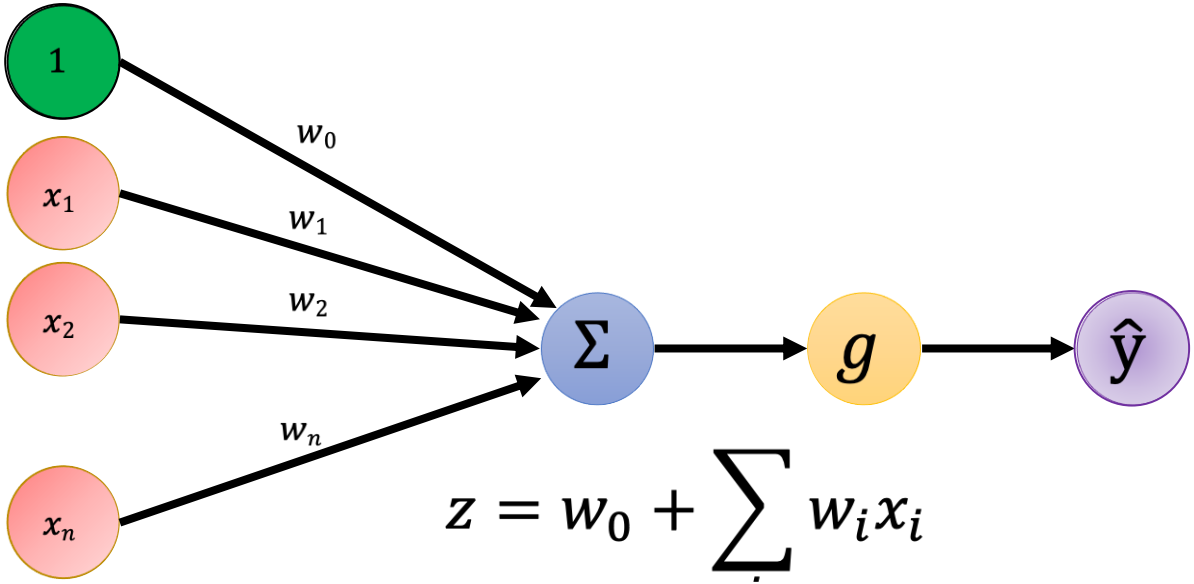


Non-linear activation -> approximate arbitrarily complex functions

Common Activation Functions

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$
		

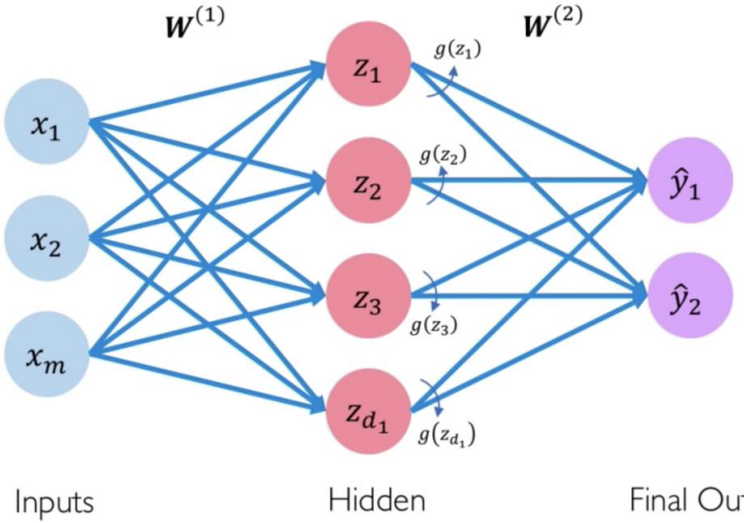
The perceptron: forward propagation



$$z = w_0 + \sum_i w_i x_i$$

$$\hat{y} = g(\sum_i w_i x_i)$$

Single Layer Neural Network



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) w_{j,i}^{(2)}\right)$$

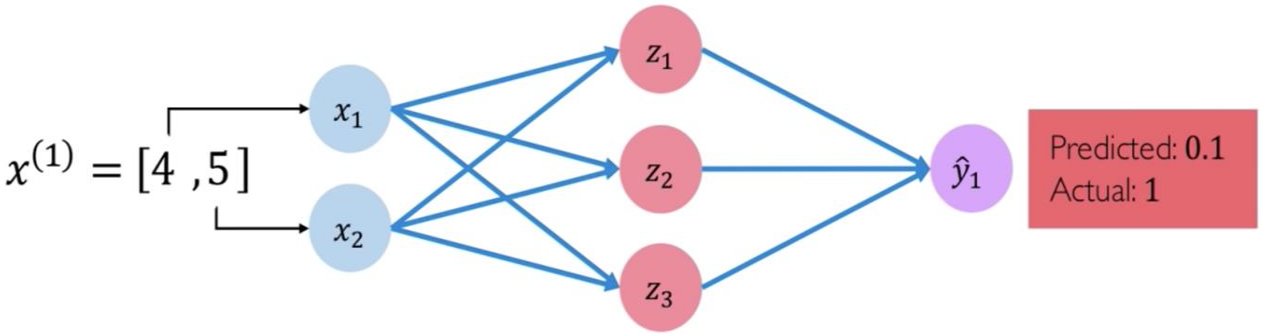
Picture source: http://introtodeeplearning.com/

Ok, but how do they (deep) learn?

Picture source: http://introtodeeplearning.com/

Loss function

The **loss** of our network measures the cost incurred from incorrect predictions.



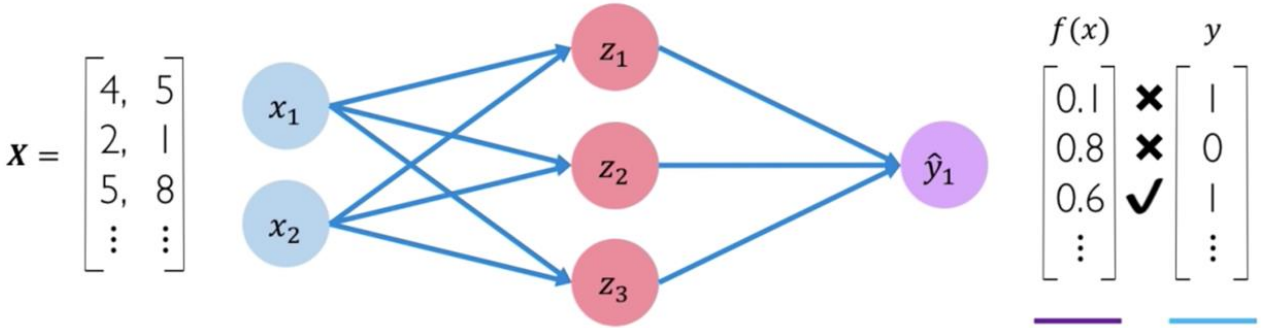
$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Picture source: http://introtodeeplearning.com/

Empirical Loss

The **empirical loss** measures the total loss over our entire dataset.

