
DSTA
Defence Science &
Technology Agency
brainhack
BrainHack 2021

V1: Introduction to Colab

Agenda

- Familiarise with Google Colaboratory for Python development
- Familiarise with Notebook Shortcuts
- Sample codes

V1: Introduction to Colab

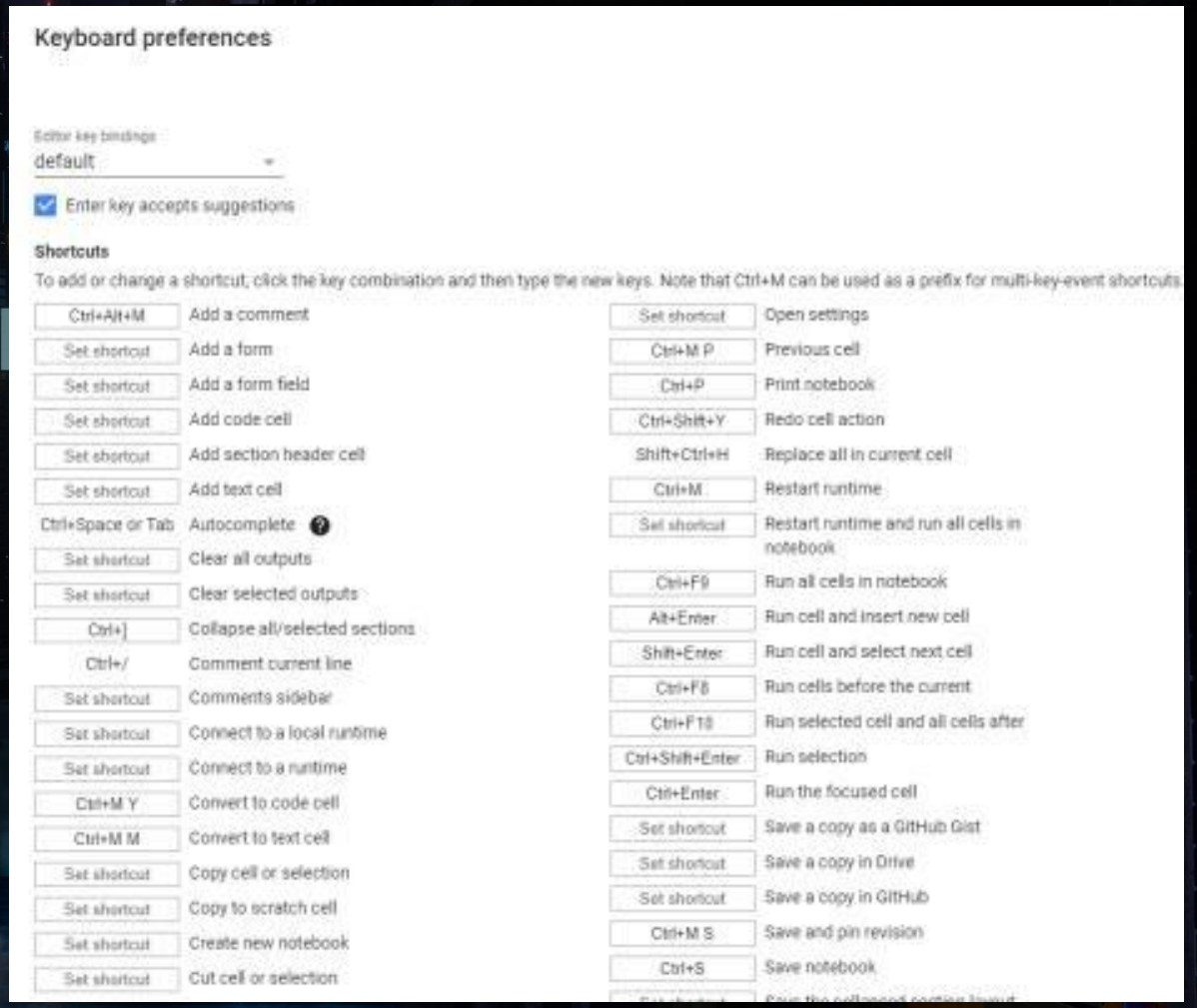
Colab

- Use Colab with a GPU runtime to speed up operations. *Runtime > Change runtime type > GPU.*
- **GPU** (Graphics Processing Unit) is a specialized processor with dedicated memory that performs extensive graphical and mathematical computations. Frees up CPU for other jobs.
- Tensor Processing Unit (TPU) is an AI accelerator application-specific integrated circuit (ASIC) specifically for neural network machine learning, particularly on TensorFlow.

V1: Introduction to Colab

Favourite Shortcuts

- Crtl + M Y → Convert to code cell
- Crtl + M M → Convert to text cell
- Crtl + M → Restart runtime
- Crtl + F9 → Run all cells
- Crtl + Shift + Y → Redo cell action
- Crtl + Shift + P → Show command palette
- Tab → Show suggestion
- Hover → See the inputs to function



