IVA Training

Presented by: Ahmed Elfakharany

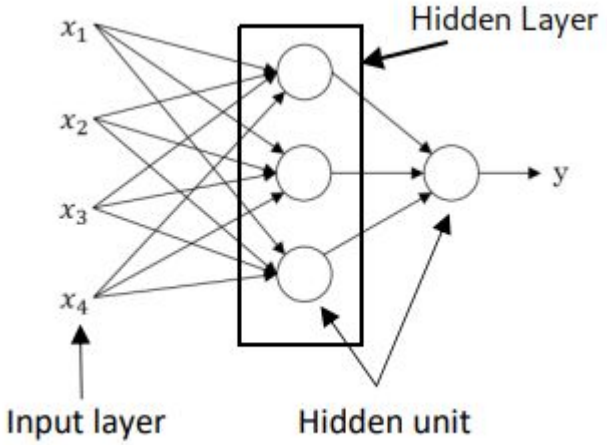
Day 1

Agenda

1. What is a Neural Network (NN) ?
2. Supervised Learning With NN
3. Logistic Regression
4. Gradient Descent
5. NN representation
6. Computing NN output
7. Activation Functions
8. Gradient Descent for NN
9. Random Initialisation
10. Deep NN

What is a Neural Network (NN) ?

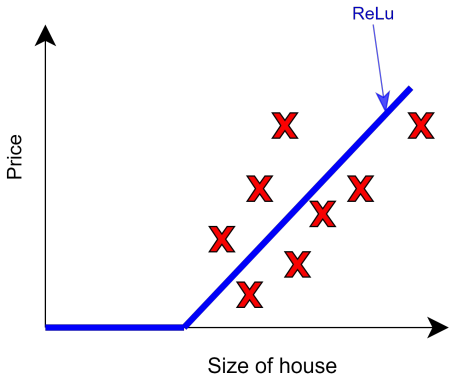
- It is a powerful machine learning algorithm inspired by how the brain works.
- It is composed of **layers**, each layers are composed of **neurons (units).**
- It mainly has an input layer, hidden layers and output layers.



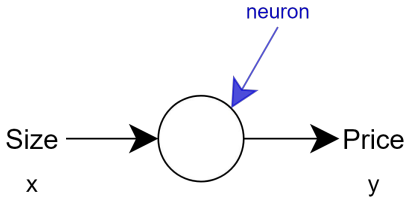
What is a Neural Network (NN) ?

Example: Single neural network

- We need to predict the price of a house given its size.
- The dataset we have has size of houses and their prices.
- It is a linear regression problem because the price as a function of size is a continuous output.
- Prices can never be negative so the function that should be learnt is called Rectified Linear Unit (ReLU) which starts at zero.



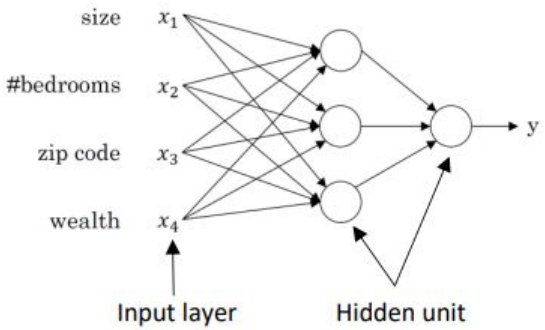
- The input is the size of the house (x)
- The output is the price (y)
- The “neuron” learns to implement the function ReLU (blue line)



What is a Neural Network (NN) ?

Example: Multiple neural network

- The price of a house can be affected by other features such as size, number of bedrooms, zip code and wealth.
- The role of the neural network is to predicted the price and it will automatically learn the functions in the hidden units.
- We only need to give the inputs x and the output y.

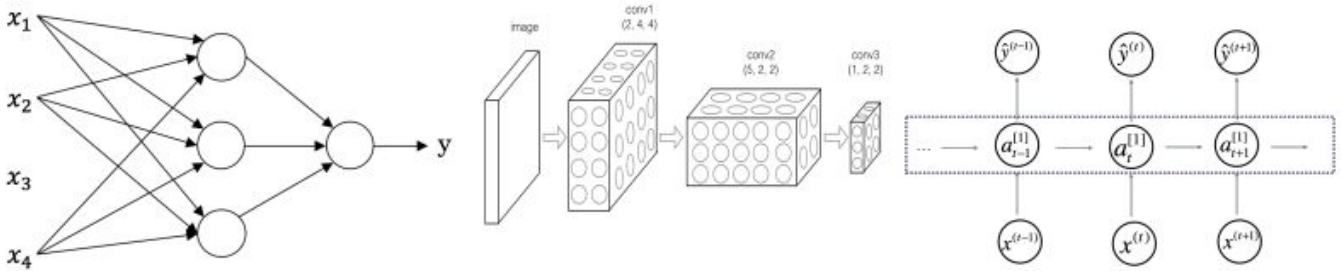


Supervised Learning With NN

- In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.
- Supervised learning problems are categorized into "regression" and "classification" problems.
    - In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function
    - In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

Supervised Learning With NN
Kinds of NN



Standard NN Convolutional NN Recurrent NN

Supervised Learning With NN
Structured vs unstructured data

Structured Data

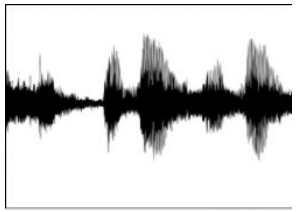
- Structured data refers to things that has a defined meaning such as price, age.

Size	#bedrooms	...	Price (1000$s)
2104	3		400
1600	3		330
2400	3		369
⋮	⋮		⋮
3000	4		540

User Age	Ad Id	...	Click
41	93242		1
80	93287		0
18	87312		1
⋮	⋮		⋮
27	71244		1

Unstructured Data

- Unstructured data refers to thing like pixel, raw audio, text.



Audio

Image

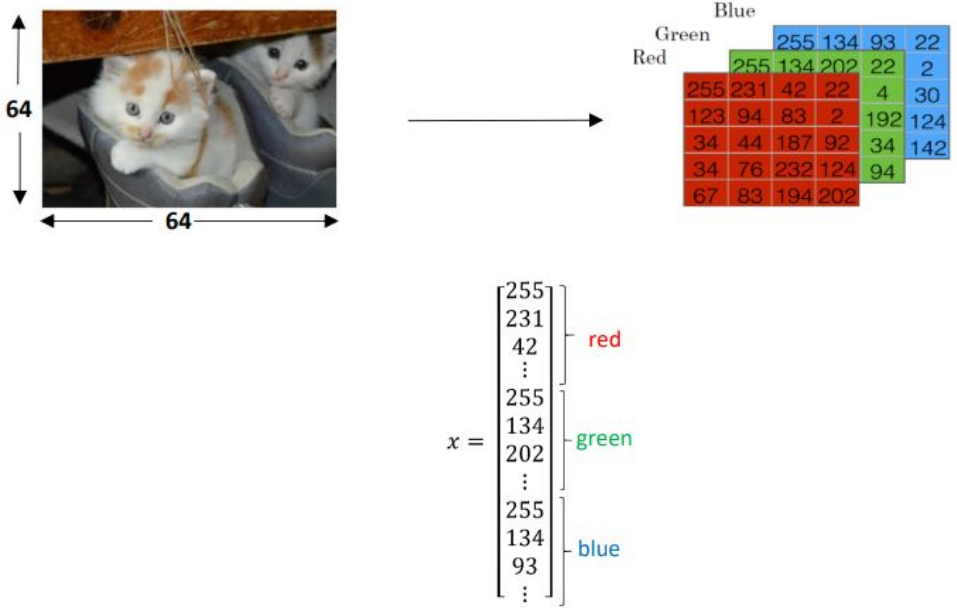
Four scores and seven years ago…

Text

Before diving into NN, let's talk about logistic regression to explain some concepts first

Logistic Regression
Binary Classification

- In a binary classification problem, the result is a discrete value output (0 or 1).
- Example: to classify a cat image, the image is converted to a feature vector.
- To create a feature vector, x, the pixel intensity values will be “unroll” or “reshape” for each color.
- The dimension of the input feature vector x is nx = 64 * 64 * 3 = 12288.



Logistic Regression

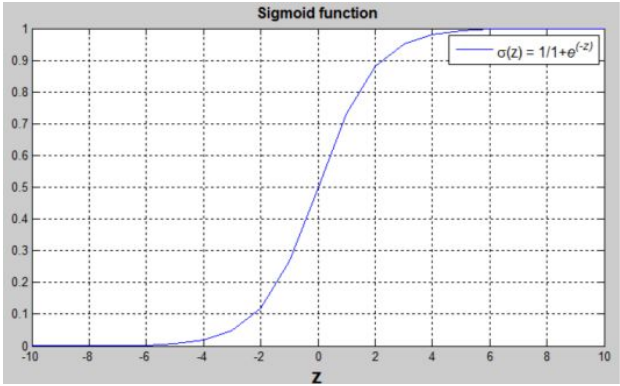
- Logistic regression is a learning algorithm used in a supervised learning problem when the output y are all either zero or one.
- The goal of logistic regression is to minimize the error between its predictions and training data.

$Given\ x$, $\hat{y} = P(y = 1 \mid x)$, where $0 \leq \hat{y} \leq 1$

Logistic Regression

The parameters used in Logistic regression are:

- The input features vector: $x \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The training label: $y \in 0,1$
- The weights: $w \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The threshold: $b \in \mathbb{R}$
- The output: $\hat{y} = \sigma(w^T x + b)$
- Sigmoid function: $s = \sigma(w^T x + b) = \sigma(z) = \frac{1}{1+e^{-z}}$



$(w^T x + b)$ is a linear function ($ax + b$), but since we are looking for a probability constraint between [0,1], the sigmoid function is used.

Logistic Regression
Cost Function

$$\hat{y} = \sigma\left(w^T x + b\right),\ \sigma(z) = \frac{1}{1+e^{-z}}$$

$$Given\ \left\{\left(x^{(1)}, y^{(1)}\right), \ldots\ldots, \left(x^{(m)}, y^{(m)}\right)\right\},\ want\ \hat{y}^{(i)} \approx y^{(i)}$$

$$Loss\ (error)\ funtion:\ L(\hat{y}, y) = -\left(y \log \hat{y} + (1-y) \log\left(1-\hat{y}\right)\right)$$

$$Cost\ function\ for\ the\ entrie\ dataset:$$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right)$$

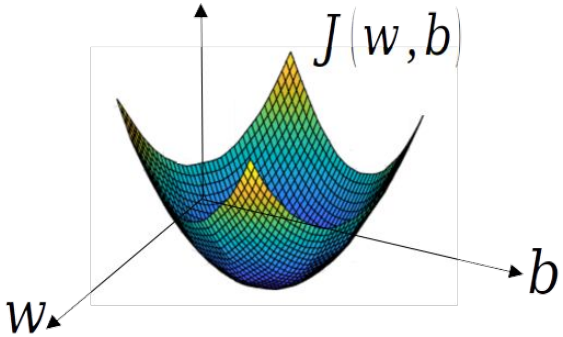
Gradient Descent

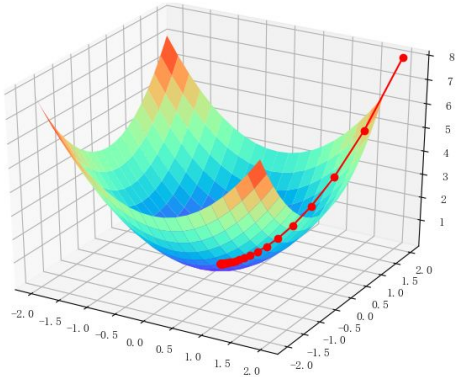
- Gradient descent is used to find *w* and *b* that minimizes the cost function *J*.
- Gradient descent keeps repeating the following equation till the error is under an accepted threshold

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

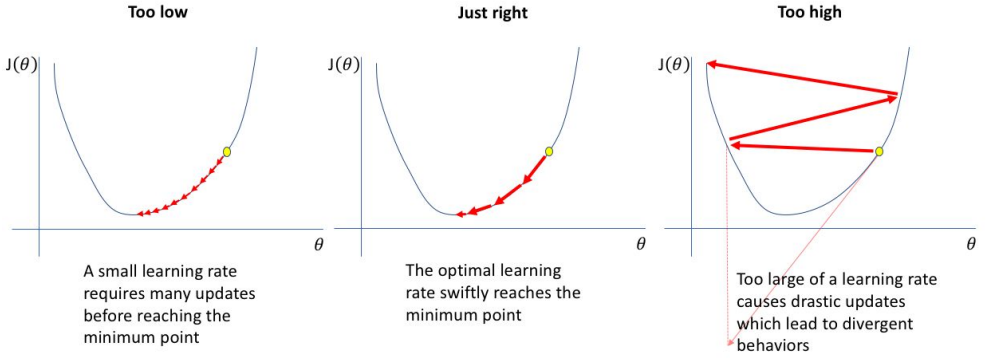
$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

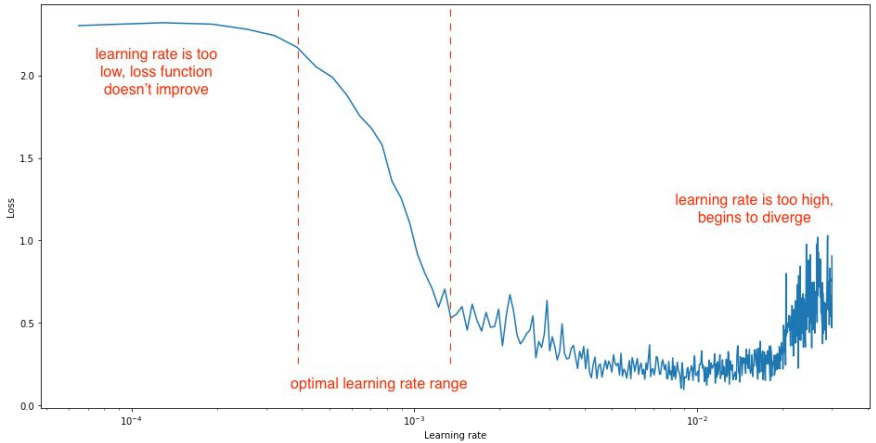
- Where α is called the **learning rate.**





Gradient Descent
Learning Rate



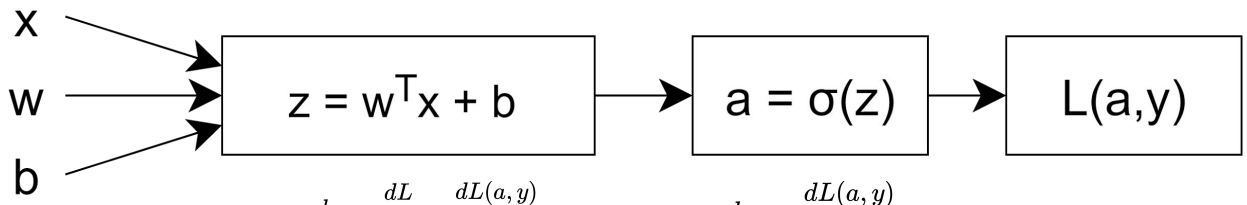


$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

Gradient Descent

Logistic Regression Derivatives



$$dz = \frac{dL}{dz} = \frac{dL(a,y)}{dz}$$
$$= \frac{dL}{da} \cdot \frac{da}{dz}$$
$$= a - y$$

$$da = \frac{dL(a,y)}{da}$$
$$da = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{\partial L}{\partial w} = dw = x \cdot dz$$
$$db = dz$$