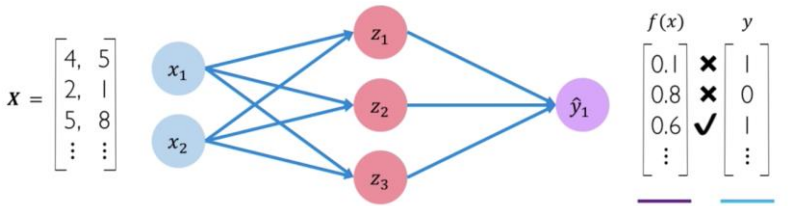


Picture source: http://introtodeeplearning.com/

Different loss functions

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

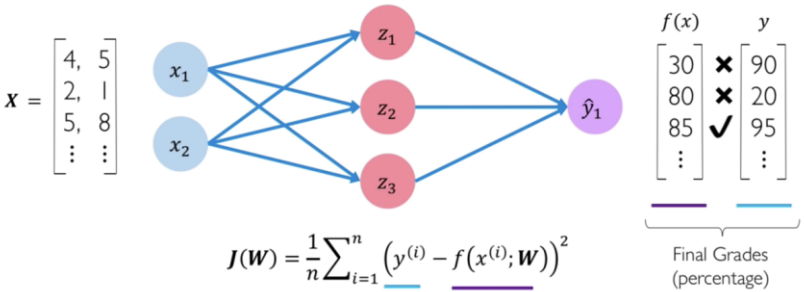


$$J(W) = -\frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log \underbrace{\left(f\left(x^{(i)}; W\right)\right)}_{\text{Predicted}} + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log \left(1 - \underbrace{f\left(x^{(i)}; W\right)}_{\text{Predicted}}\right)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*



$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\left(y^{(i)} - f\left(x^{(i)}; W\right)\right)^2}_{\text{Actual}} \quad \underbrace{\phantom{f\left(x^{(i)}; W\right)}}_{\text{Predicted}}$$

```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
```

Picture source: http://introtodeeplearning.com/

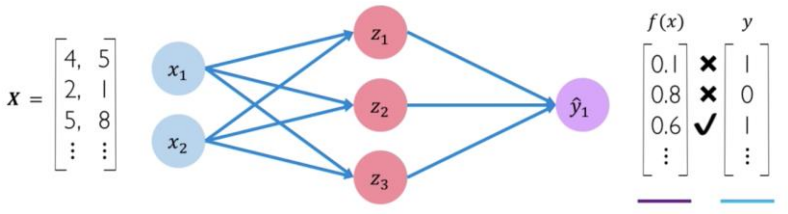
Different loss functions

Classification Tasks

Regression Tasks

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

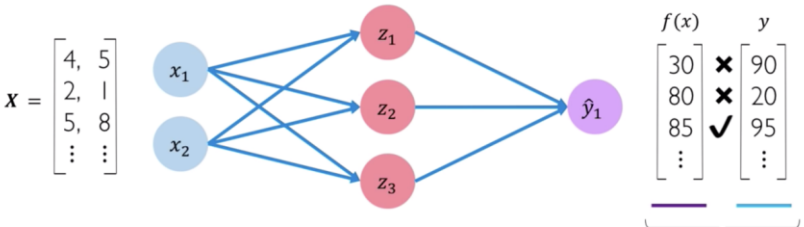


$$J(W) = -\frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log \underbrace{\left(f\left(x^{(i)}; W\right)\right)}_{\text{Predicted}} + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log \left(1 - \underbrace{f\left(x^{(i)}; W\right)}_{\text{Predicted}}\right)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*



$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \left(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\left(x^{(i)}; W\right)}_{\text{Predicted}}\right)^2$$

```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
```

Picture source: http://introtodeeplearning.com/

How to optimize the loss function?

We need to find the network weights leading to the lowest loss.

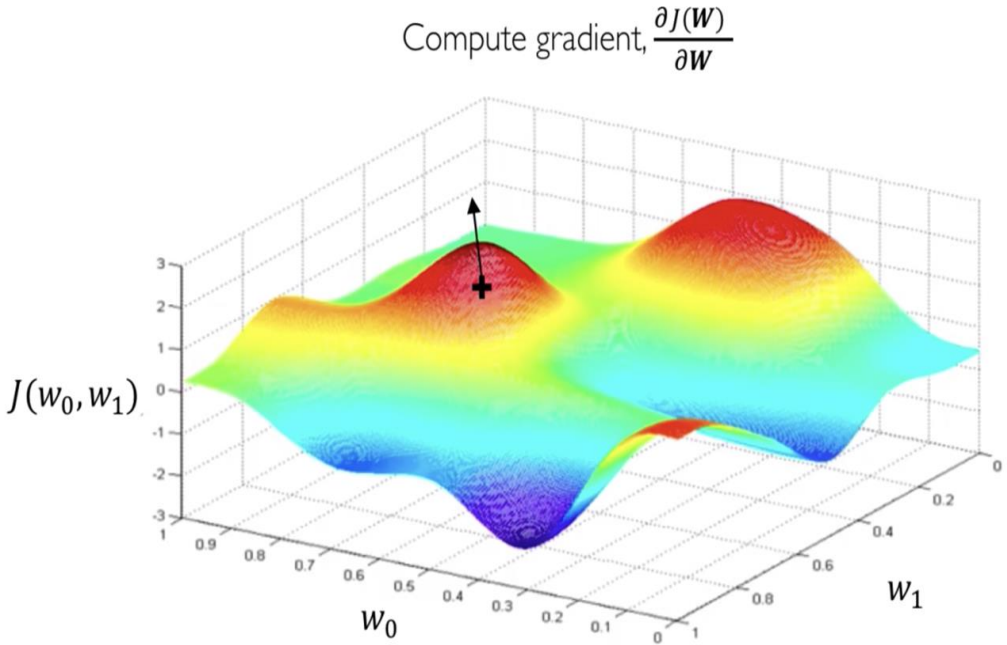
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

Remember:

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

But it looks like this…



Picture source: http://introtodeeplearning.com/

Fortunately, there is Gradient Descent!

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Fortunately, there is Gradient Descent!

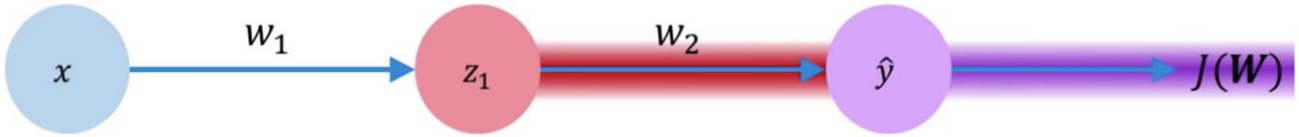
Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

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

How do we calculate this?