V2: Introduction to PyTorch I

Agenda

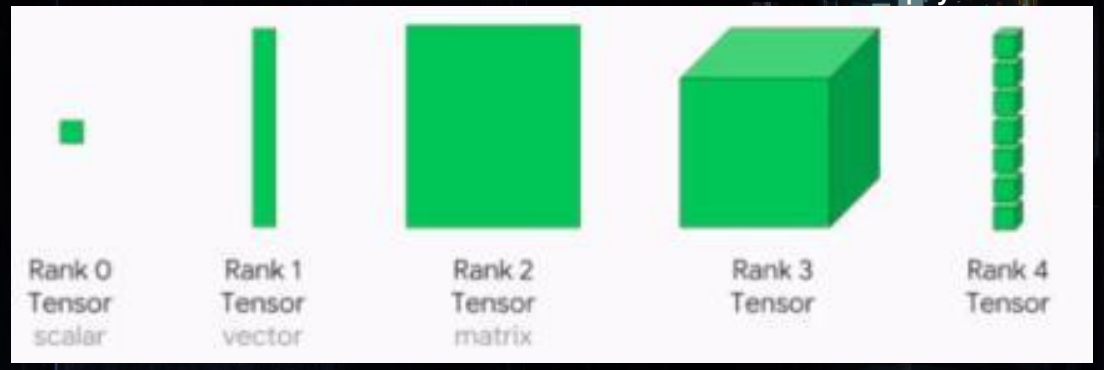
- Familiarise with PyTorch
- Familiarise with PyTorch Tensors (Data Types)
- Sample codes

V2: Introduction to PyTorch I

PyTorch

At its core, PyTorch provides a few main features:

- An n-dimensional Tensor, similar to numpy but can run on GPUs



- Automatic differentiation for building and training neural networks
  - f(x) {…} → df(x) {…}

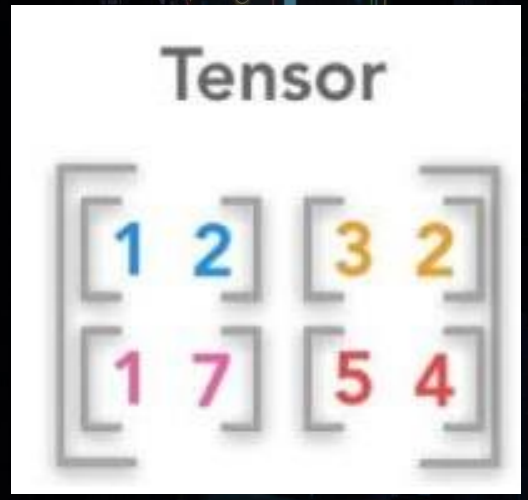
Domain-specific libraries such as TorchText, TorchVision, and TorchAudio, all of which include datasets.

Source: *Patidar, 2019 from Medium.com*

V2: Introduction to PyTorch I

Tensors

- In PyTorch, we use tensors to encode the inputs and outputs of a model, as well as the model's parameters.
- `x_data = torch.tensor(x)`



- Over 100 tensor operations, including arithmetic, linear algebra, matrix manipulation (transposing, indexing, slicing), sampling and more are comprehensively described at https://PyTorch.org/docs/stable/torch.html

Source: *Matthew, 2019 from KDnuggets.com*

V3: Introduction to PyTorch II

Agenda

- Familiarise with PyTorch Datasets & DataLoaders (Data Structures)
- Familiarise with PyTorch Data Transformation
- Sample codes

Custom Dataset

- A custom Dataset class must implement three functions: `__init__`, `__len__`, and `__getitem__`.

Initialize the dataset parameters

```
class CustomImageDataset Dataset:
    def __init__(self, annotations_file, img_dir, transform=None,
    target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform
```

```
tshirt1.jpg, 0
tshirt2.jpg, 0
......
ankleboot999.jpg, 9
```

DSTA
Defence Science & Technology Agency

V3: Introduction to PyTorch II

brainhack

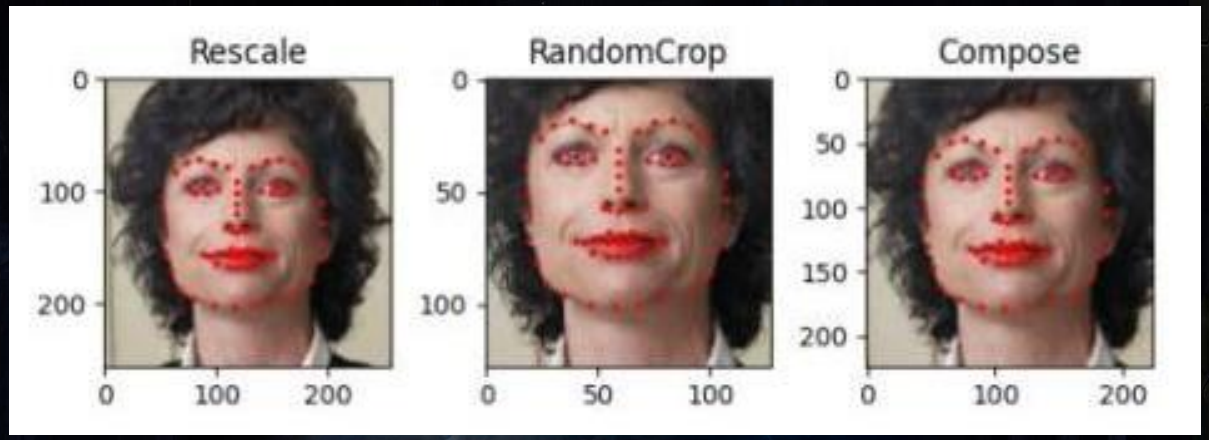
Transformation

- **`transforms`** manipulates the data and make it suitable for training.
- Data does not always come in its final processed form that is required for training machine learning algorithms.
- All TorchVision datasets have two parameters ``**`transform`**`` to modify the features and ``**`target_transform`**`` to modify the labels.
- The **`torchvision.transforms`** module offers several commonly-used transforms out of the box.