$if\ x \epsilon \mathbb{R}^{12},\ a \epsilon \mathbb{R}^{1}$
$then\ w \epsilon \mathbb{R}^{12\times 1},\ b \epsilon \mathbb{R}^{1}$

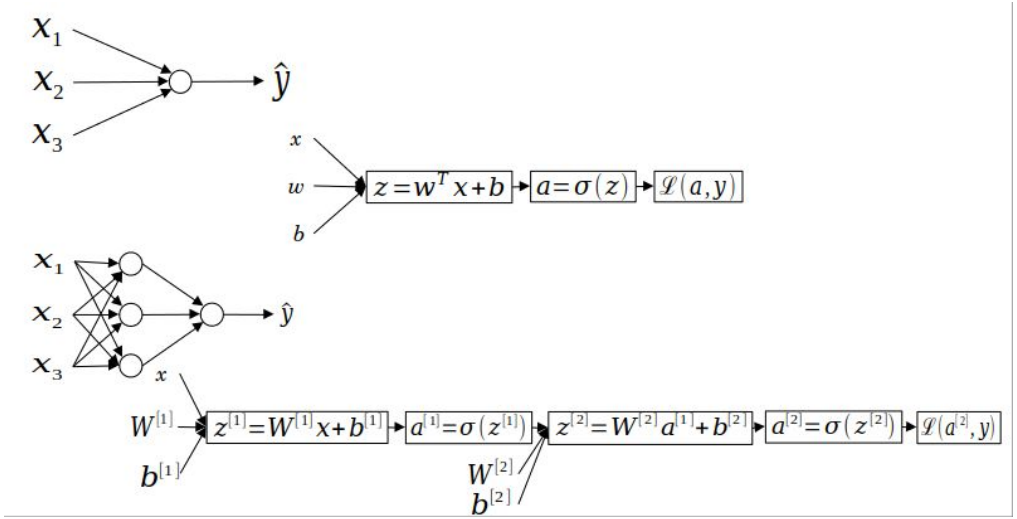
Gradient Descent

Logistic Regression Algorithm

$$
\begin{aligned}
&\textit{while } J > \textit{threshold}: \\
&\quad J = 0;\ dw = 0;\ db = 0 \\
&\quad \textit{for } i = 1 \text{ to } m: \\
&\qquad z^{(i)} = w^T x^{(i)} + b \\
&\qquad a^{(i)} = \sigma\left(z^{(i)}\right) \\
&\qquad J+= -\left[y^{(i)} \log a^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - a^{(i)}\right)\right] \\
&\qquad dz^{(i)} = a^{(i)} - y^{(i)} \\
&\qquad dw+= x^{(i)}.dz(i) \\
&\qquad db+= dz^{(i)} \\
&\quad J/= m;\ dw/= dw;\ db/= m \\
&\quad w := w - \alpha\, dw \\
&\quad b := b - \alpha\, db
\end{aligned}
$$

NN representation

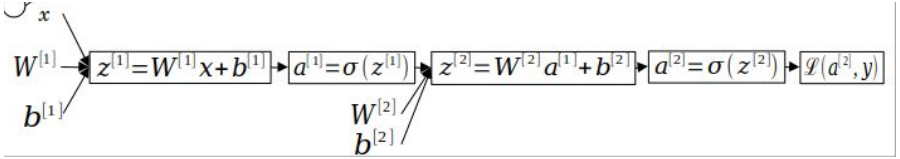
Remember this Figure from Logistic regression, that's basically a single neuron single layer classification neural network.



$if\ x \epsilon \mathbb{R}^3,\ a^{[1]} \epsilon \mathbb{R}^3$
$then\ W^{[1]} \epsilon \mathbb{R}^{3\times3},\ b^{[1]} \epsilon \mathbb{R}^3$

$a^{[2]} \epsilon \mathbb{R}^1$
$then\ W^{[2]} \epsilon \mathbb{R}^{1\times3},\ b^{[2]} \epsilon \mathbb{R}^1$

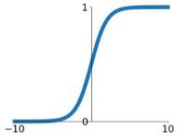
Activation Functions



Activation Functions

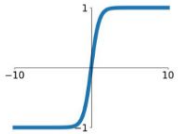
Sigmoid

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



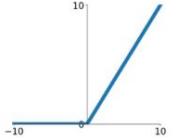
tanh

$\tanh(x)$



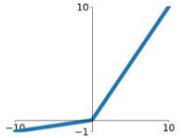
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

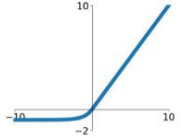


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

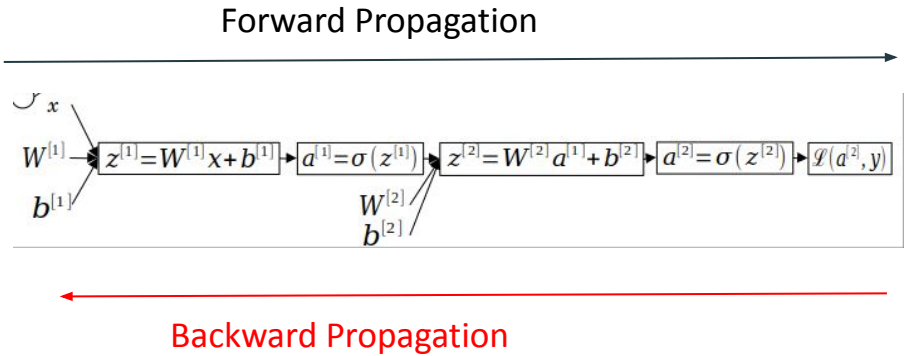
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



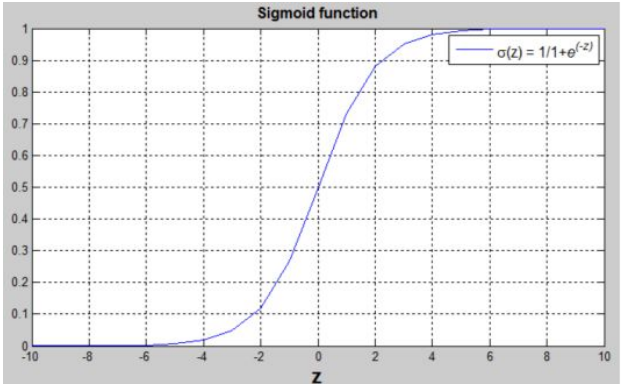
Gradient Descent for NN

- Parameters in terms of W and b
- Cost function J (or L)

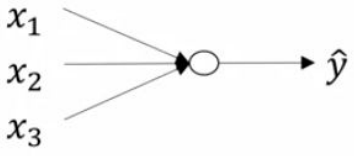


Random Initialization

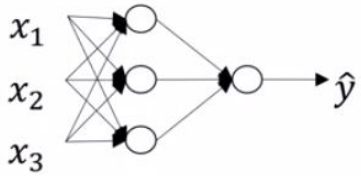
- Initializing W with Zero won’t yield training as the outputs are all the same and the chain rule will train all the weights to be the same.
- It's a good practice to initialize the weights of the neural network with random initialization sampled from a normal distribution with mean = 0 and std = 1. This helps get values good for sigmoid or tanh.



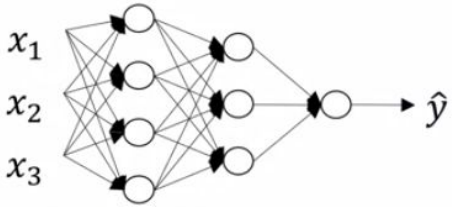
Deep NN



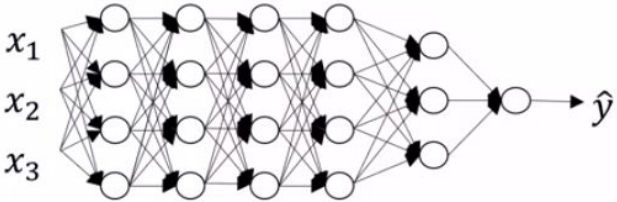
logistic regression



1 hidden layer

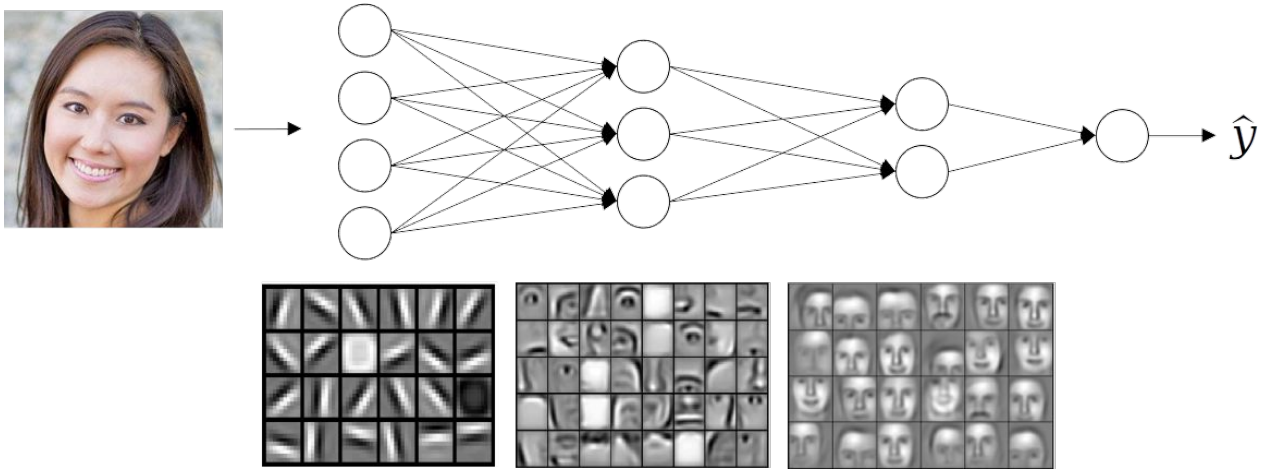


2 hidden layers



5 hidden layers

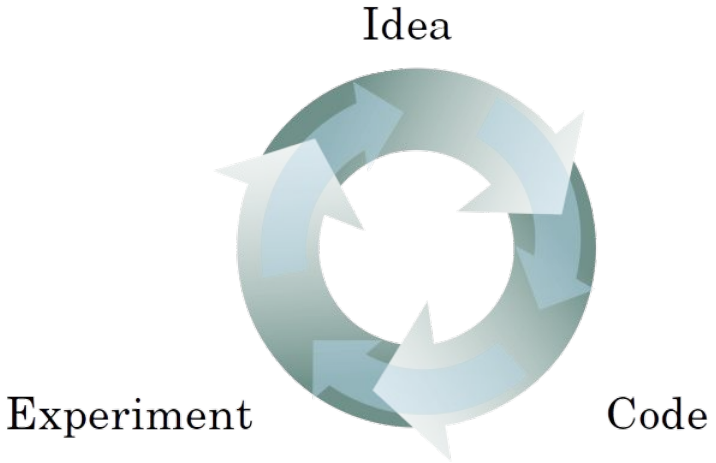
Deep NN



Deep NN

Hyperparameters

- Learning rate
- # iterations
- # hidden layers
- # neurons per layer
- Activation functions



Thanks.
Questions?

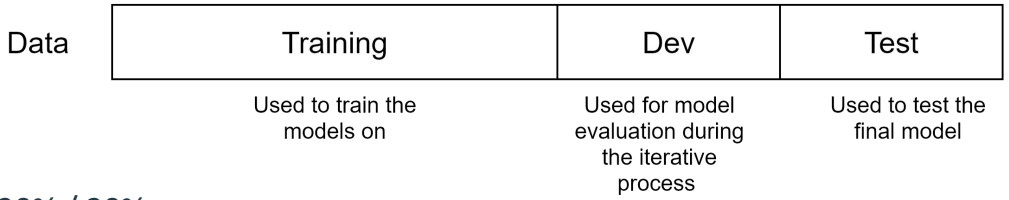
Day 2

Agenda

1. Train/ Dev / Test sets
2. Bias / variance
3. Regularization
4. Data normalization
5. Weights initialization
6. Optimization
7. Hyperparameters tuning
8. Batch norm
9. Softmax classification

Train/ Dev / Test sets

- Like we explained last time, Machine Learning is an iterative process.
- The dataset is split into: train, dev (validation) and test sets



- Good practice: 70% / 30% or 60% / 20% / 20%
- Big Data: data set size: 1,000,000 , it would sufficient to go 98% / 1% / 1%

Train/ Dev / Test sets

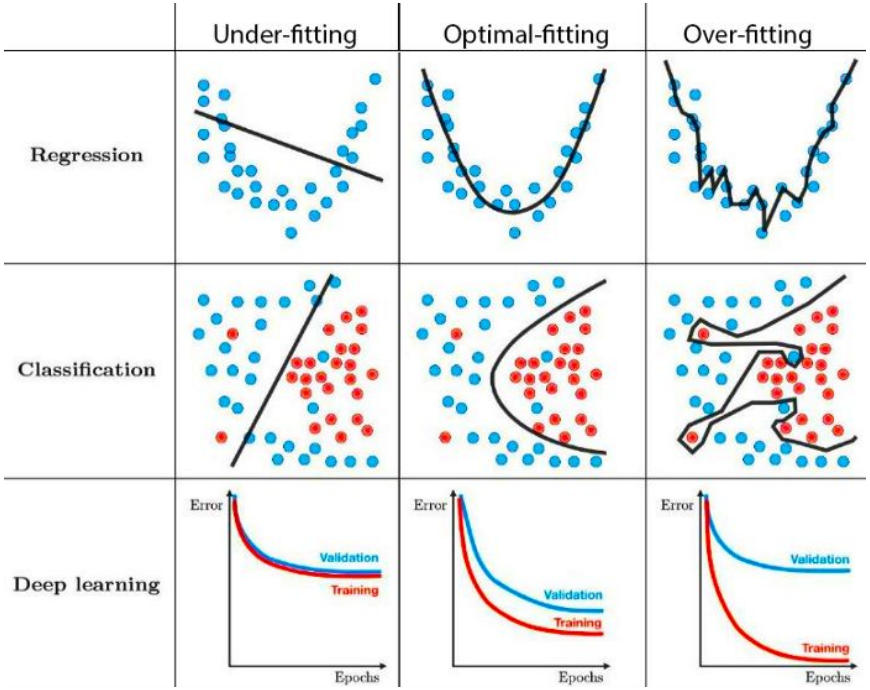
Mismatch of train/test distribution

- If the training set is from a source (images taken from simulation) and the Dev/Test set is from a different source (images taken from a camera). That means that both sets come from different distributions.
- One solution is to split all images (the one from simulation and the ones from a camera) among the train/ dev/ test sets.