Backpropagation with chain rule!



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

Picture source: http://introtodeeplearning.com/

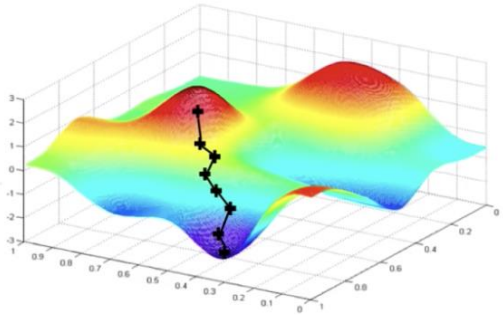
Computing gradients is expensive!

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Can be very **computationally intensive** to compute!



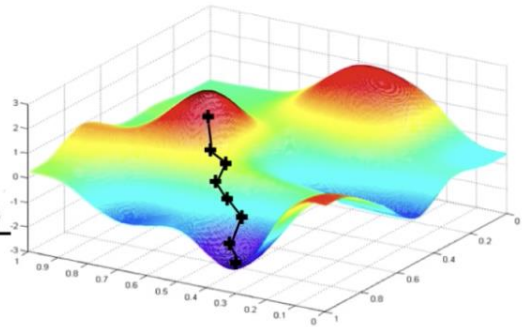
Picture source: http://introtodeeplearning.com/

Computing gradients is expensive!

Stochastic Gradient Descent

Algorithm

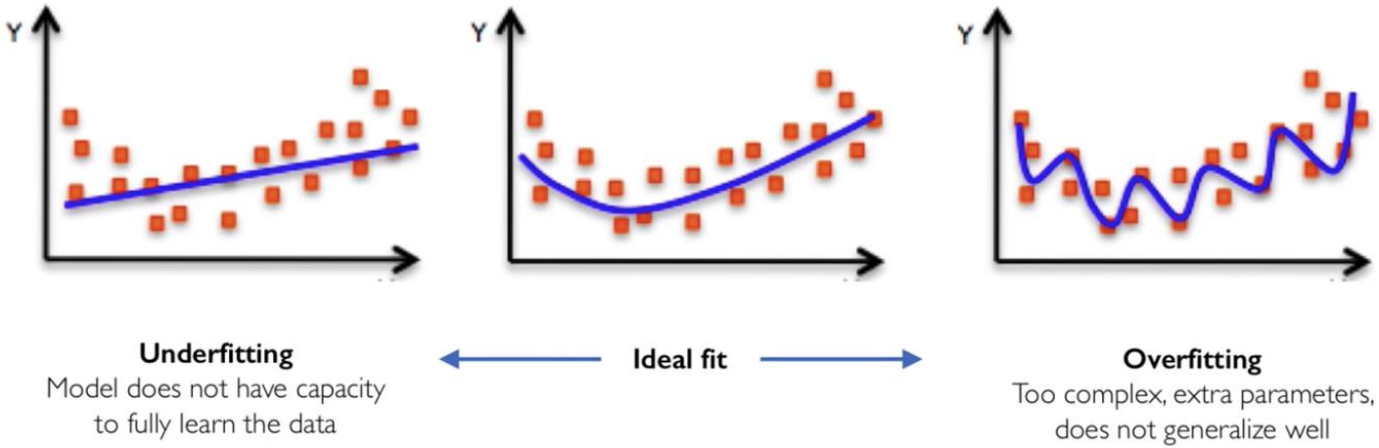
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



We use **mini-batches** while training allowing for a smoother convergence and larger learning rates.

Picture source: http://introtodeeplearning.com/

Another classic ML problem: overfitting



Picture source: http://introtodeeplearning.com/

Let’s overcome this! Ideas?

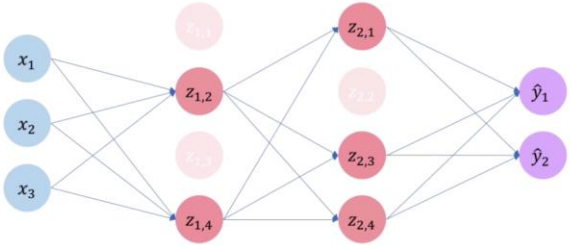
Picture source: http://introtodeeplearning.com/

Let’s overcome this! Ideas?

Regularization 1: Dropout

- During training, randomly set some activations to 0
  - Typically ‘drop’ 50% of activations in layer
  - Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```



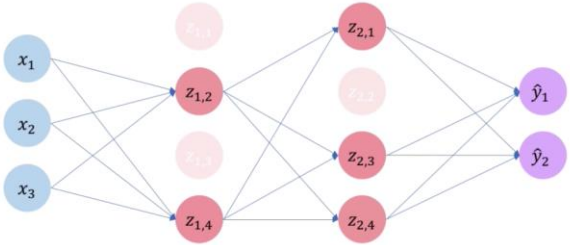
Picture source: http://introtodeeplearning.com/

Let’s overcome this! Ideas?

Regularization 1: Dropout

- During training, randomly set some activations to 0
  - Typically ‘drop’ 50% of activations in layer
  - Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```



Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

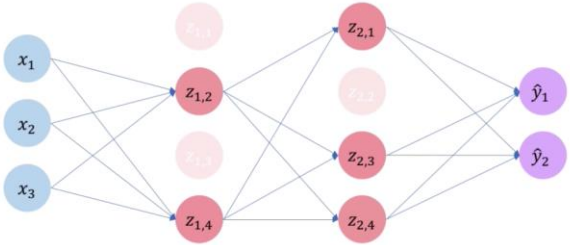
Picture source: http://introtodeeplearning.com/

Let’s overcome this! Ideas?