V3: Introduction to PyTorch II

Custom Dataset

- Read the raw data
- Transform the raw data features to the required and proper format to input to the model
- Multiple transformations can be done on the images in sequence
- `composed = transforms.Compose([Rescale(256), RandomCrop(224)])`



Source: *PyTorch Torchvision.Transforms*

V4: Arrays Manipulation I

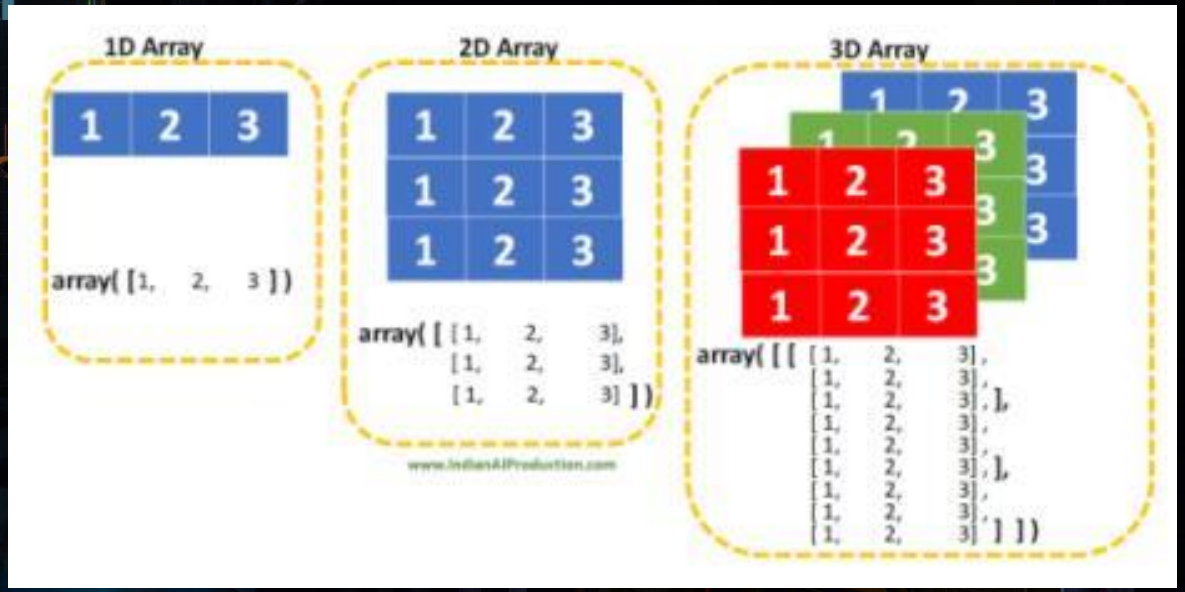
Agenda

- Familiarise with arrays
- Familiarise with array indexing, slicing, datatypes
- Sample codes

V4: Arrays Manipulation I

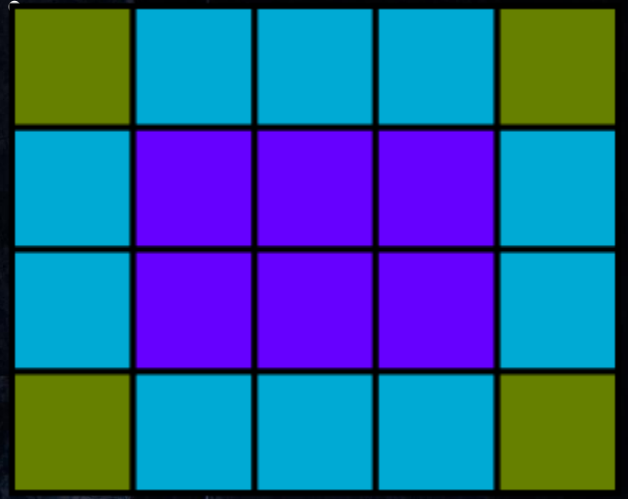
Image Array

- NumPy array is much faster in computational operations
- it is **multidimensional**
- it has a **fixed size** defined upon creation
- Image array is 3 dimensional (R,G,B)