Bias / variance

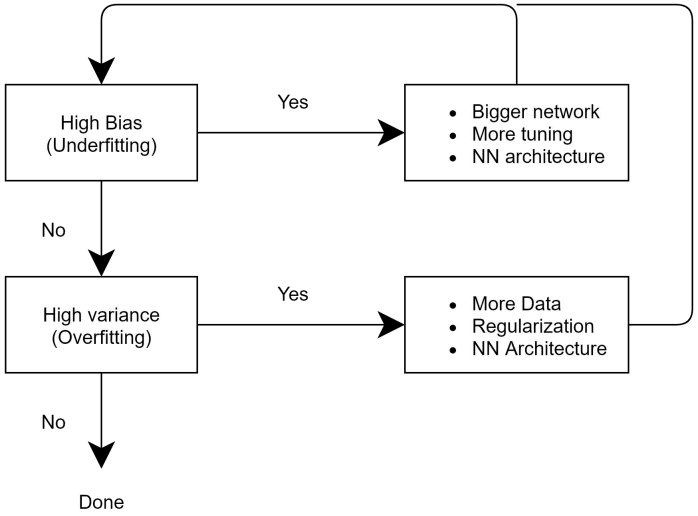
Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%
condition	High variance	High bias	High bias and high variance	Low bias and low variance
comment	overfitting	underfitting		Good fit

Bias / variance



Bias / variance

Solutions



Regularization

L1 & L2

$$L2 : J\left(w^{[i]}, b^{[i]}, \ldots, w^{[L]}, b^{[L]}\right) = \frac{1}{m}\sum_{i=1}^{m} l\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\sum_{l=1}^{L}\left|\left|w^{[l]}\right|\right|_F^2$$

$$\left|\left|w^{[l]}\right|\right|_F^2 = \sum_{i=1}^{n^l}\sum_{j=1}^{n^{[l-1]}}\left(w_{i,j}^{[l]}\right)^2$$

$$dw^{[l]} = \frac{\partial J}{\partial w^{[l]}} + \frac{\lambda}{m}w^{[l]}$$

$$w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$$

$$w^{[l]} := w^{[l]} - \frac{\alpha\lambda}{m}w^{[l]} - \alpha\frac{\partial J}{\partial w^{[l]}}$$

λ is called regularization parameter or weight decay

Regularization

L1 & L2

$$L1 : J\left(w^{[i]}, b^{[i]}, \ldots, w^{[L]}, b^{[L]}\right) = \frac{1}{m}\sum_{i=1}^{m} l\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\sum_{l=1}^{L}\left|\left|w^{[l]}\right|\right|_F$$

$$\left|\left|w^{[l]}\right|\right|_F = \sum_{i=1}^{n^l}\sum_{j=1}^{n^{[l-1]}}\left(\left|w_{i,j}^{[l]}\right|\right)$$

$$dw^{[l]} = \frac{\partial J}{\partial w^{[l]}} + \frac{\lambda}{m}w^{[l]}$$

$$w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$$

$$w^{[l]} := w^{[l]} - \frac{\alpha\lambda}{m}w^{[l]} - \alpha\frac{\partial J}{\partial w^{[l]}}$$

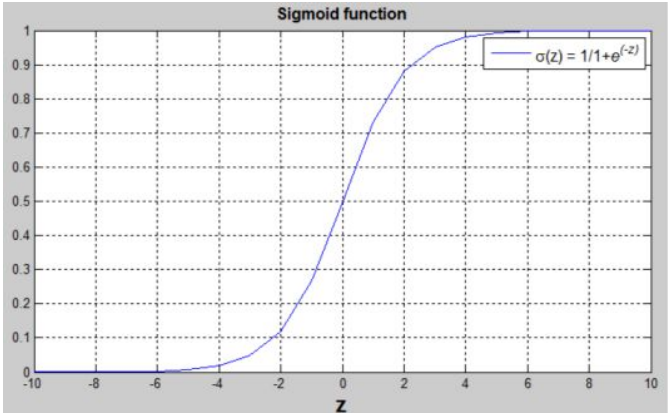
L1 regularization, then w will end up being sparse.

And what that means is that the w vector will have a lot of zeros in it.

Regularization
L1 & L2

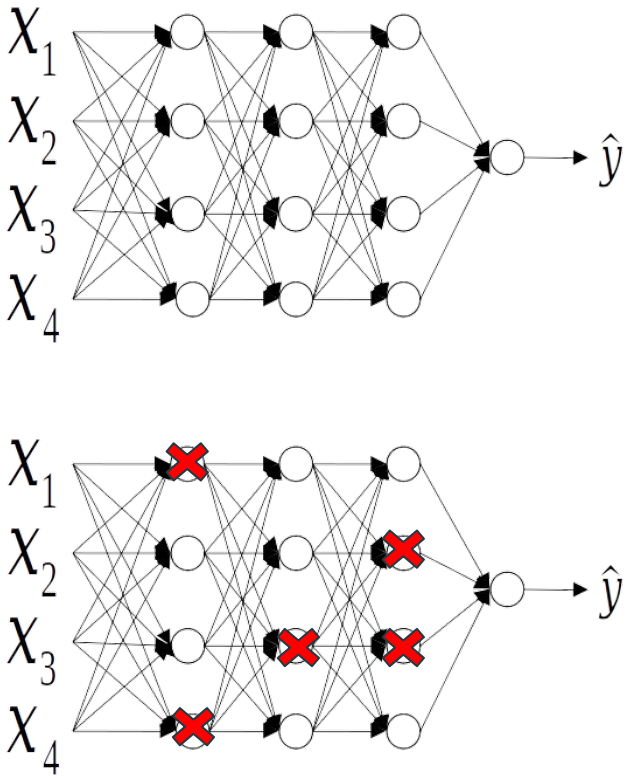
Why does it reduce Overfitting (variance)?

- By intuition, The higher the λ the lower the absolute value of weights will be after training, which in some activations like sigmoid or tanh it will be in the region of max gradients which helps with learning



Regularization
Dropout

- Dropout is counter intuitive.
- You randomly shut down neurons in a layer with probability P (keep_prop=1-p) each iteration.
- Intuition, it forces the NN not to rely on any one feature, so each neuron needs to rely on multiple features during learning.
- Another intuition is that each iteration you train a different smaller NN which is forced to learn.
- Works during training only.
- Each layer has its own dropout setting.



Regularization

Data Augmentation

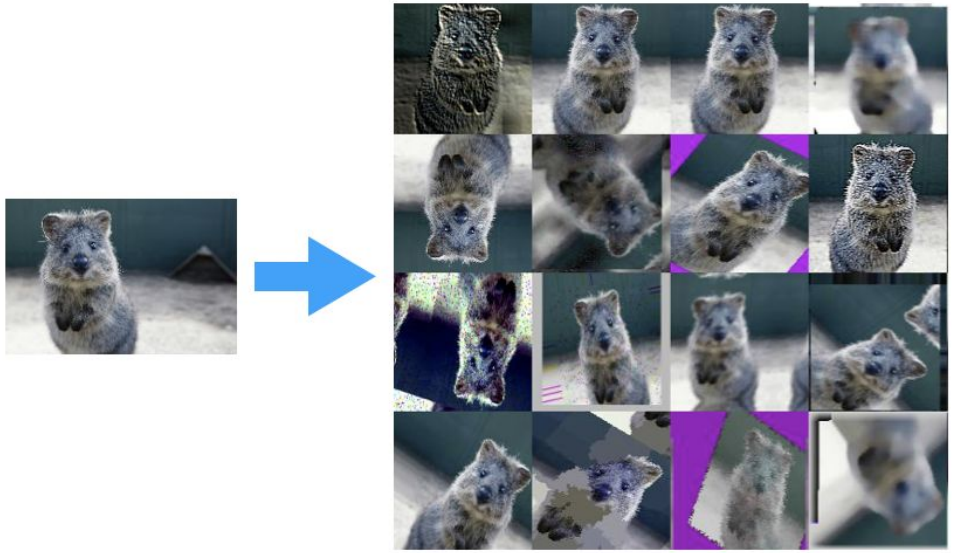
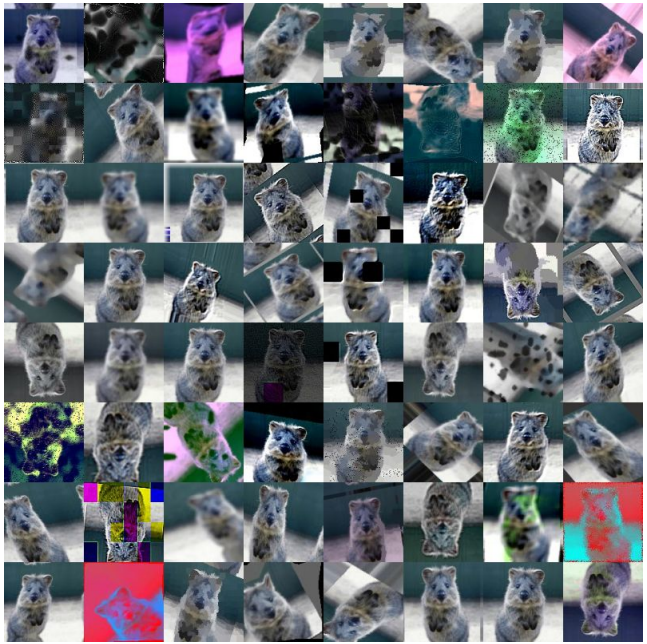
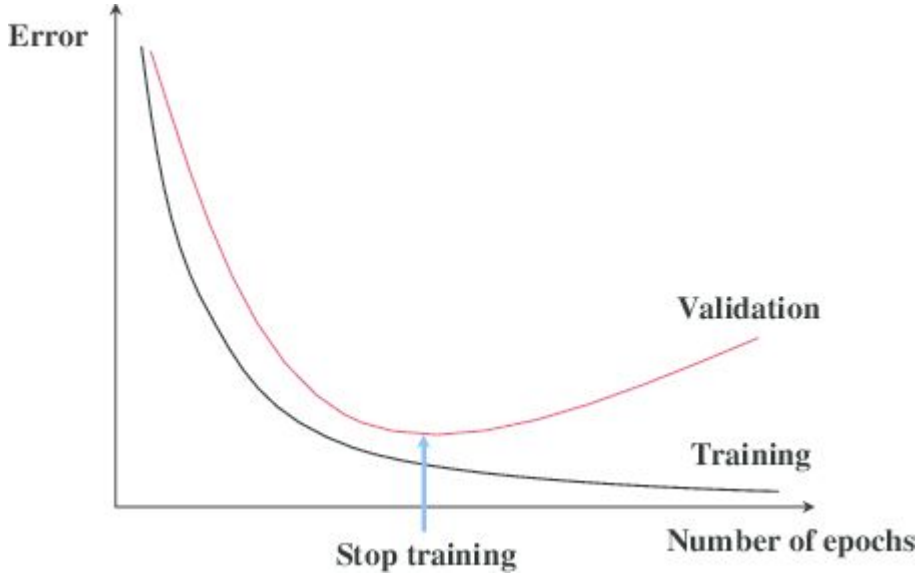


Figure credit: https://github.com/aleju/imgaug



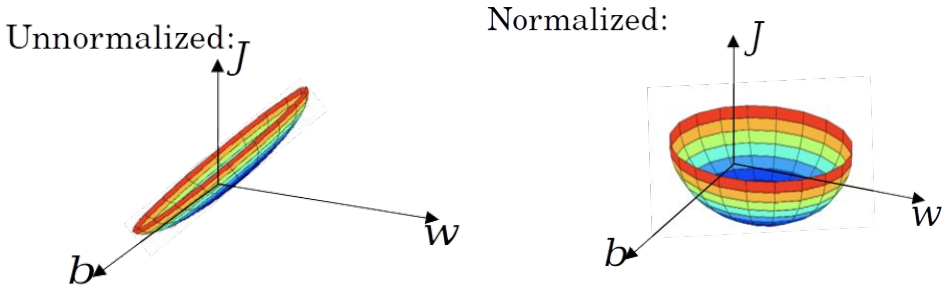
Regularization

Early Stopping



Data normalization

- One technique is to subtract the mean of the training data from the inputs then divide by the standard deviation.
- Apply the same technique on the test and dev set using the same mean and std of the training set.
- This helps to speed up the training. Because each input feature might have a different range:
  - X1 might be [1,1000]
  - X2 might be [0,1]



Weights Initialization

- There is the problem of gradients vanishing or exploding during training.
- The initialization of the weights can be one method to avoid gradient problems.

Initialization	Activation function	Formula
He	ReLU	U(0, sqrt(2/n))
Xavier, Glorot	tanh	U(0, sqrt(1/n))

Optimization Algorithms
Mini-batch GD

- If you have a large training set, it might not fit your memory.
- The solution is dividing the training set into mini-batches that each can fit in your memory.
- Your perform forward and backward propagation on each mini-batch.
- when all mini-batches are processed, this called an epoch.
- It’s good to use mini-batch sizes that are powers of 2 (64,128,256,512).
- Mini-batch updates are noisy.

Optimization Algorithms

Gradient descent with momentum

- Speeds up conversion
- Reduces getting stuck in a local minima
- Beta = 0.9

$$
\begin{aligned}
&On\ iteration\ t: \\
&\quad \text{compute dw, db on current mini-batch} \\
&\quad V_{dw} = \beta V_{dw} + (1-\beta)\, dw \\
&\quad V_{db} = \beta V_{db} + (1-\beta) db \\
&\quad w := w - \alpha V_{dw},\ b := b - \alpha V_{db}
\end{aligned}
$$

Optimization Algorithms
RMSprop

- Speeds up conversion
- Reduces getting stuck in a local minima
- Reduces oscillations of GD

$$
\begin{aligned}
&On\ iteration\ t: \\
&\quad \text{compute dw, db on current mini-batch} \\
&\quad s_{dw} = \beta_2 s_{dw} + (1-\beta_2)\, dw^2 \\
&\quad s_{db} = \beta_2 s_{db} + (1-\beta_2) db^2 \\
&\quad w := w - \alpha \frac{dw}{\sqrt{s_{dw}} + \epsilon},\ b := b - \alpha \frac{db}{\sqrt{s_{db}} + \epsilon}
\end{aligned}
$$

Optimization Algorithms
Adam

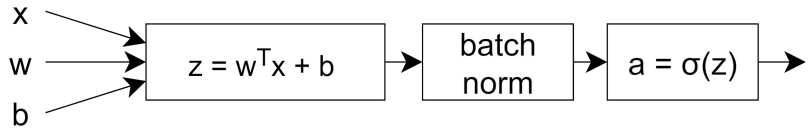
- A combination of momentum and RMSProp
- Speeds up conversion
- Reduces getting stuck in a local minima
- Reduces oscillations of GD
- Beta1 = 0.9, beta2=0.999, epsilon=1e-8

$$
\begin{aligned}
&On\ iteration\ t:\\
&\quad \text{compute dw, db on current mini-batch}\\
&\quad V_{dw} = \beta_1 V_{dw} + (1-\beta_1)\, dw\\
&\quad V_{db} = \beta_1 V_{db} + (1-\beta_1) db\\
&\quad s_{dw} = \beta_2 V_{dw} + (1-\beta_2)\, dw^2\\
&\quad s_{db} = \beta_2 V_{db} + (1-\beta_2) db^2\\
&\quad V_{dw} = V_{dw}/\left(1-\beta_1^t\right),\ V_{db} = V_{db}/\left(1-\beta_1^t\right)\\
&\quad s_{dw} = s_{dw}/\left(1-\beta_2^t\right),\ s_{db} = s_{db}/\left(1-\beta_2^t\right)\\
&\quad w := w - \alpha \frac{V_{dw}}{\sqrt{s_{dw}}+\epsilon},\ b := b - \alpha \frac{V_{db}}{\sqrt{s_{db}}+\epsilon}
\end{aligned}
$$

Hyperparameters tuning

- There are lots of hyperparameters to tune.
- It may take a long time to find good hyperparameters.
- But keep in mind, you are mainly a practitioner not a researcher, lookup papers that did the application you are doing and use their hyperparameters.