Regularization 1: Dropout

- During training, randomly set some activations to 0
  - Typically ‘drop’ 50% of activations in layer
  - Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```



Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

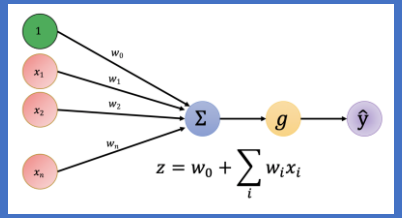
Regularization 3: L1 or L2 penalty on the weights

Picture source: http://introtodeeplearning.com/

Review

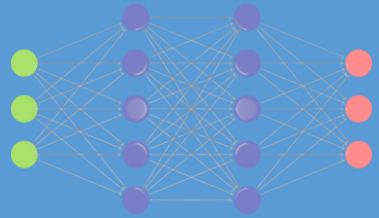
The Perceptron

- Structure
- Nonlinear activation functions



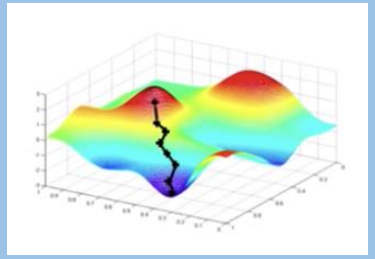
Neural Networks

- Stacking layers of perceptrons
- Backpropagation and gradient descent



Training in Practice (in theory)

- Adaptive learning
- Batching
- Regularization

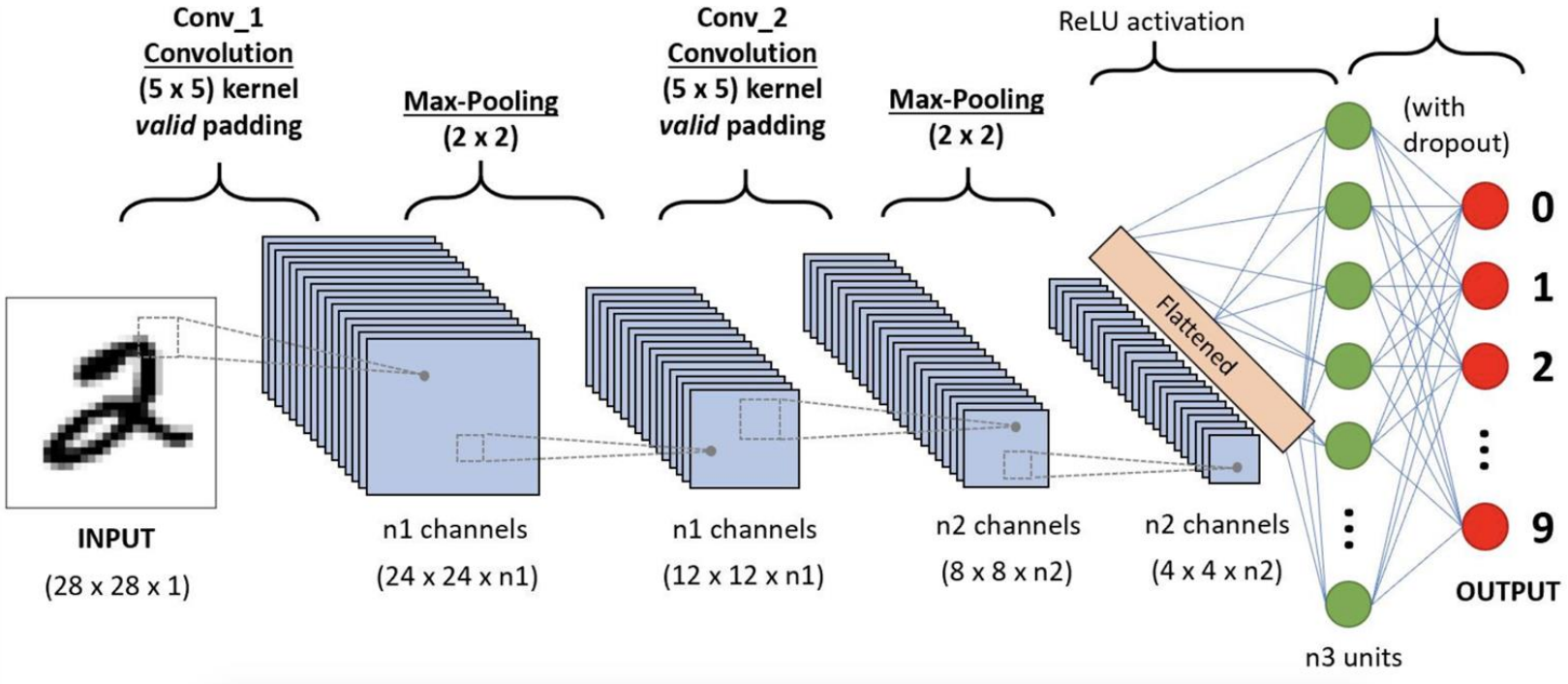


Please ask any questions!

Break

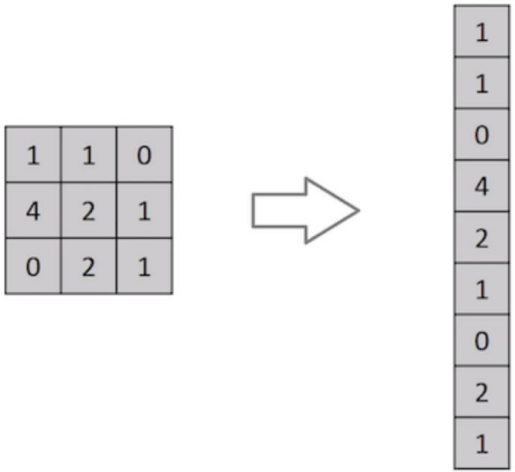
Convolutional Neural Networks

Let’s now deal with images!