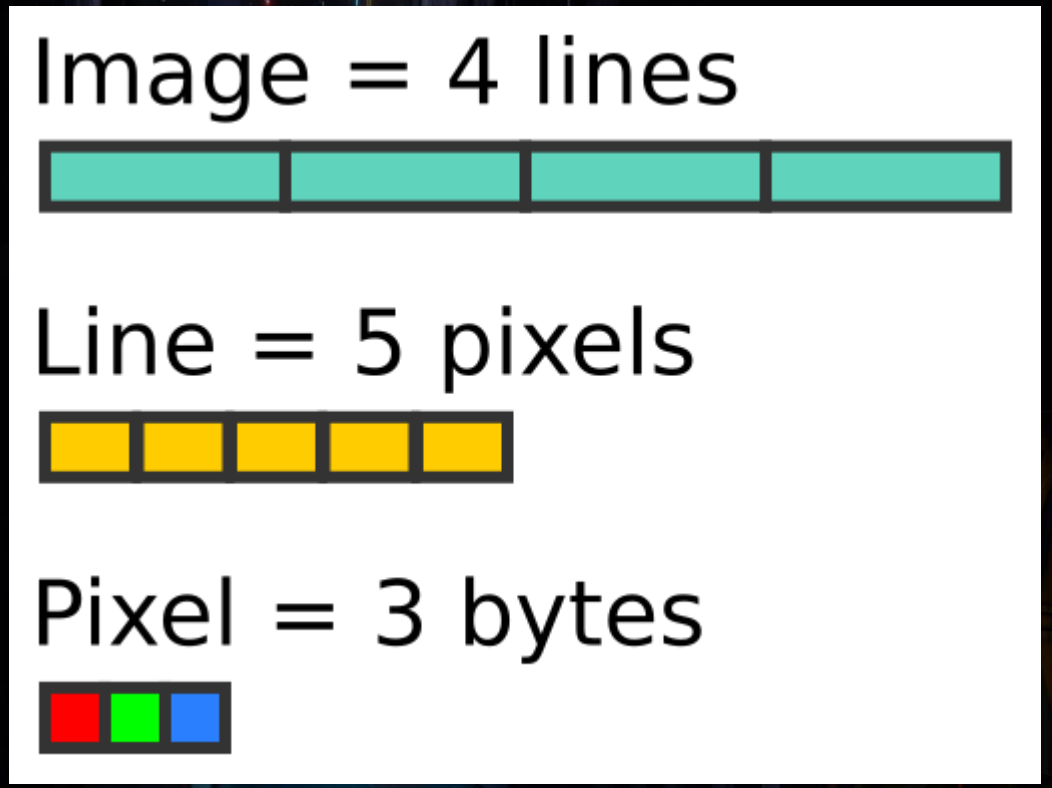


V4: Arrays Manipulation I

Image Array



4 x 5 RGB image



Height x width x 3

```
array = np.zeros([height, width, 3], dtype=np.uint8)
```

Source: *Martin, 2017, Python Informer*

V4: Arrays Manipulation I

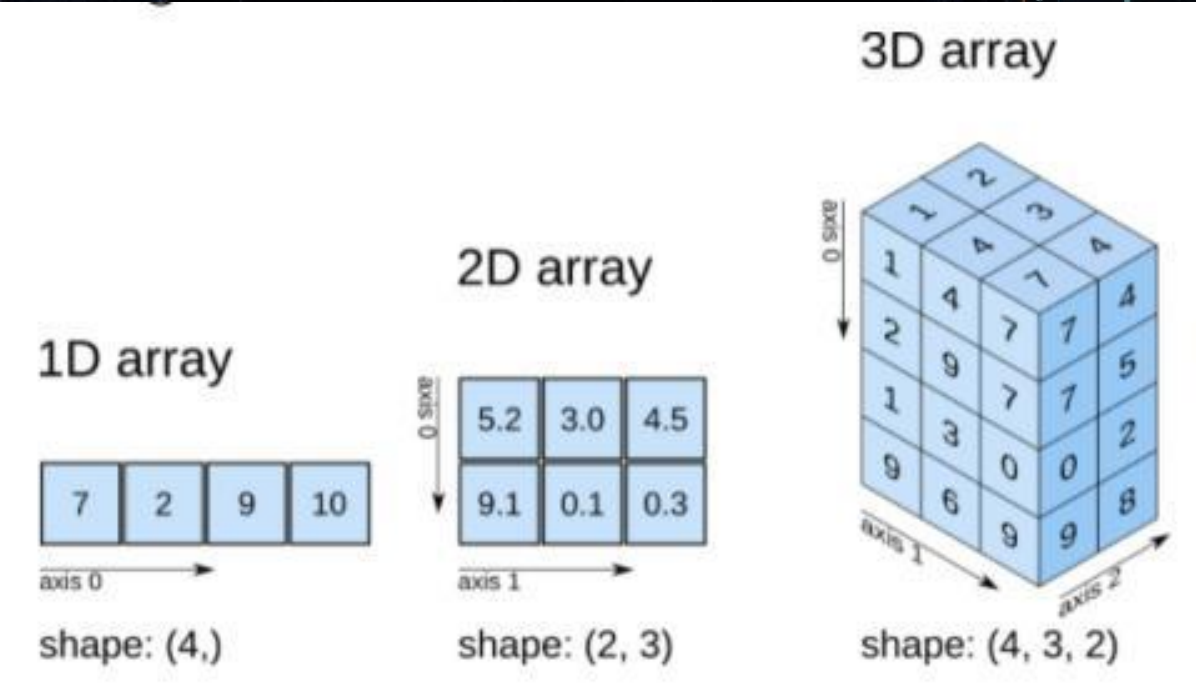
Image Array

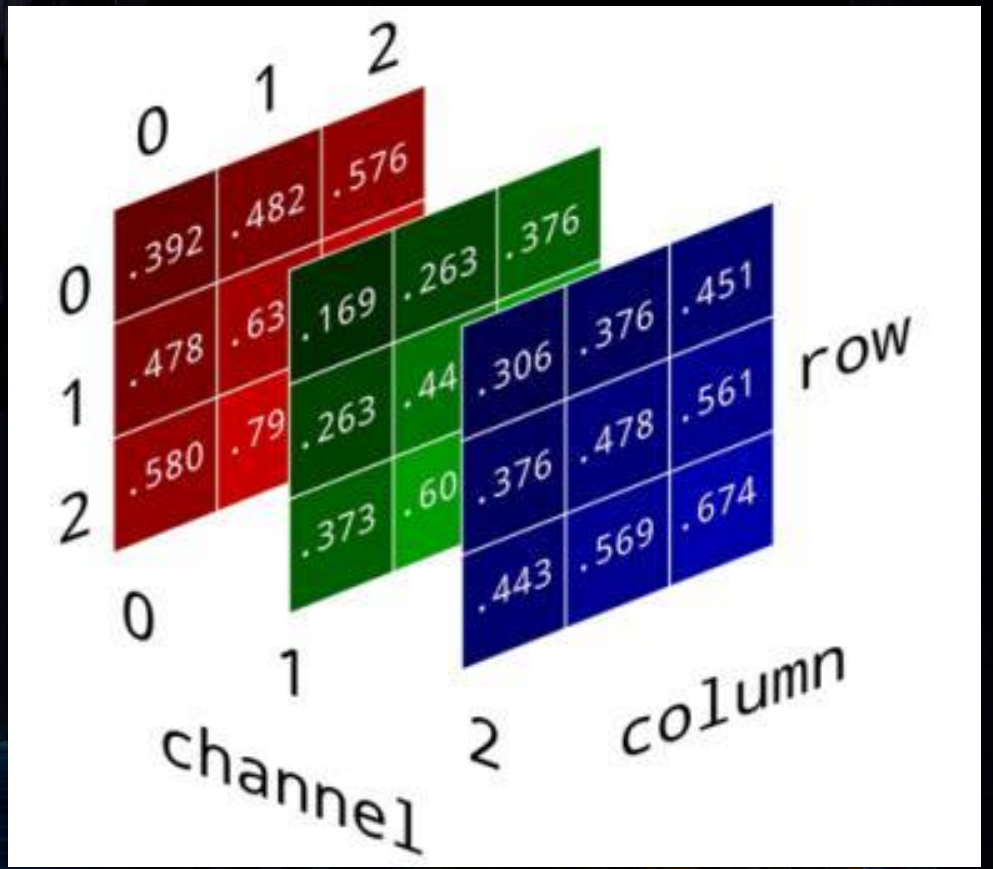
- Red (255, 0, 0)
- Green (0, 255, 0)
- Blue (0, 0, 255)
- White (255,255,255)
- Black (0,0,0)