Batch Norm



- Normalizing (z) across the minibatch before it goes into the activation function.
- It speeds up training as it is doing a similar thing to the normalising inputs
- During testing, the mean of the means and stds used during training is used.

$$\mu = \frac{\sum_i z^{(i)}}{m}$$

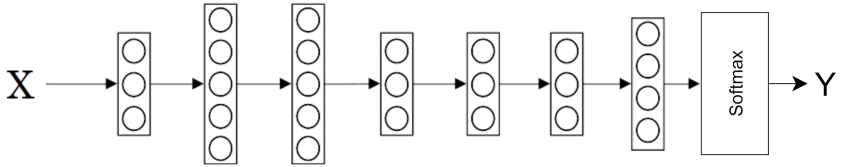
$$\sigma^2 = \frac{\sum_i \left(z^{(i)} - \mu\right)^2}{m}$$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\hat{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

γ, β are learnable parameters

Softmax Classifier



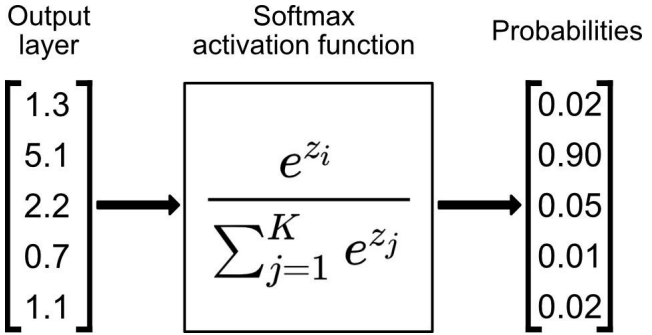
- Used for multi class classification.
- It is the activation function of the final layer.
- It outputs a probability distribution over all classes.
- The labels should be one hot encoded

Human-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Machine-Readable

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0



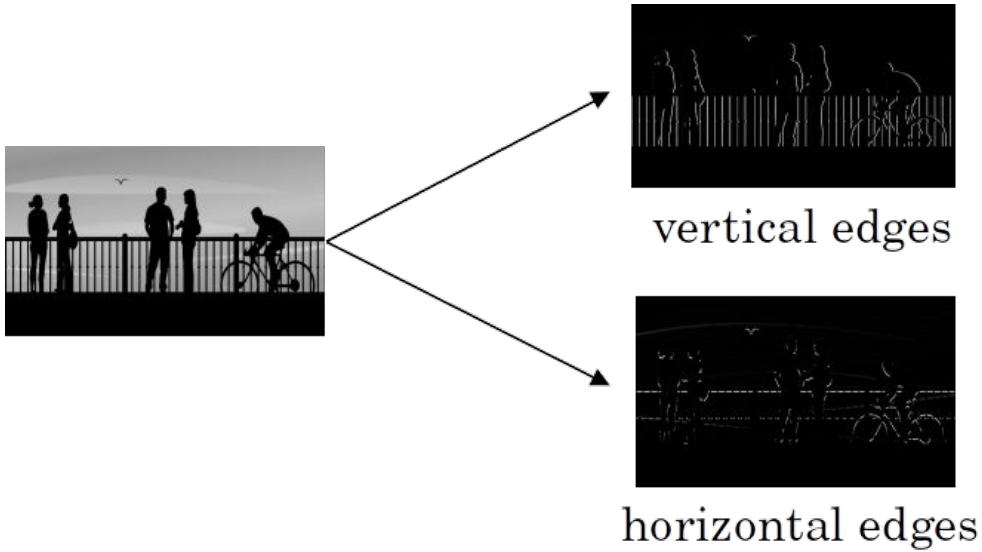
Thanks.
Questions?

Day 3

Agenda

1. Computer vision (CV) filters
2. One layer Convolutional Neural Network (CNN)
3. Pooling layers
4. CNN
5. Lenet
6. AlexNet
7. VGG-16
8. Resnets
9. Inception
10. Transfer learning
11. Object detection:
12. Segmentation

Computer vision (CV) filters



Computer vision (CV) filters

Vertical Edge detection

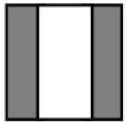
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

*



Computer vision (CV) filters

Vertical Edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

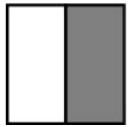
*

1	0	-1
1	0	-1
1	0	-1

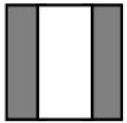
10*1	10*0	10*-1
10*1	10*0	10*-1
10*1	10*0	10*-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