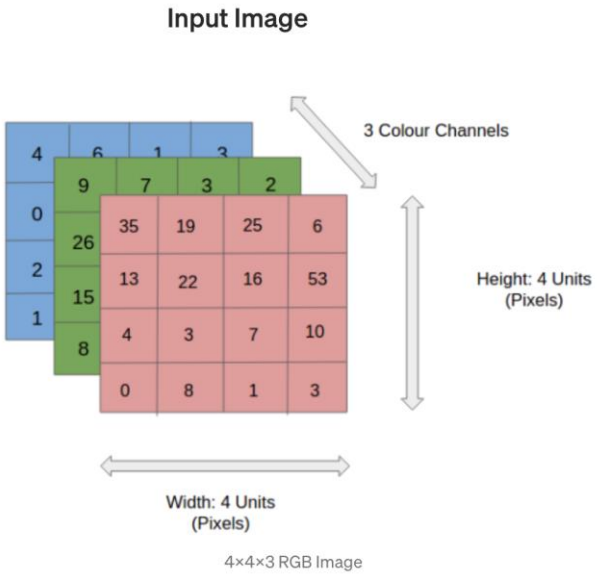


Images source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

3-dimensional data

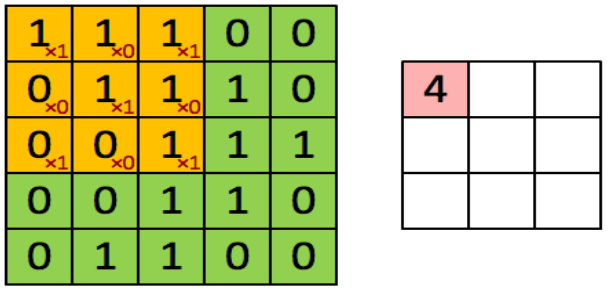


Flattening of a 3×3 image matrix into a 9×1 vector



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

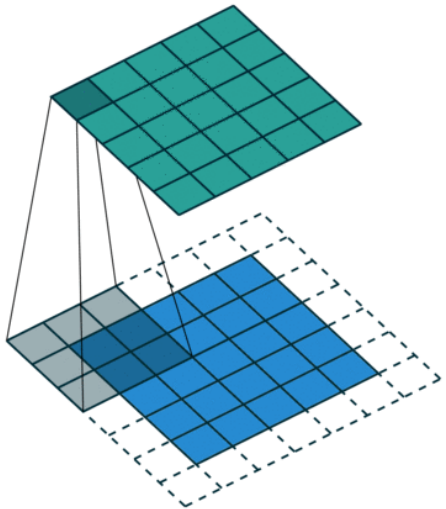
Convolution Operation



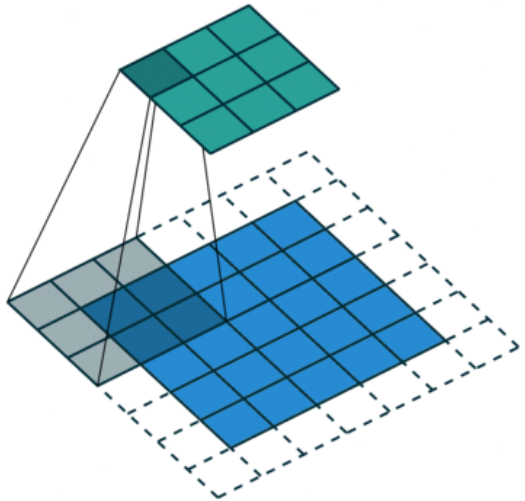
Images source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolution Operation

Padding: 1
Stride: 1



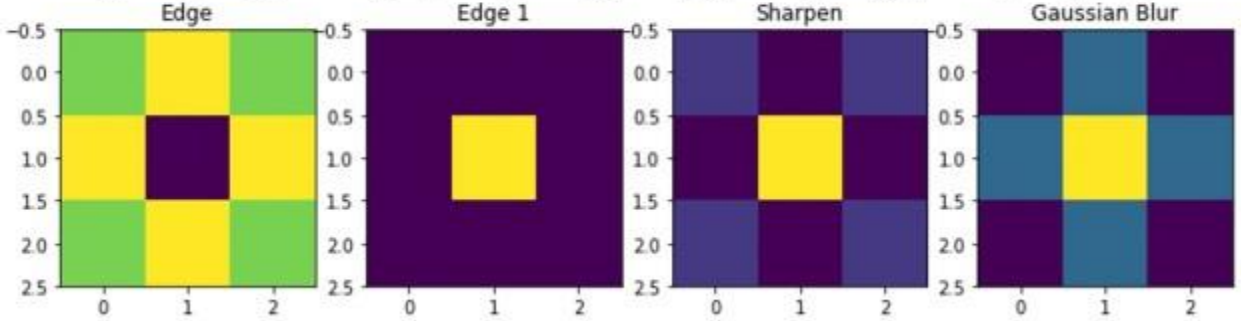
Padding: 1
Stride: 2



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

Different kernels for different feature extractions

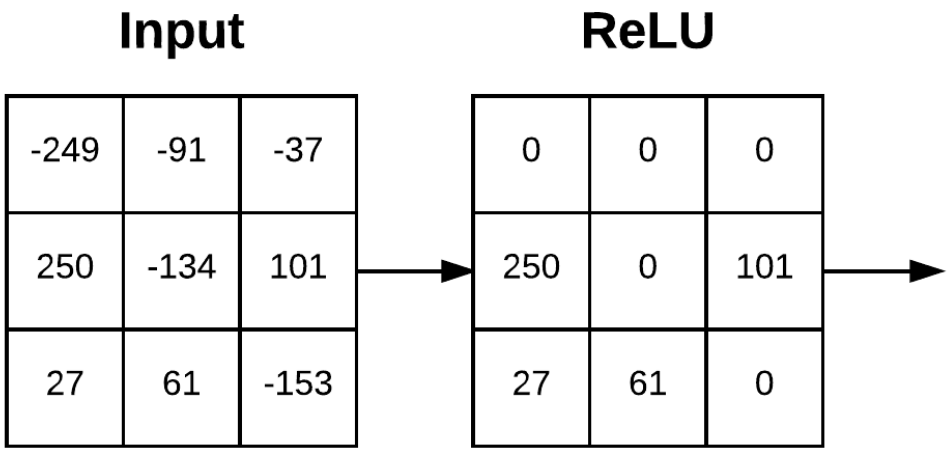
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \times \frac{1}{16}$$



Images source: https://ucanalytics.com/blogs/convolutional-neural-networks-cnn-simplified-part-4/

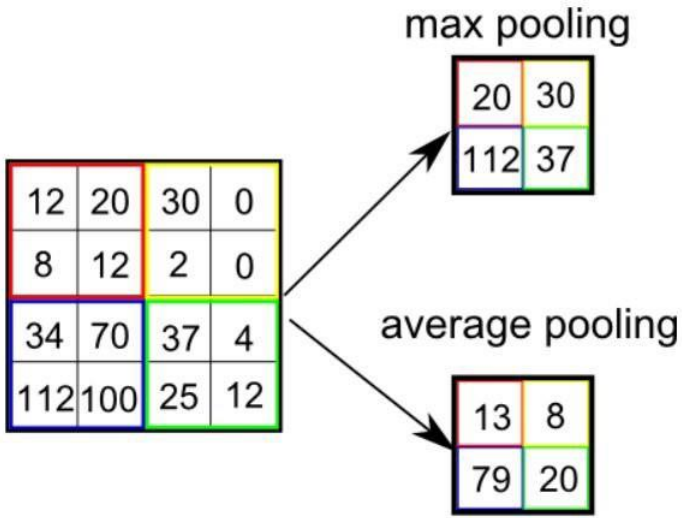
Activation layer

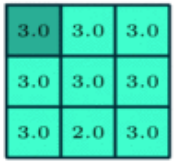
After each convolutional layer, we apply an activation function (usually ReLU)