https://www.w3schools.com/colors/colors_rgb.asp

V4: Arrays Manipulation I

Axes



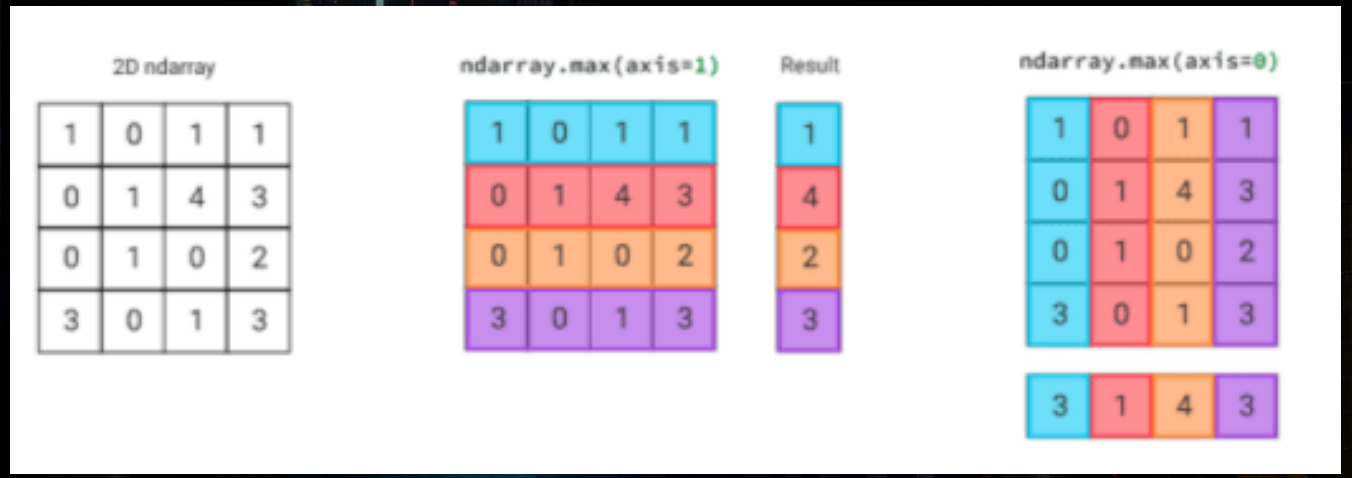


V4: Arrays Manipulation I

Axes

- 3D – axis(0,1,2)
- 4D – axis(0,1,2,3)
- nD – axis(0,1,2,3…n-1)

Perform transformations along the axes



V5: Arrays Manipulation II

Agenda

- Familiarise with array operations
- Familiarise with array broadcasting
- Sample codes

DSTA
Defence Science & Technology Agency

V5: Arrays Manipulation II

brainhack

Calculations using arrays are limited

```
a = [1, 2, 3]
b = [1, 2, 3]
c = a + b
c = [1 + 1, 2 + 2, 3 + 3]
```

same dimensions and sizes

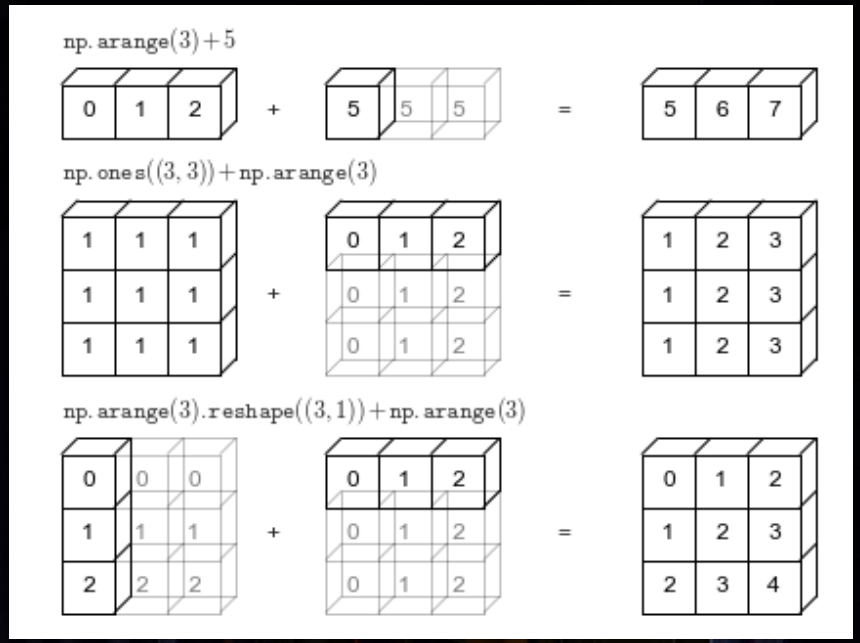
Vector product

- (Only n*m matrixes can cross-multiple with m*k) – 3x2 cross-multiply 2x4 will work
- But not 2x2 and 3x4