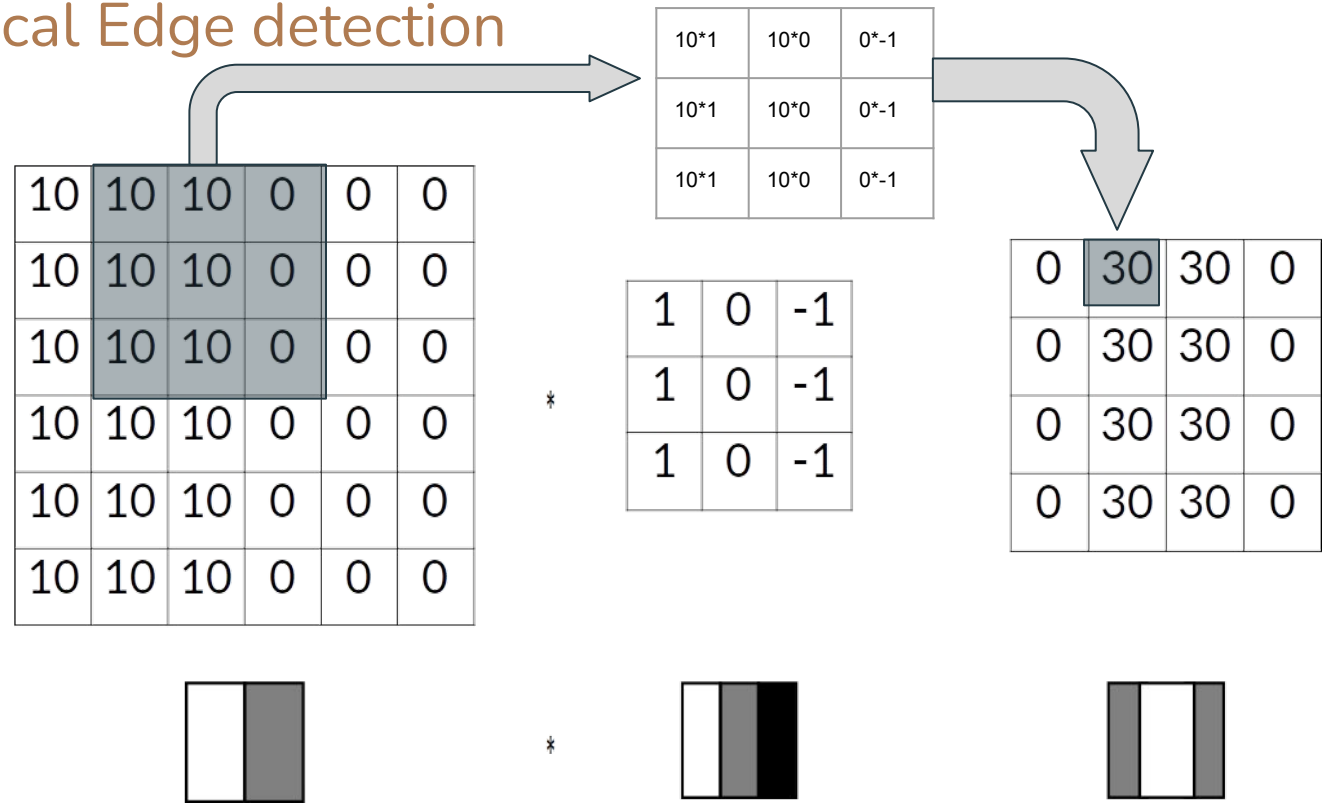


*



Computer vision (CV) filters

Vertical Edge detection



Computer vision (CV) filters

Vertical Edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

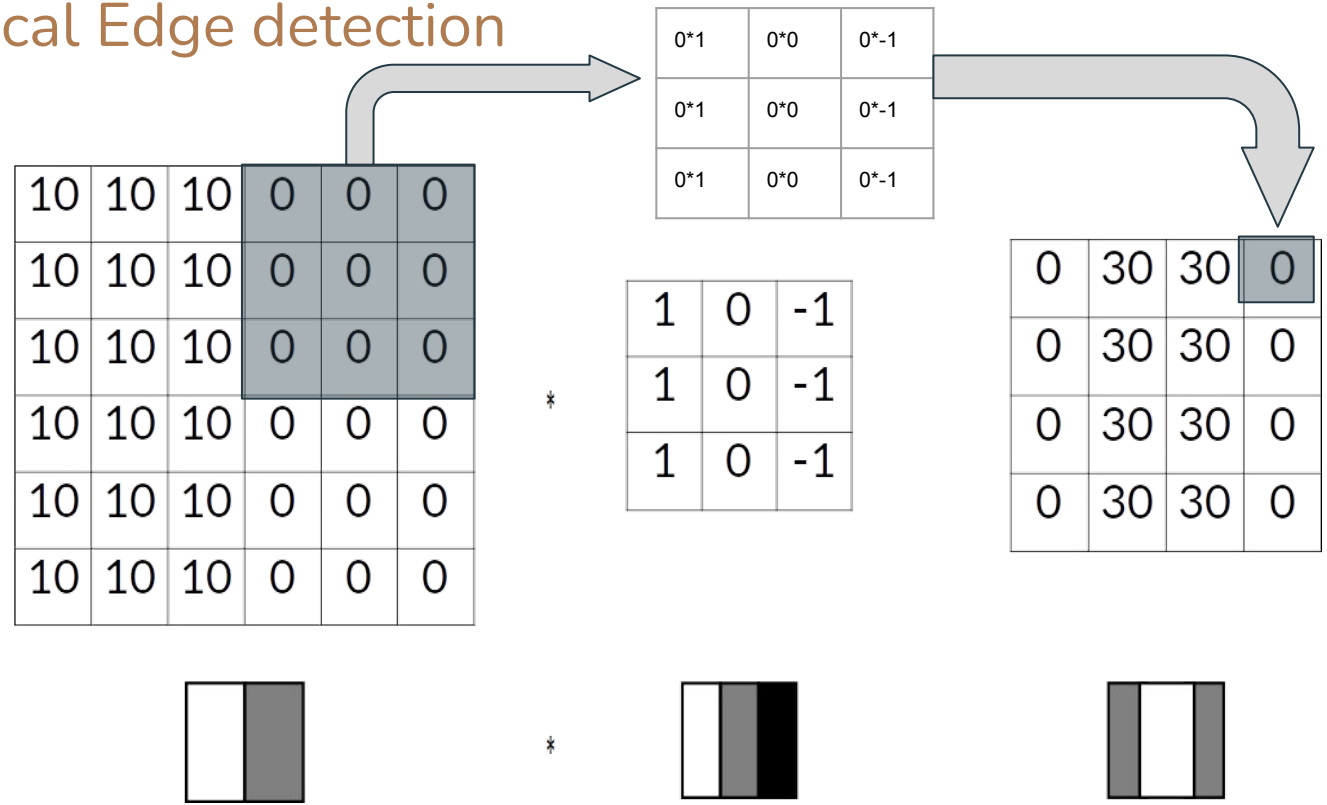
1	0	-1
1	0	-1
1	0	-1

10*1	0*0	0*-1
10*1	0*0	0*-1
10*1	0*0	0*-1

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

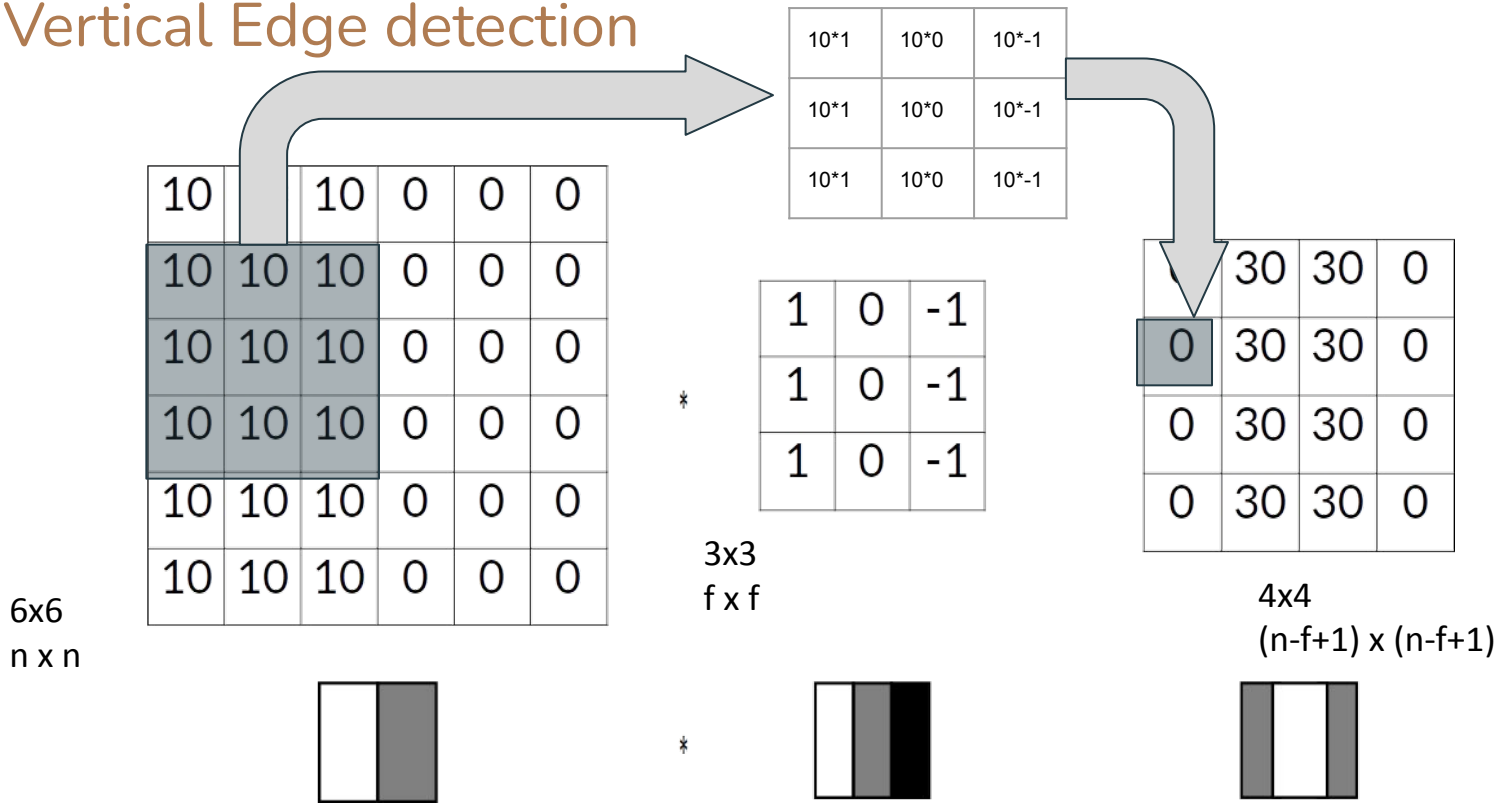
Computer vision (CV) filters

Vertical Edge detection



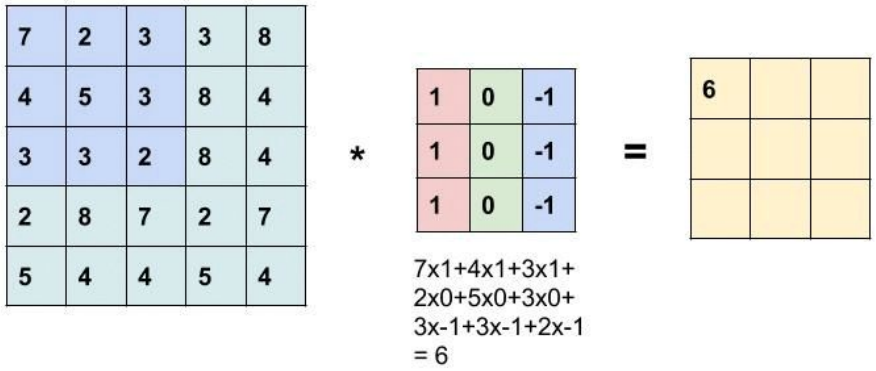
Computer vision (CV) filters

Vertical Edge detection



Computer vision (CV) filters

Vertical Edge detection



Computer vision (CV) filters

Horizontal Edge detection

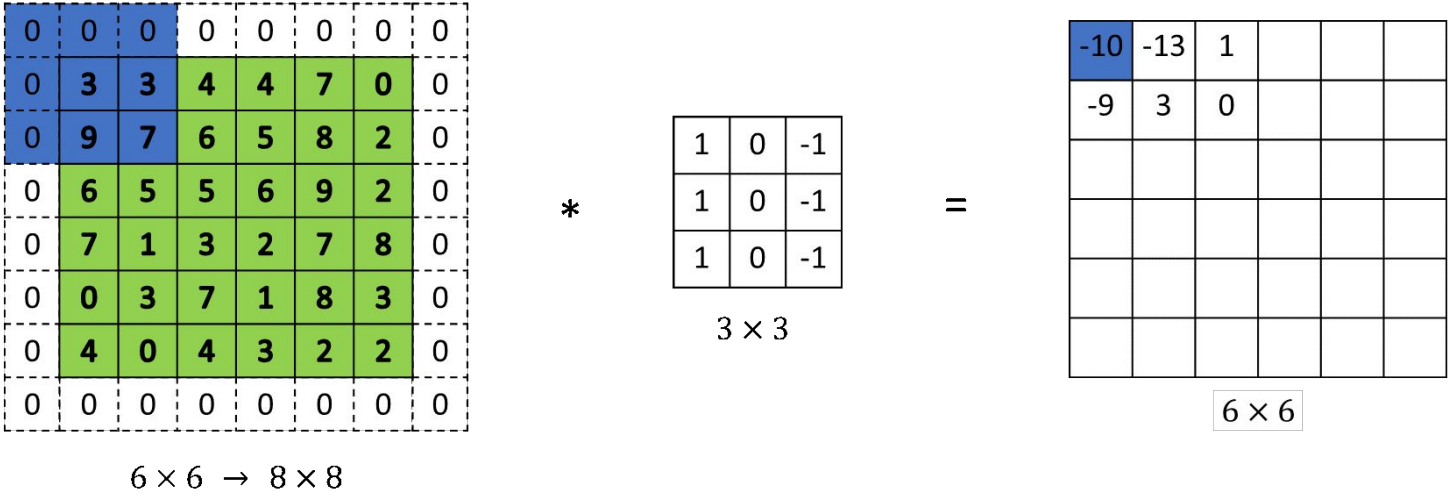
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

1	1	1
0	0	0
-1	-1	-1

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Computer vision (CV) filters

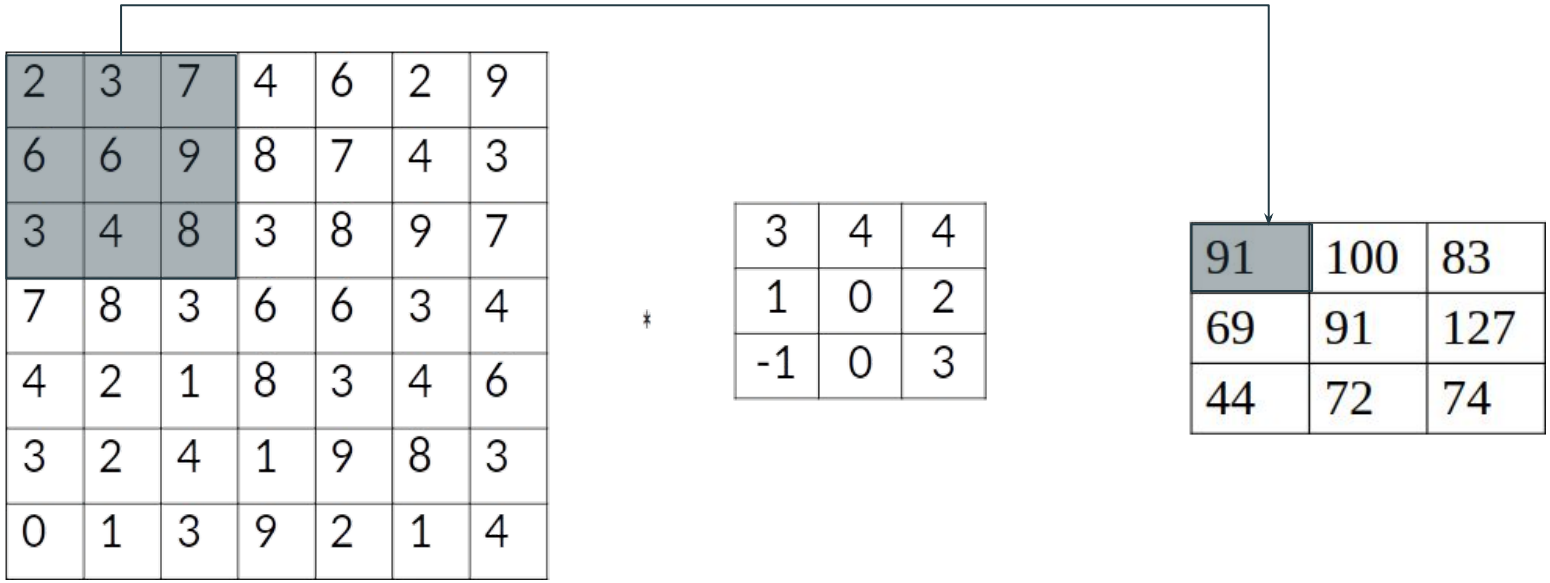
Padding



- Valid convolution: Convolution with no padding
- Same convolution: Convolution with padding. Amount to pad is P=(f-1)/2