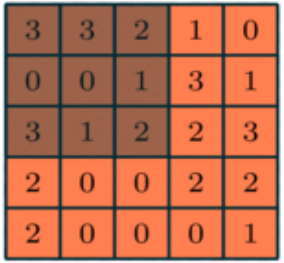


Images source: https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/

Pooling



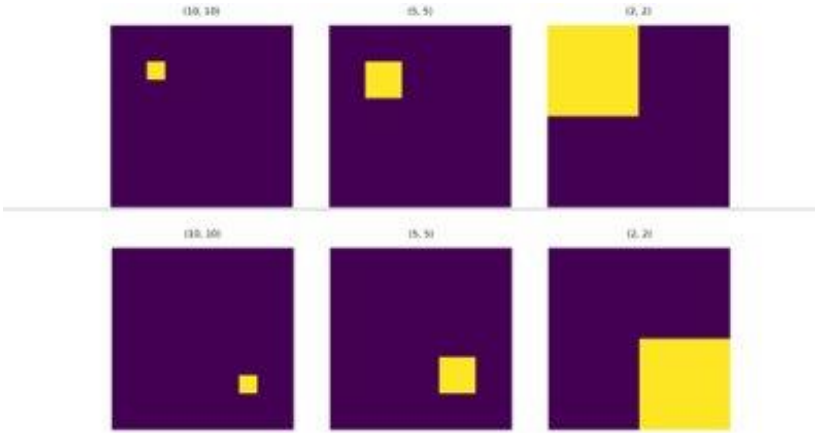


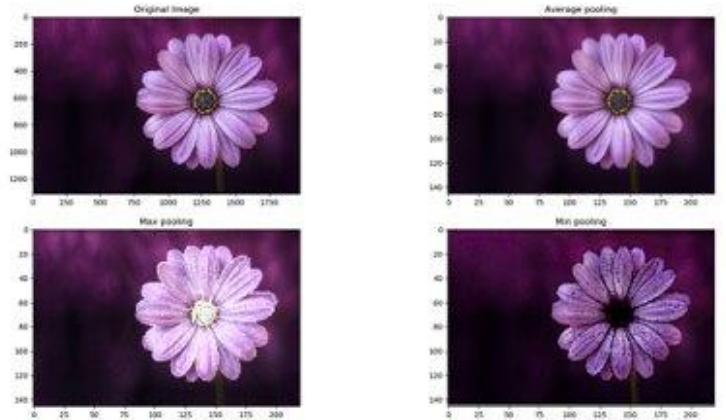


Images source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Pooling (comparison)

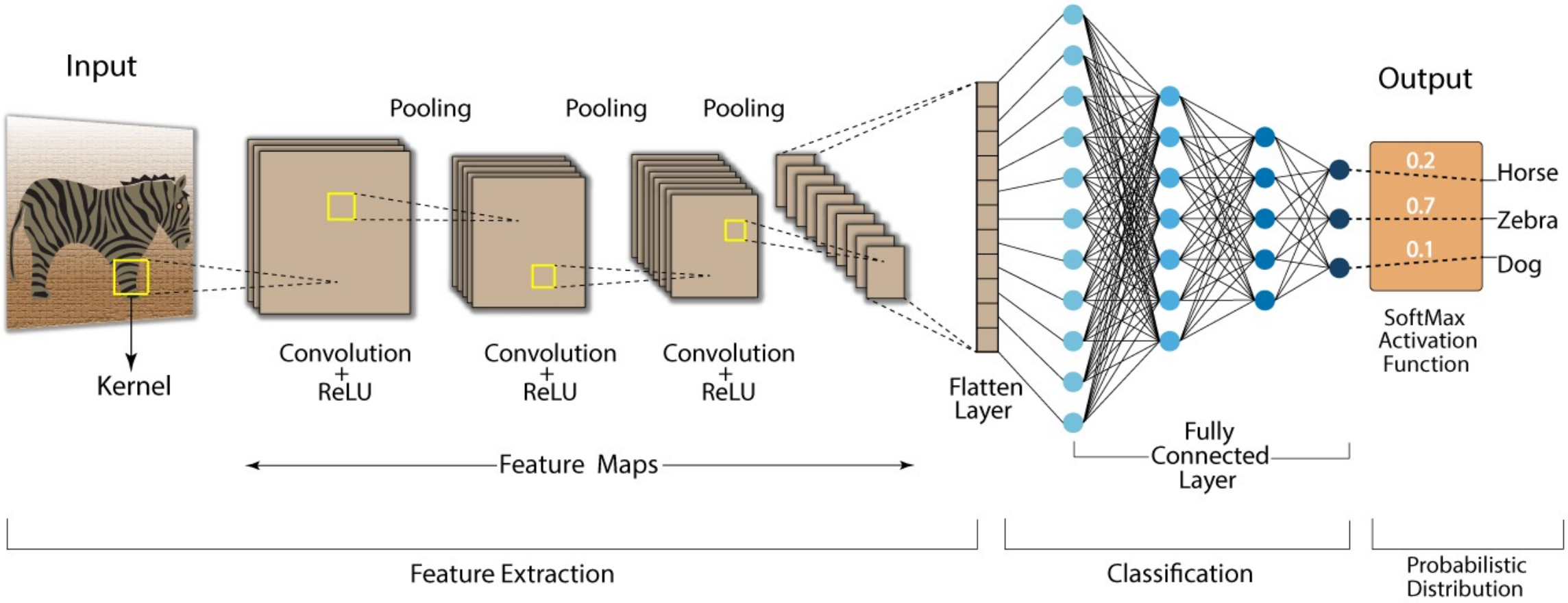
Max pooling





Images source:
https://blog.paperspace.com/pooling-and-translation-invariance-in-convolutional-neural-networks/
https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9

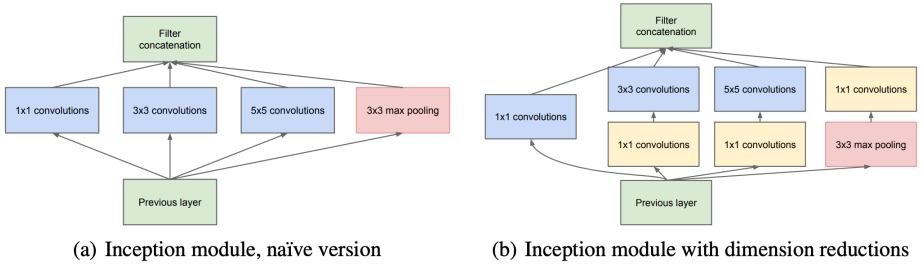
Overall Architecture of a CNN



Images source: https://www.analyticsvidhya.com/blog/2021/05/20-questions-to-test-your-skills-on-cnn-convolutional-neural-networks/