V5: Arrays Manipulation II

Broadcasting

- Broadcasting is a powerful mechanism that allows NumPy to work with arrays of different shapes
- Frequently a larger array needs to do operation with a smaller array multiple times.
- Smaller array "broadcast" across larger array (there's still a requirement to the shape)
- *Ci,j = Ai,j + bj. In other words, the vector b is added to each row of the matrix.*



Source: Python, Data Science Handbook

V6: Introduction to Deep Learning (DL)

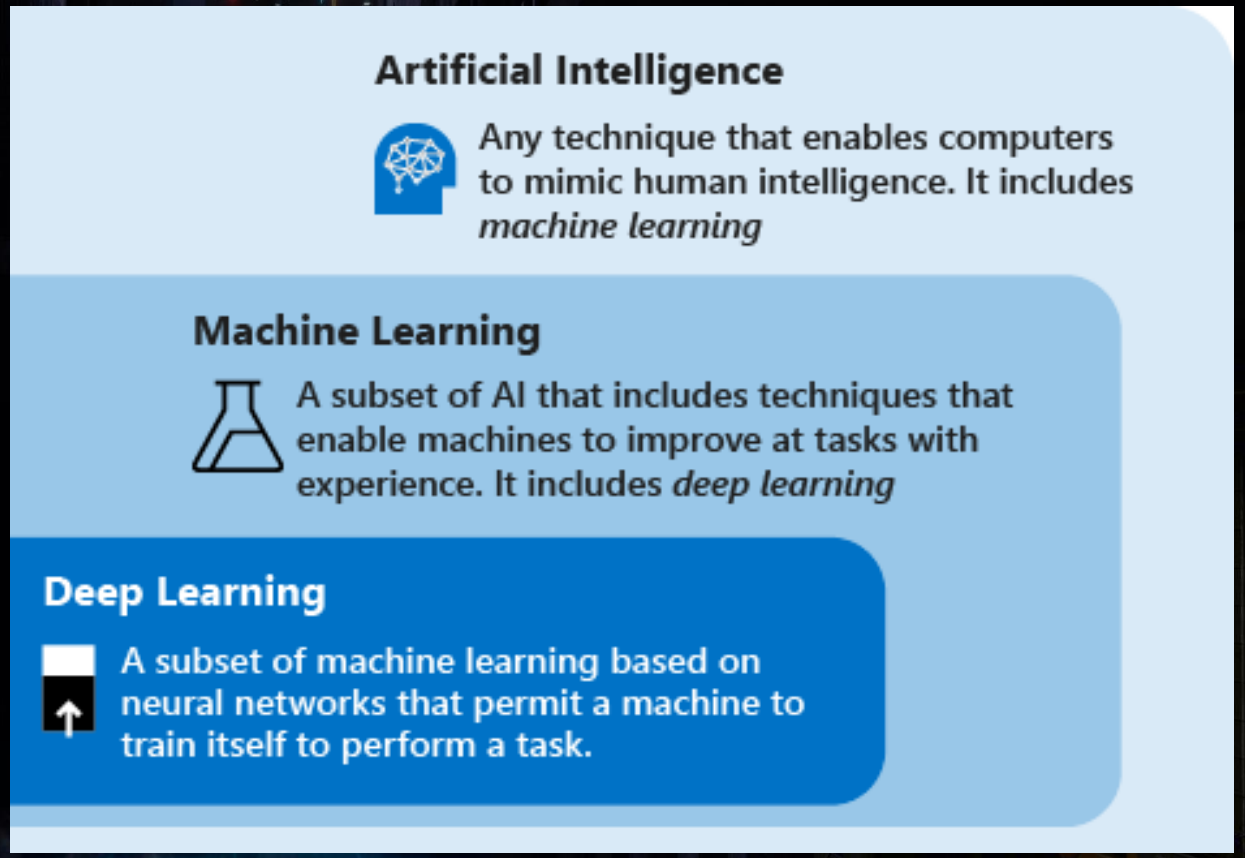
Agenda

- Appreciate difference between Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL)
- Appreciate DL
- Appreciate DL Use Cases

V6: Introduction to Deep Learning (DL)

AI, ML, DL

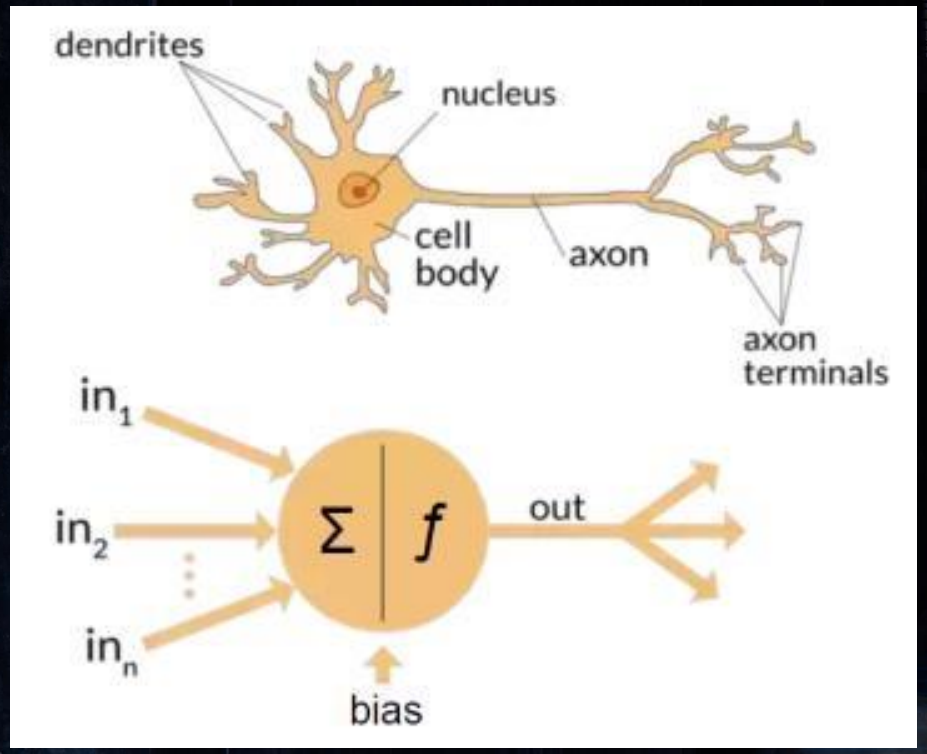
- AI – Systems
- ML – Algorithms
- DL – Neural Networks(NN)