Computer vision (CV) filters

Strided Convolution



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

*

3	4	4
1	0	2
-1	0	3

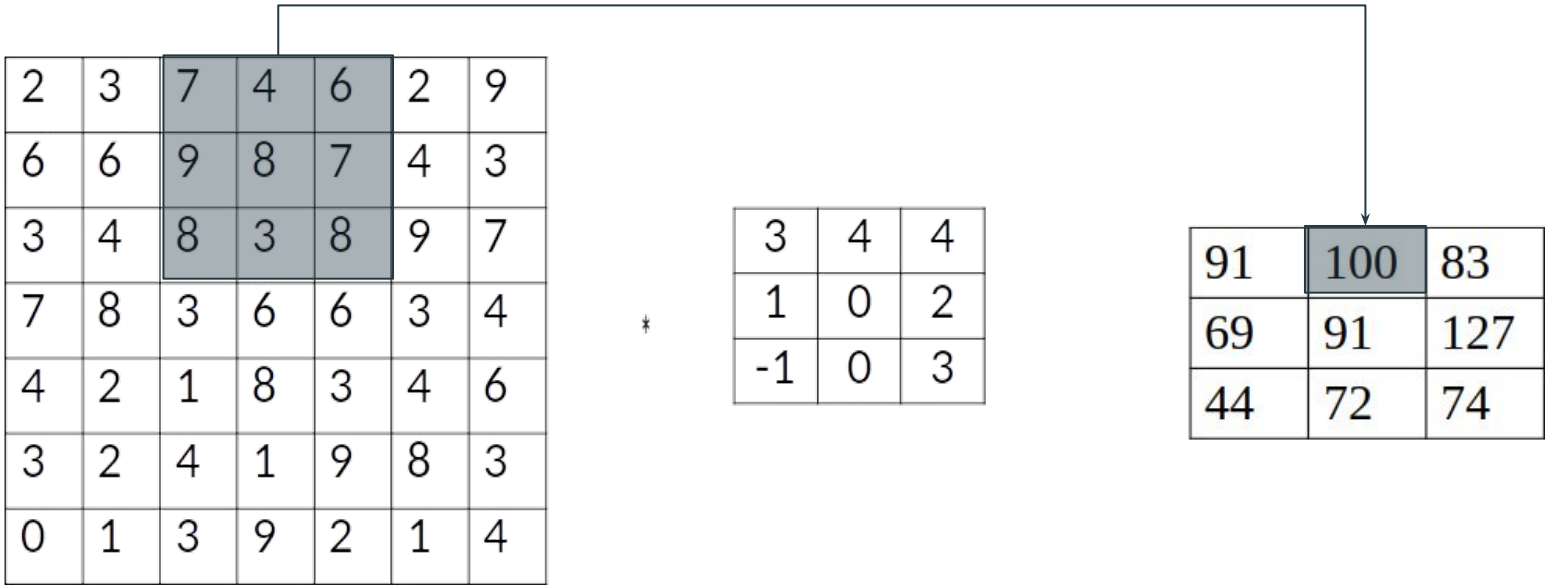
=

91	100	83
69	91	127
44	72	74

Padding p = 0, stride s = 2

Computer vision (CV) filters

Strided Convolution



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

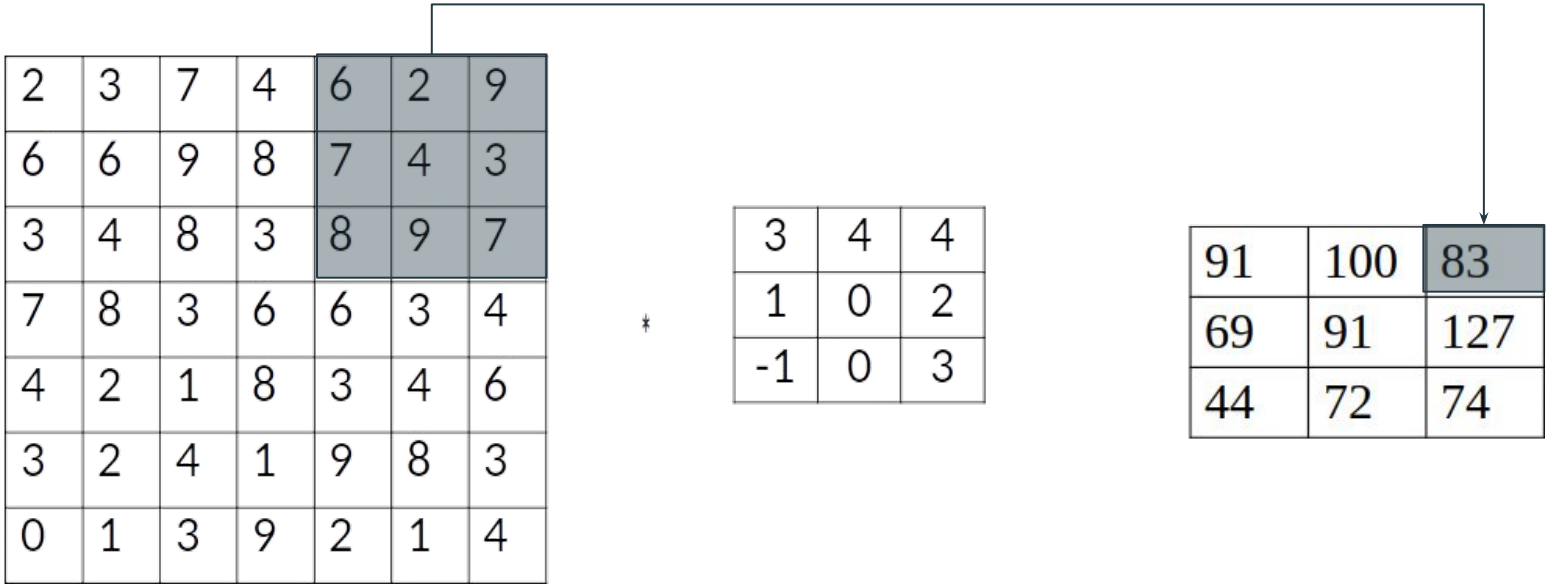
*

3	4	4
1	0	2
-1	0	3

91	100	83
69	91	127
44	72	74

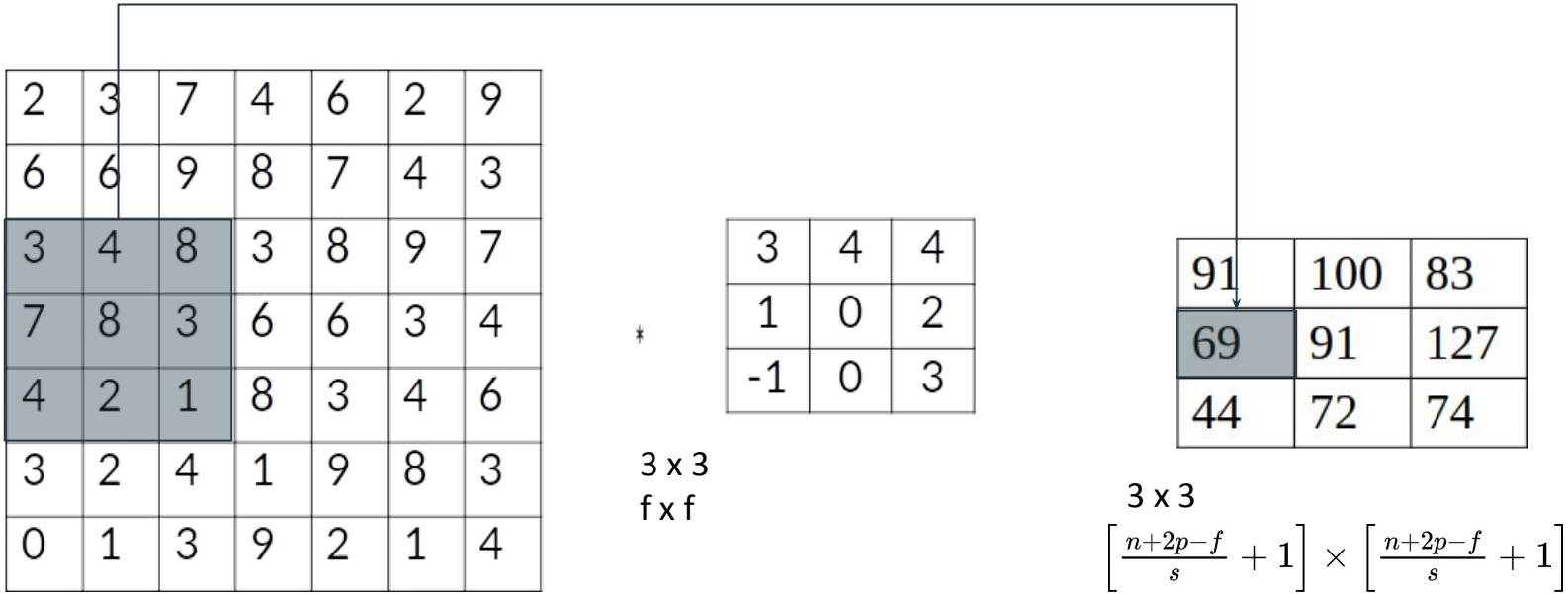
Computer vision (CV) filters

Strided Convolution

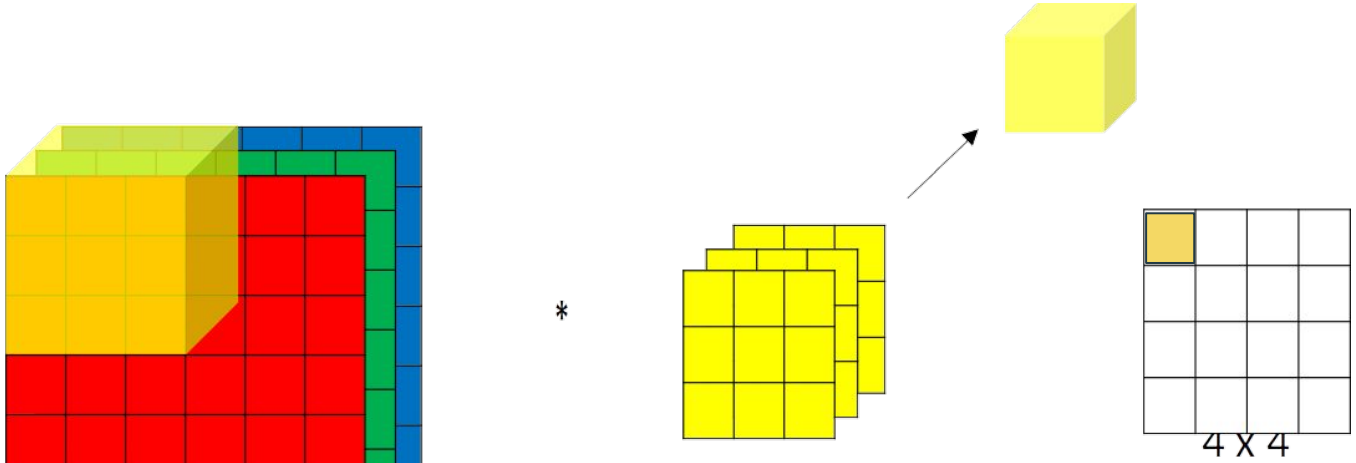


Computer vision (CV) filters

Strided Convolution



Computer vision (CV) filters
3D Convolution



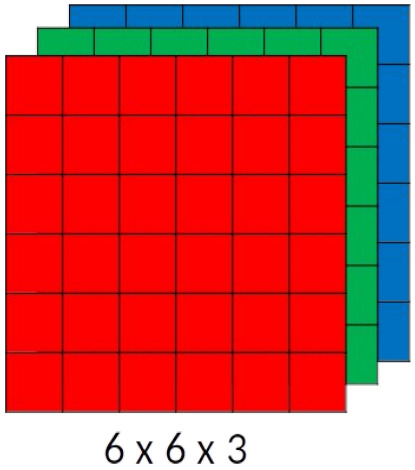
6 x 6 x 3

3 x 3 x 3

- # of channels must be the same in the image and filter

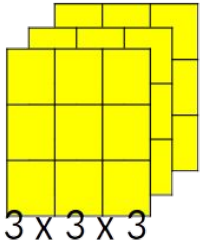
Computer vision (CV) filters

Multiple Filters

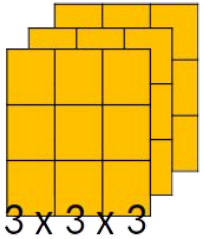


*

f x f x c

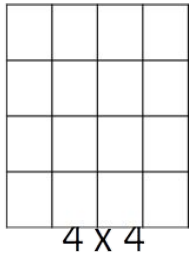


*

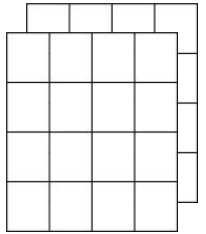


f x f x c

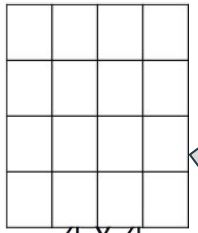
$\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right]$



$\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right]$ x nf



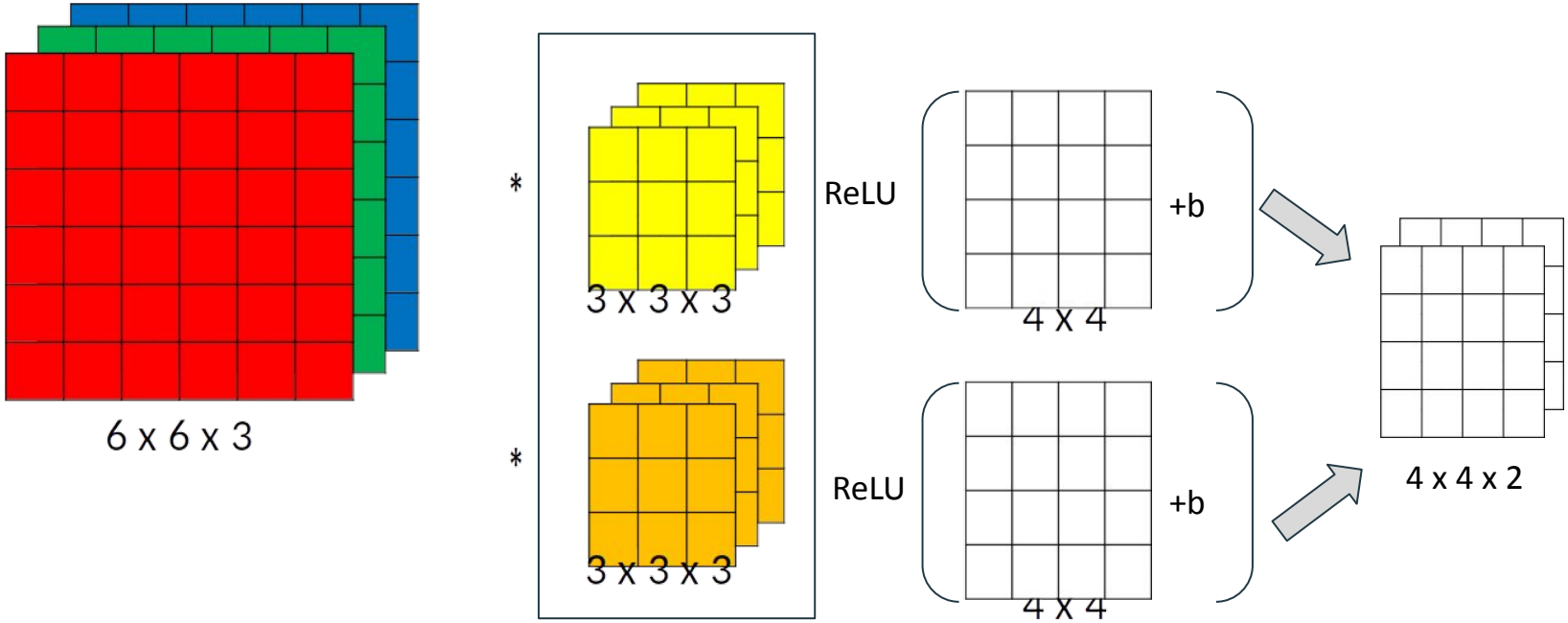
4 x 4 x 2



$\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right]$

- # of filters = nf = 2

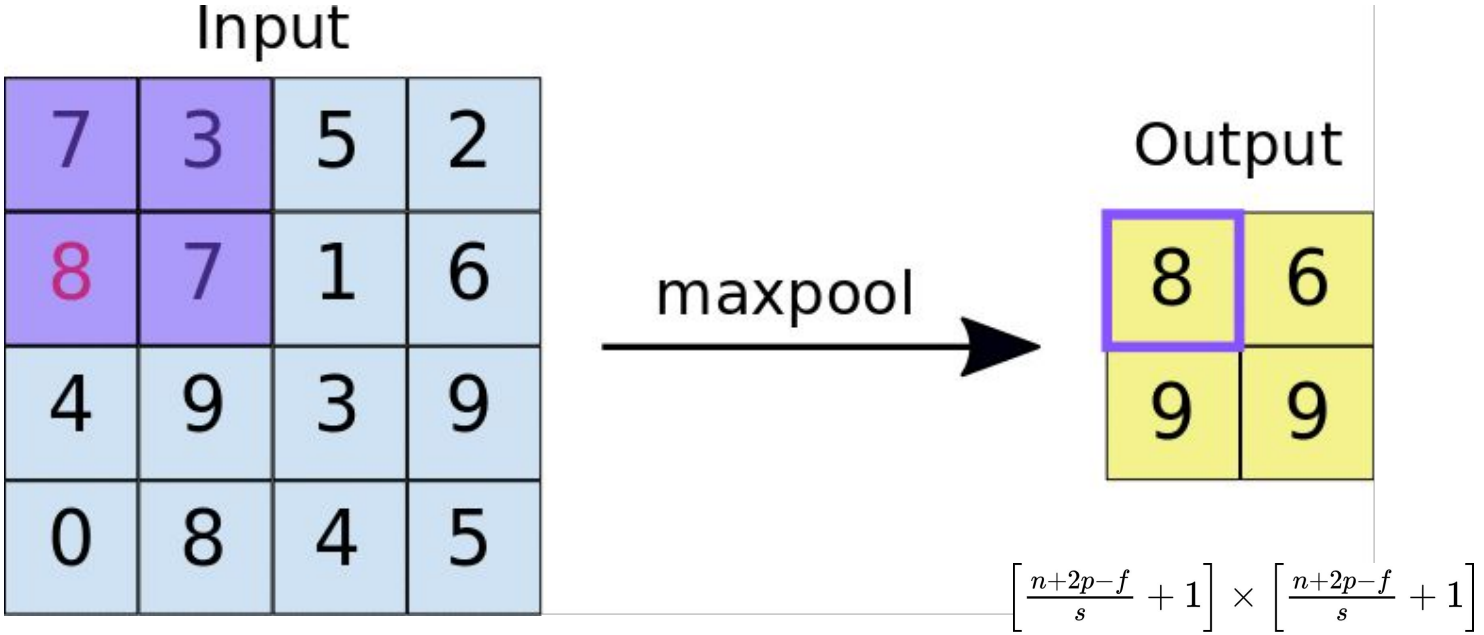
One Layer of a Convolutional Network



The values of the filters are being learnt

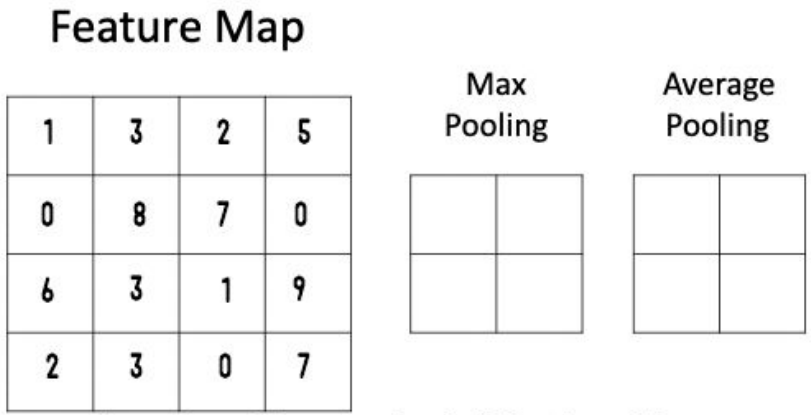
Pooling Layer

Max Pooling



$$\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right]$$

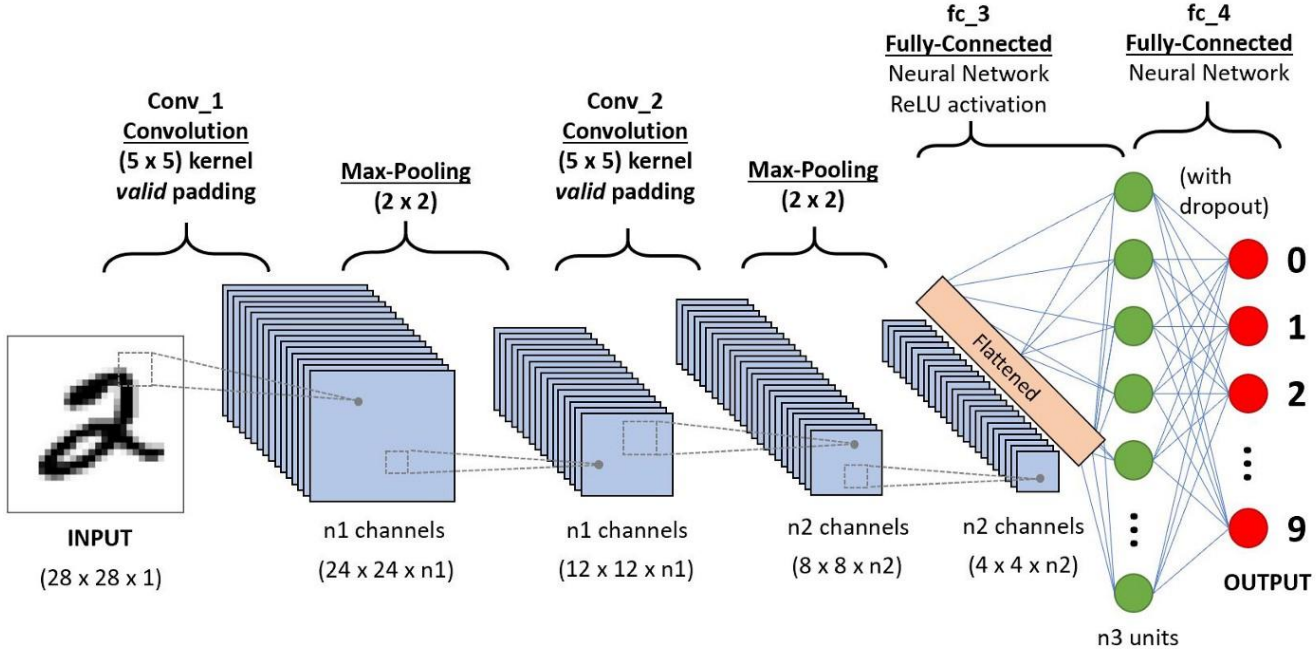
Pooling Layer
Average Pooling



Max pooling and Average pooling of a 4x4 input map with a 2x2 pooling window and stride = 2
thedatabus.io

$$\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right]$$

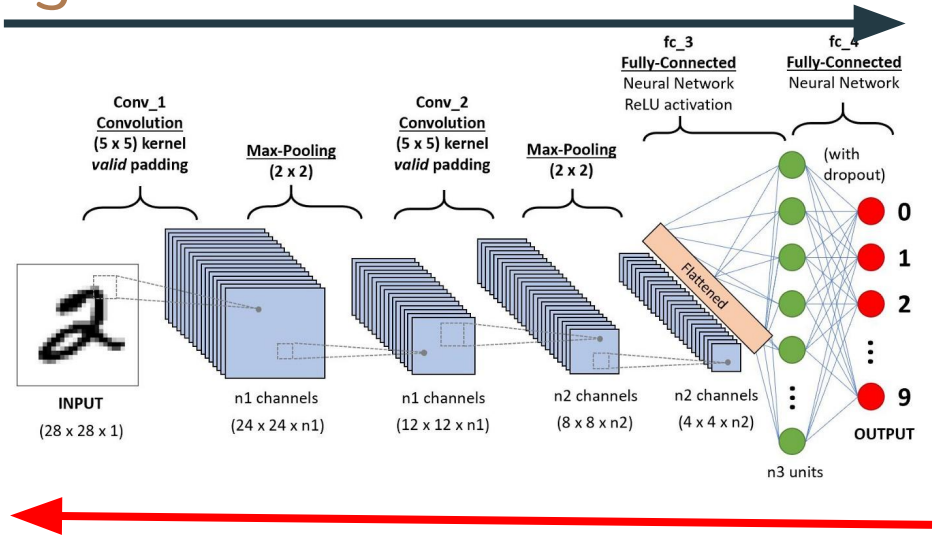
CNN


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with
dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