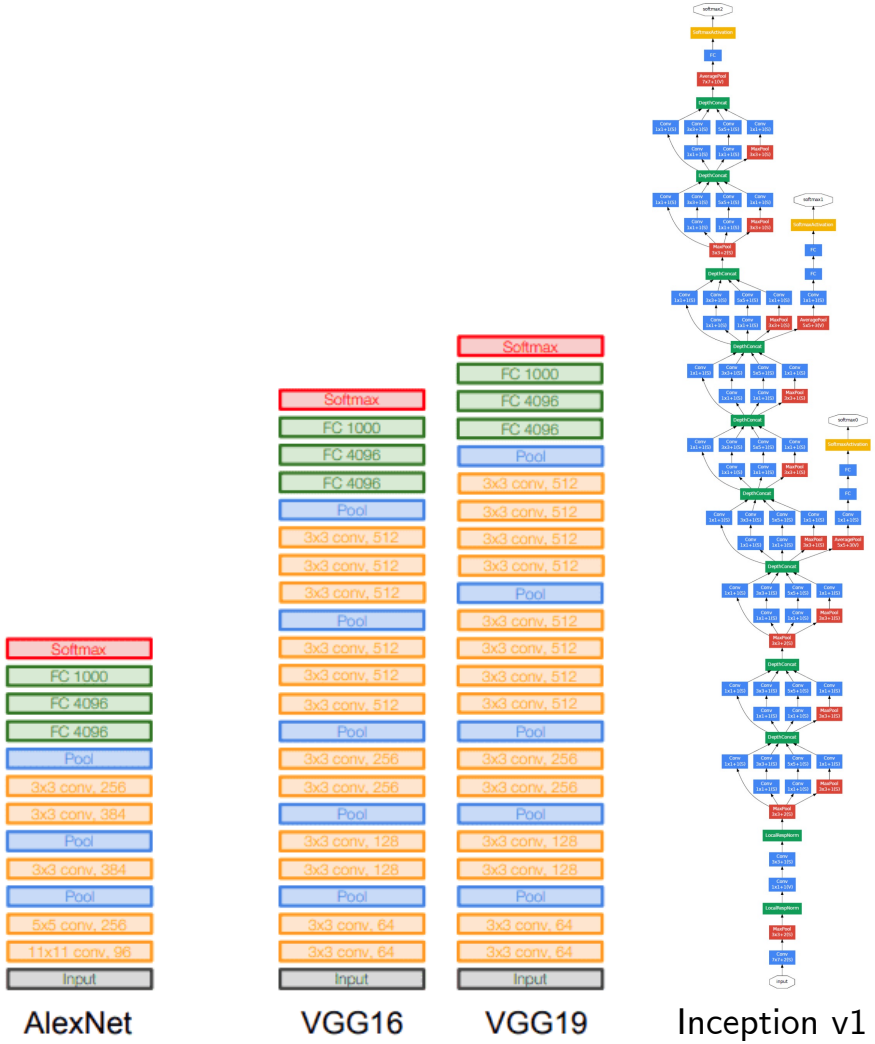
Historic CNN architectures

- Historic architectures:
- LeNet-5 (1989),
- AlexNet (2012),
- GoogleNet/Inception (2014),
- VGG (2014).

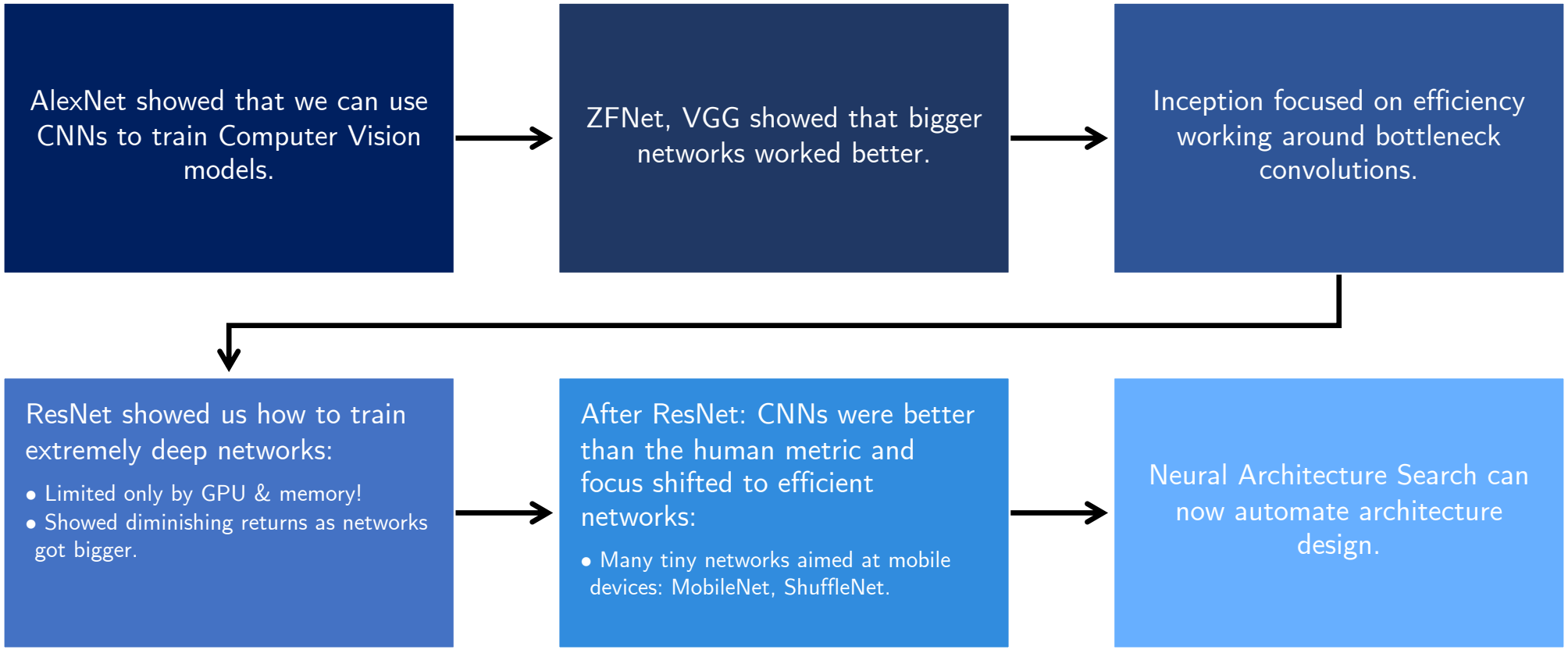


(a) Inception module, naïve version (b) Inception module with dimension reductions

Inception module from GoogleNet



CNN History: What to remember?

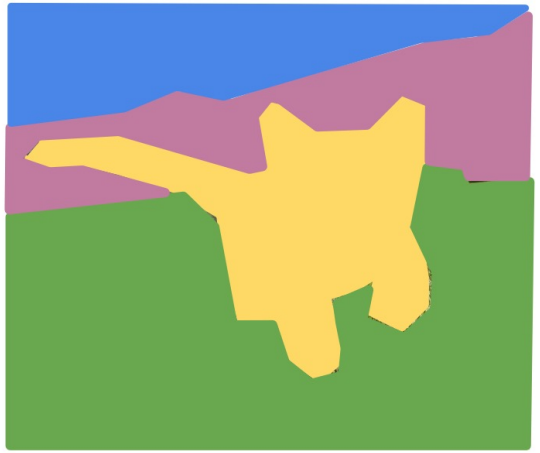


What kind of other tasks can we perform?