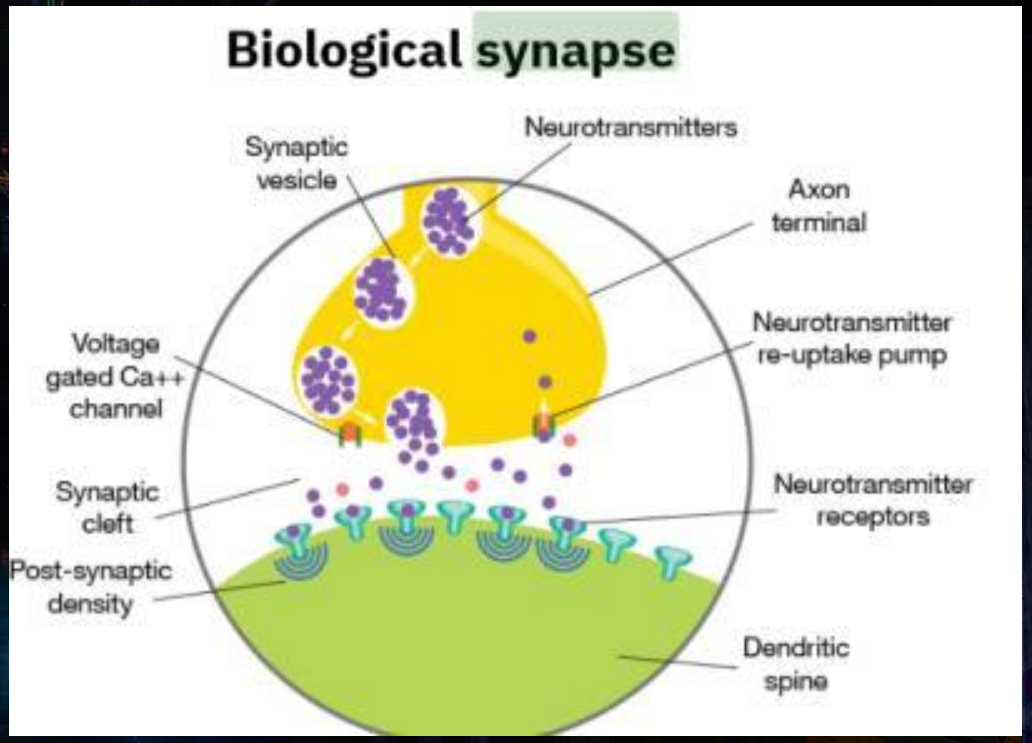
Source: *Microsoft*

V6: Introduction to Deep Learning (DL)