CNN

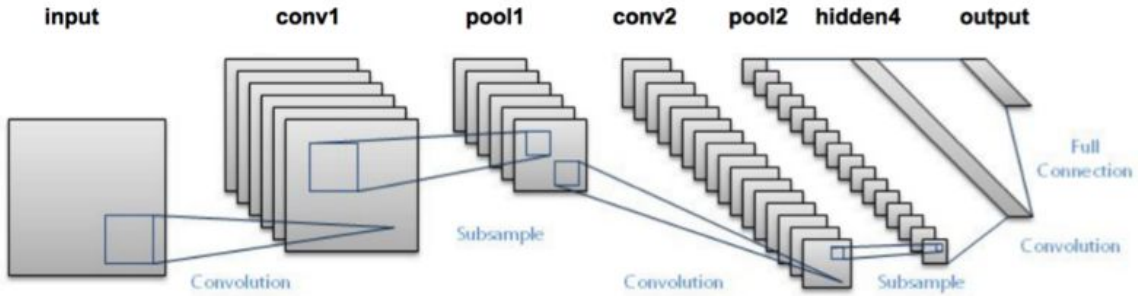
- Convolution layers are useful in image analysis as:
  - Each conv layer has smaller number of parameters than an equivalent Dense (FC) layer
  - Parameter sharing, i.e. A feature detector (such as a vertical edge detector) that’s useful in one part of the image is probably useful in another part of the image.
  - Sparsity of connections: In each layer, each output value depends only on a small number of inputs.

CNN
Put it all together



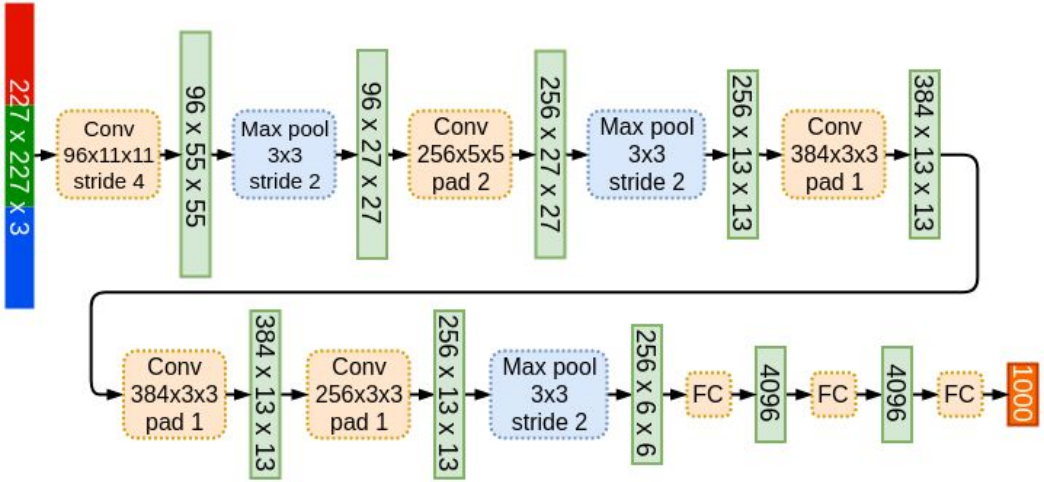
loss

Lenet



- The First CNN (1998) By Yan Lecun
- Trained on Mnist

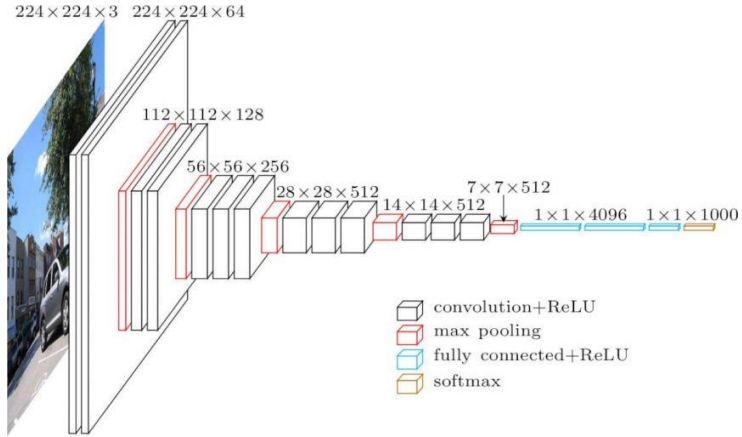
AlexNet

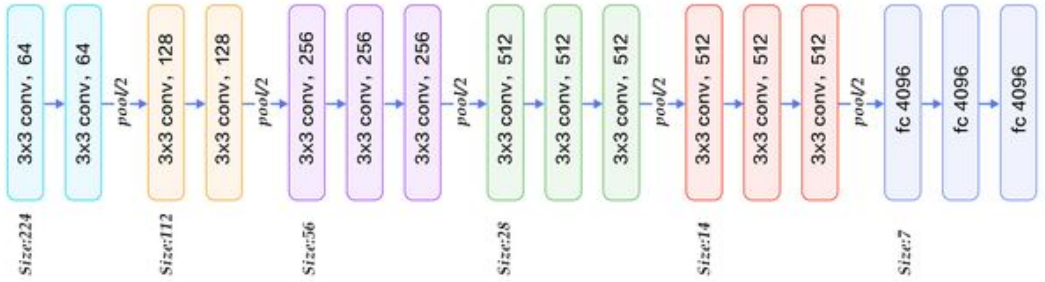


- The First paper to use GPU (2012) for training and started the deep learning era.
- Trained on Imagenet and won the competition.

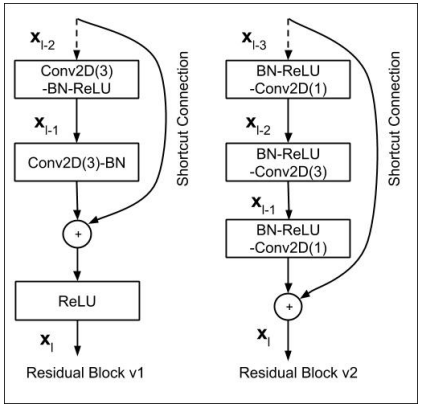
VGG-16

- Improved over Alexnet
- Trained on Imagenet.



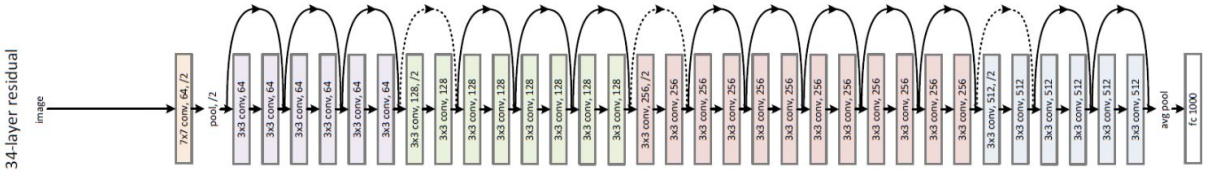


ResNet
Residual Block



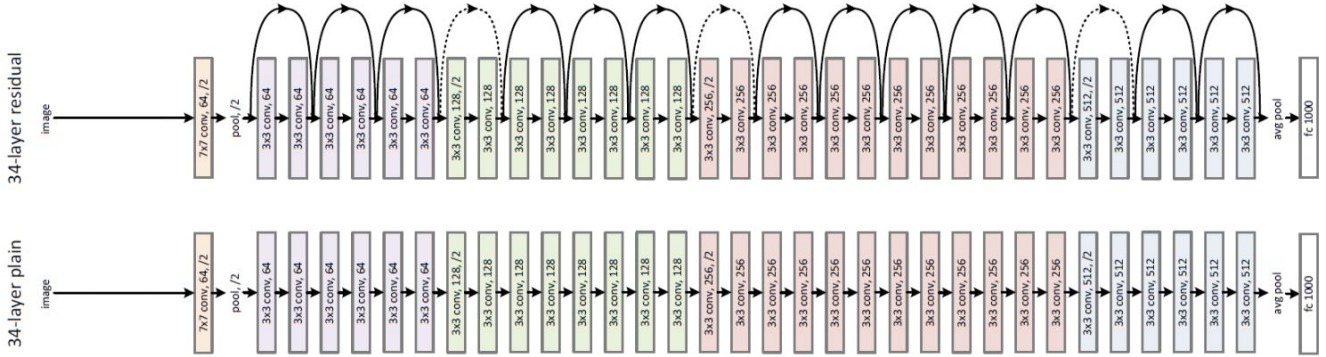
- Residual Blocks have skip connections.
- Residual Blocks allow for training much deeper neural networks (> 100 layers).

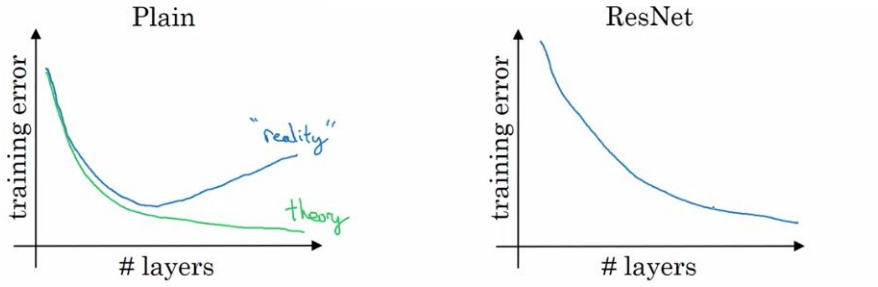
ResNet



- Residual Blocks allow for training much deeper neural networks (> 100 layers)

ResNet
Plain Vs ResNet





ResNet

- It works because as the layers go deeper, the weights of one layer in the middle can have very small values (nearly zero) that the neurons produce small outputs.
- The skip connection acts as a bridge that skips such layer.

Inception

1x1 Convolution

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

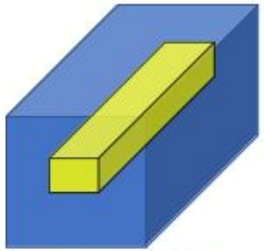
6 x 6

*

2

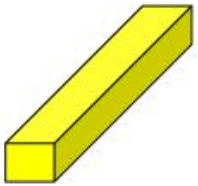
=

2	4	6	12	10	16
6	10	10	2	6	8
4	2	6	8	18	6
8	14	16	10	14	18
2	10	6	14	8	16
10	8	18	16	6	10



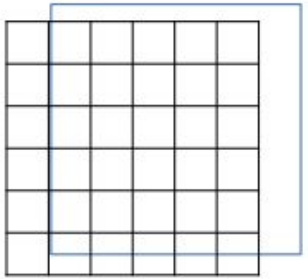
6 x 6 x 32

*



1 x 1 x 32

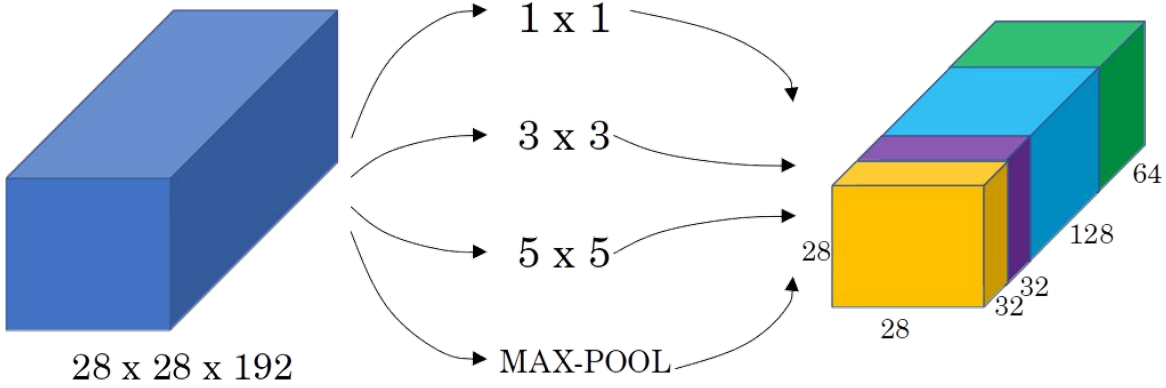
=



6 x 6 x # filters

Inception

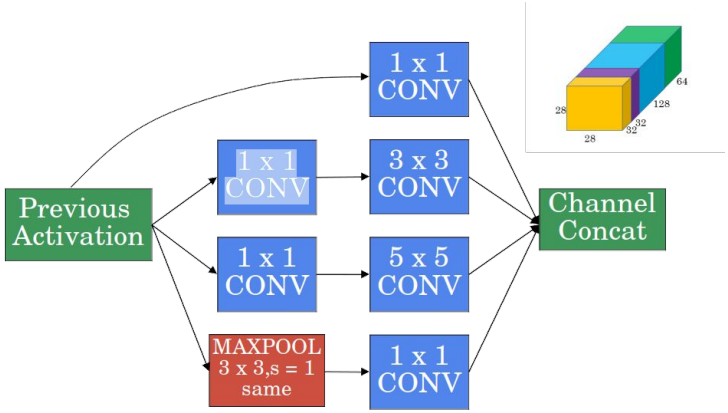
Inception block



- It combines multiple kernel sizes.
- The model learns which kernel sizes it needs
- Computational expensive