- Classification, Regression
- Segmentation
- Prediction of next image
- Content generation
- Feature extraction
- Descriptions
- ...

Other Computer Vision Tasks

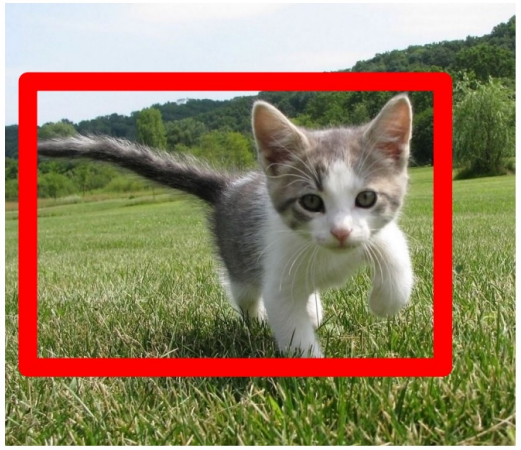
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

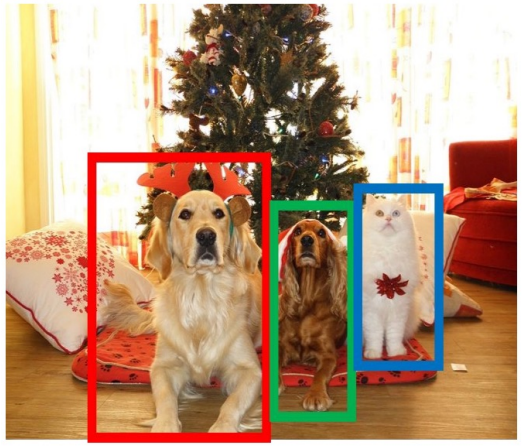
Classification + Localization



CAT

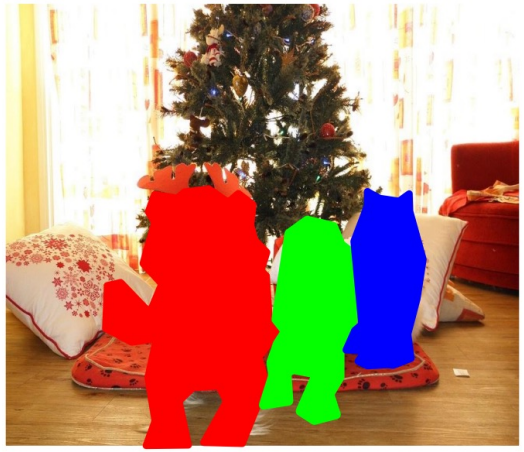
Single Object

Object Detection



DOG, DOG, CAT

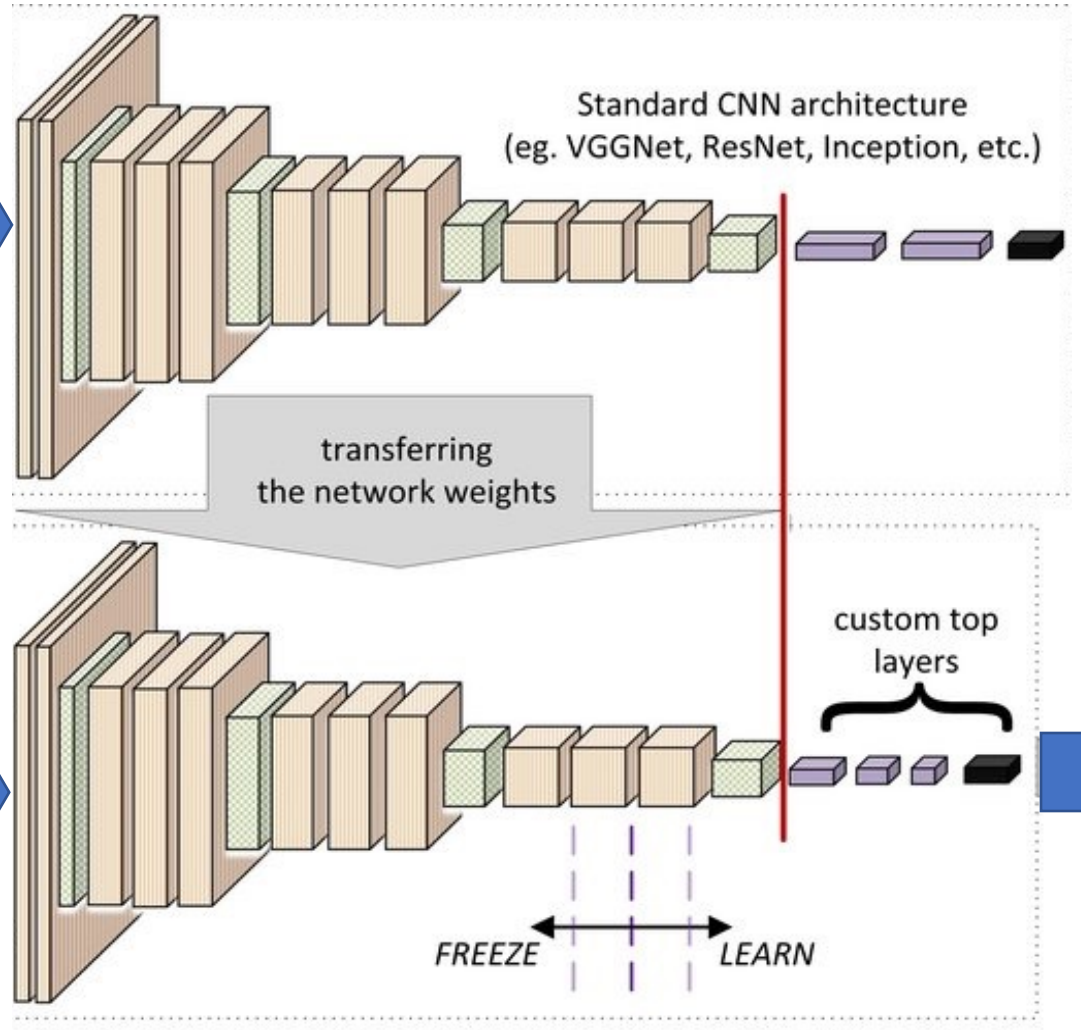
Instance Segmentation



DOG, DOG, CAT

Multiple Object

Transfer Learning



Prediction

Please ask any questions!

Thank you for your attention!