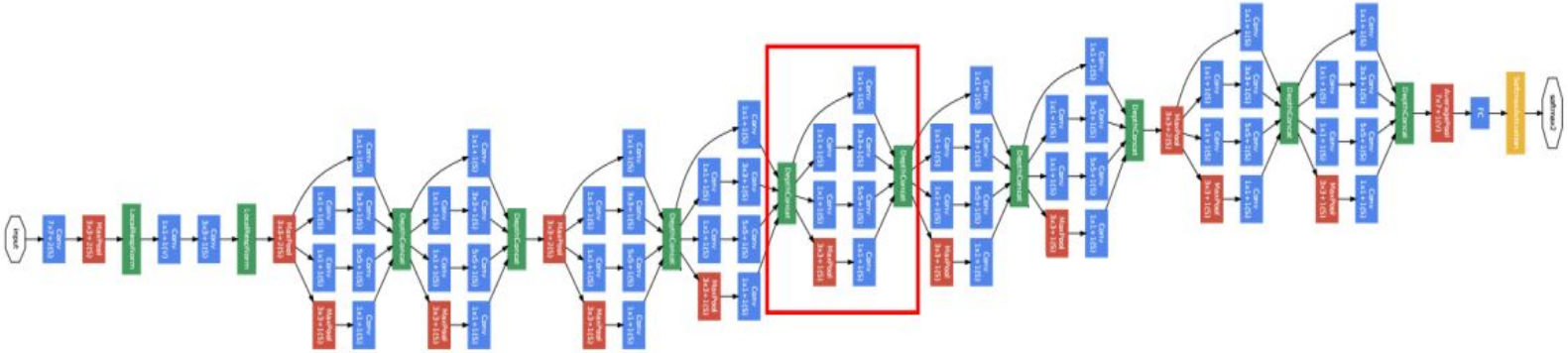
Inception

Inception block



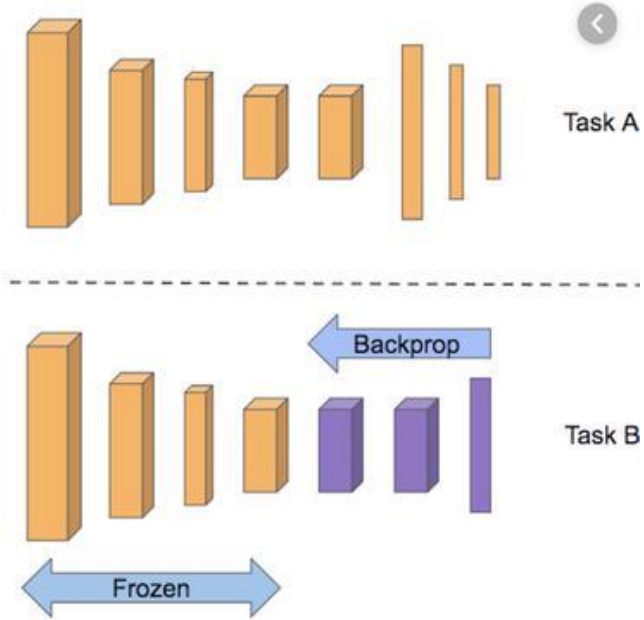
- Reduce the computation requirements 10 times by adding the extra 1x1 conv.

Inception

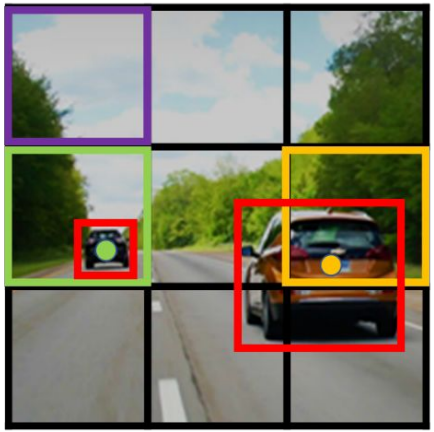


Transfer Learning

- Lots of researchers have already trained their CNN and published the weights.
- Works well especially if you have a small dataset
- You can leverage the trained models and use their weights as starting weights for you model.
- You can also do fine tuning.



Object Detection
YOLO-labels

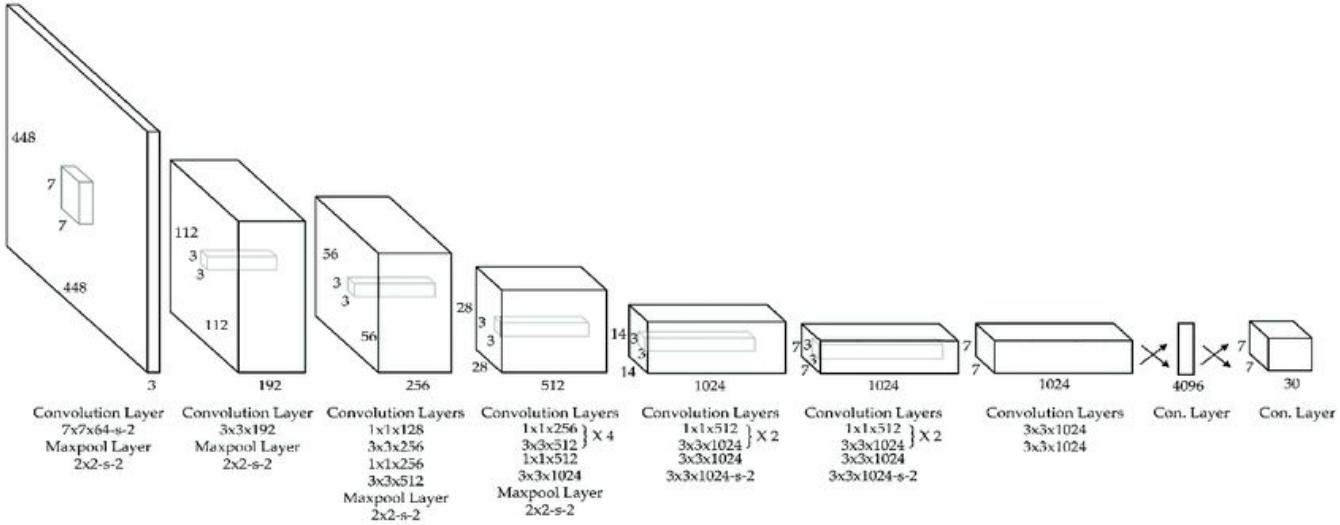


Labels for training for each grid cell:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

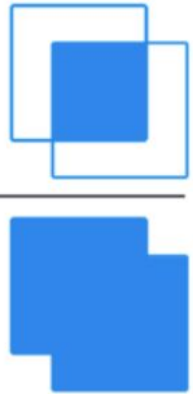
→ bounding box

Object Detection
YOLO

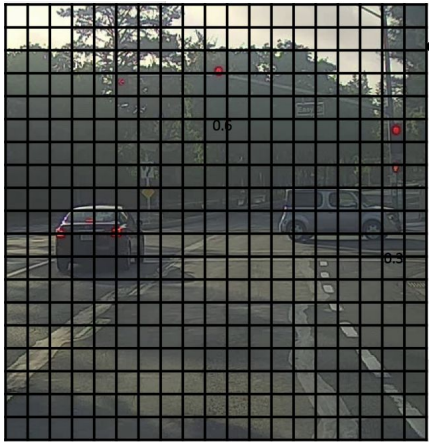


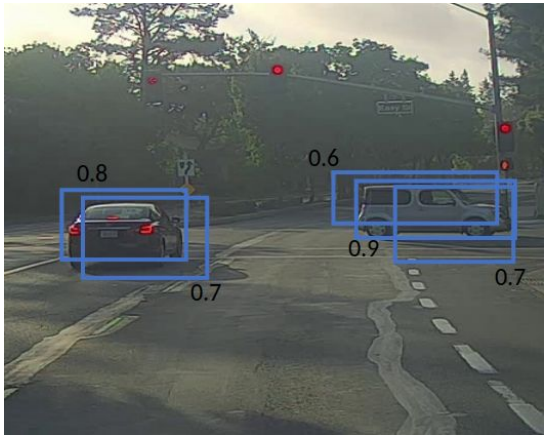
Object Detection
IOU

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

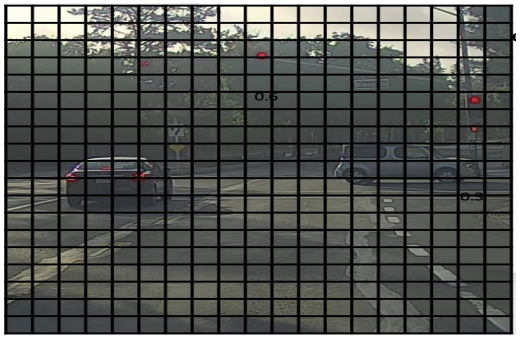


Object Detection
Non-max Suppression





Object Detection
Non-max Suppression



19×19

Each output prediction is:

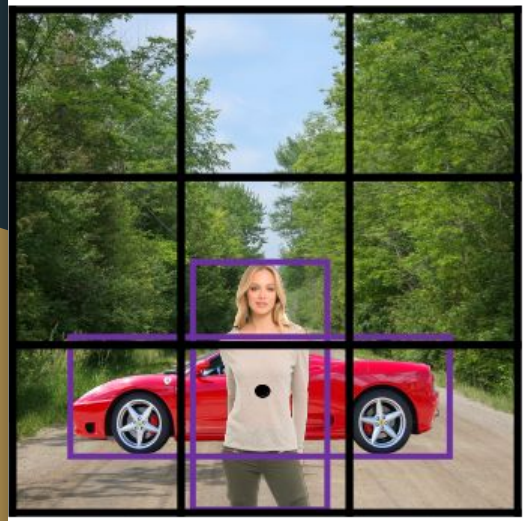
$$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$$

Discard all boxes with $p_c \leq 0.6$

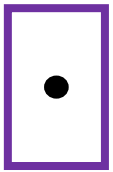
While there are any remaining boxes:

- Pick the box with the largest p_c Output that as a prediction.
- Discard any remaining box with IoU ≥ 0.5 with the box output in the previous step

Object Detection
Anchor box



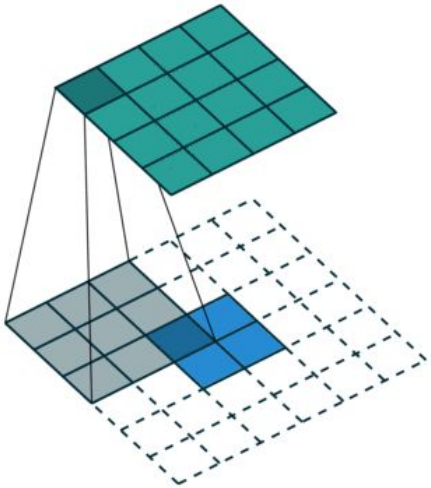
Anchor box 1:

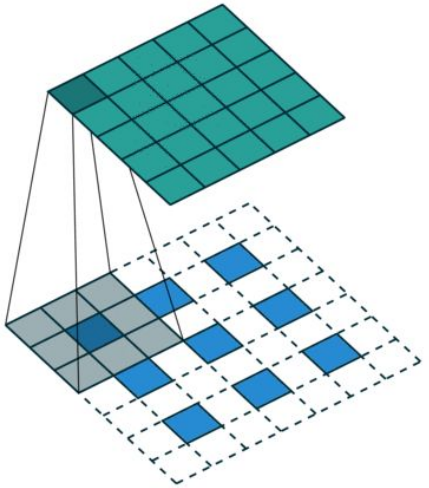


Anchor box 2:

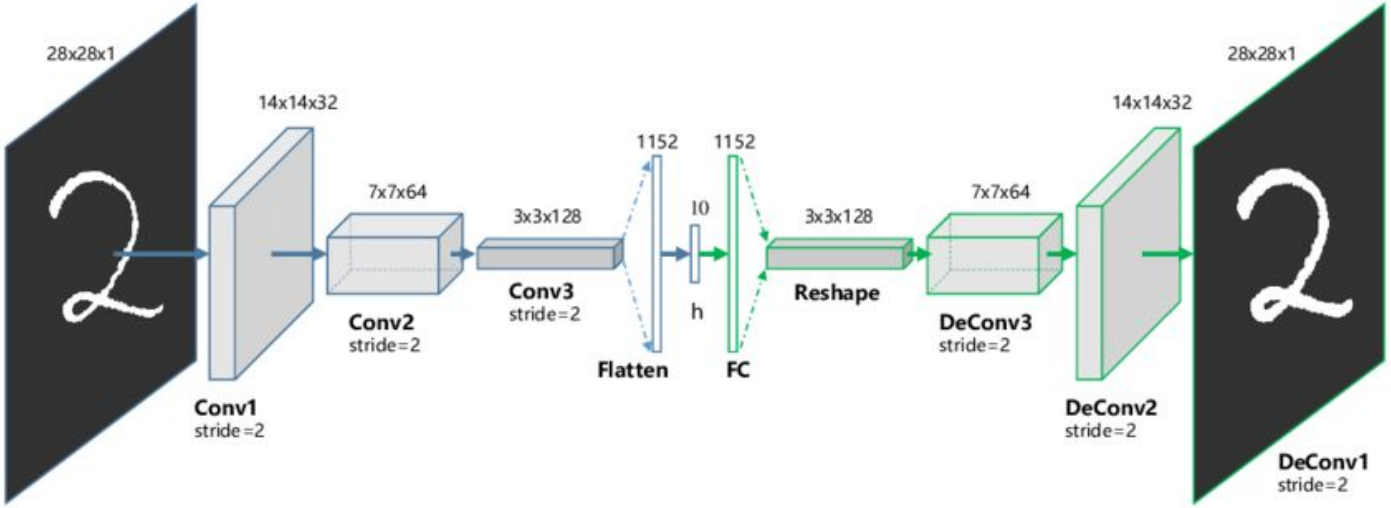
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Segmentation
Deconv

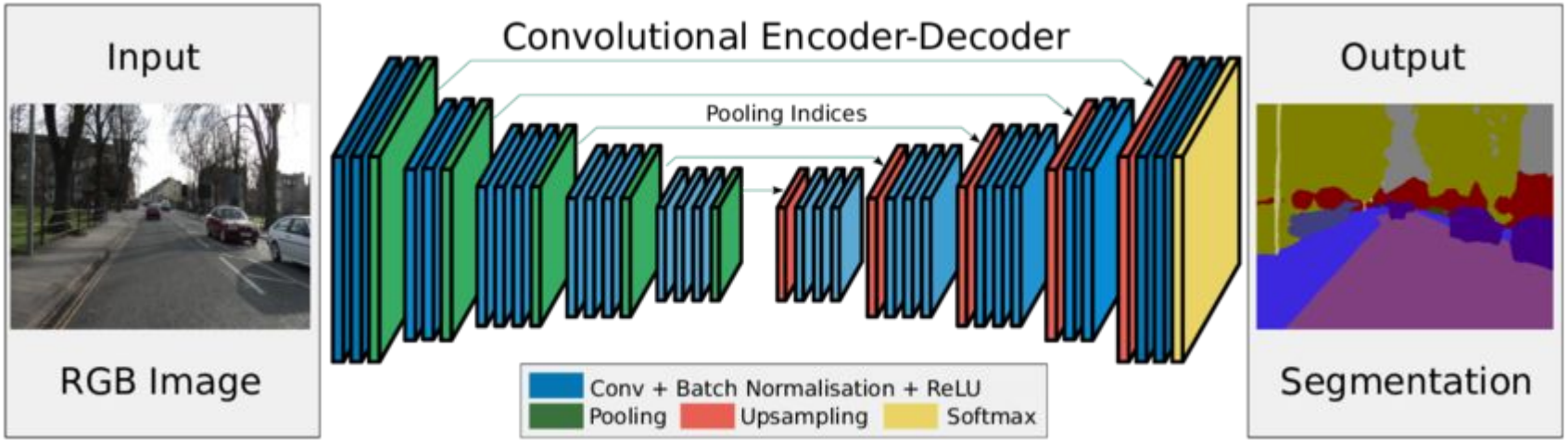




Segmentation Autoencoders



Segmentation



Thanks.
Questions?

Day 4