DL



Activation Function

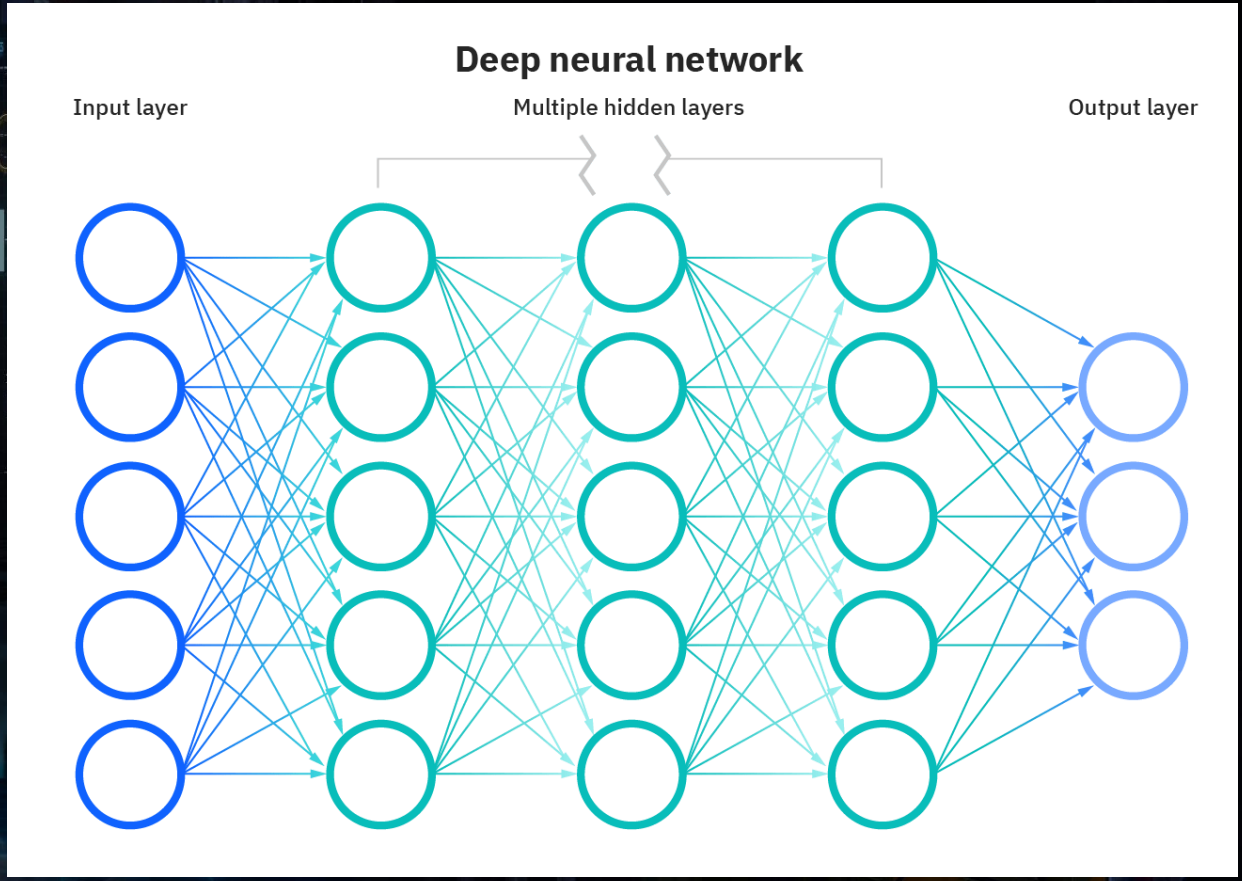


Source: *IBM*

V6: Introduction to Deep Learning (DL)

NN

- Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns.
- The networks are built from neurons.



Source: *IBM*

V6: Introduction to Deep Learning (DL)

NN

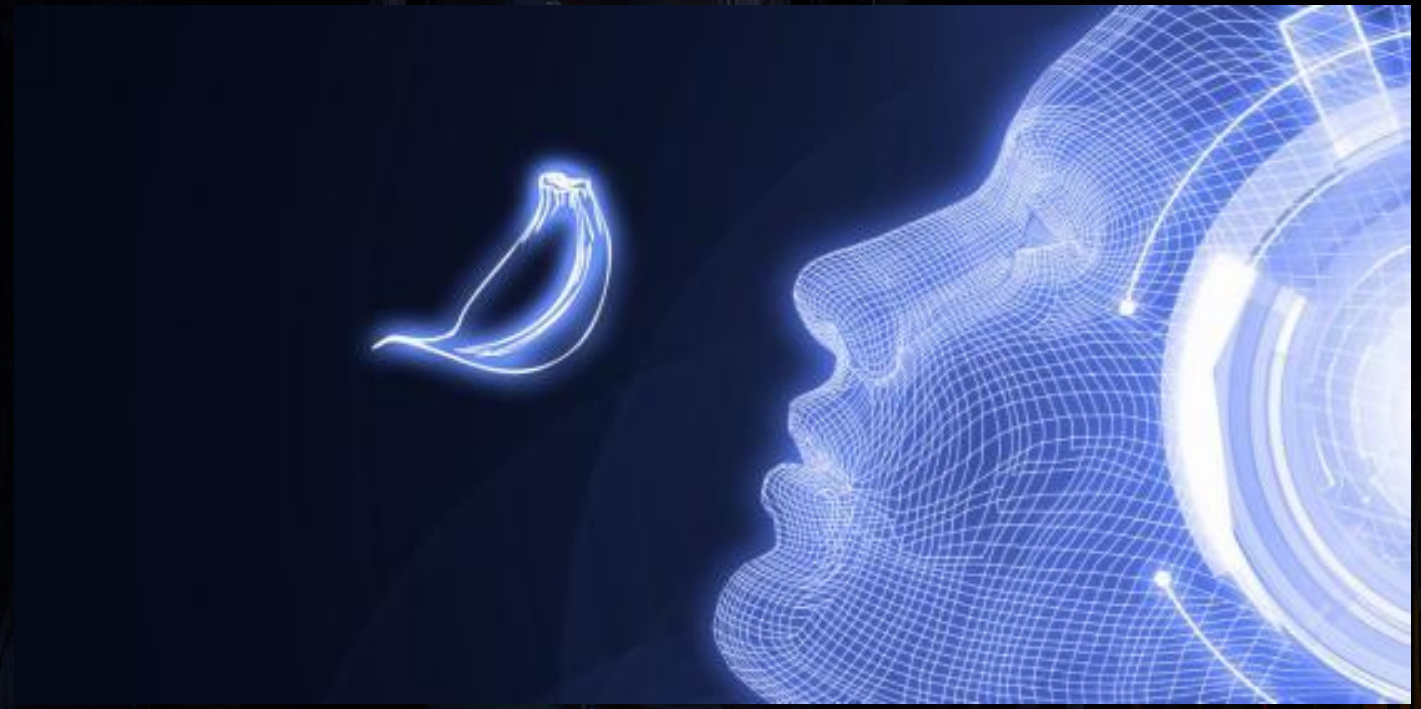
Types of Nodes in a Neural Network:

1. Input Layer
    - Input data do not perform any computation, just pass on the information to the hidden nodes.
2. Hidden Layer
    - These nodes do not have any direct connection with the software, program etc. [Black Box]
    - They perform computations and transfer information from Input nodes to Output nodes.
    - 0 to multiple Hidden Layers
3. Output Layer
    - Compute and transform data to actions for software, program etc.

Each layer comprises one or more nodes.

V6: Introduction to Deep Learning (DL)

Smell detection



Source: *Amazon Web Services*

V6: Introduction to Deep Learning (DL)

DeepComposer

Source: *Aravin, 2020, Analytics Vidhya*

V6: Introduction to Deep Learning (DL)

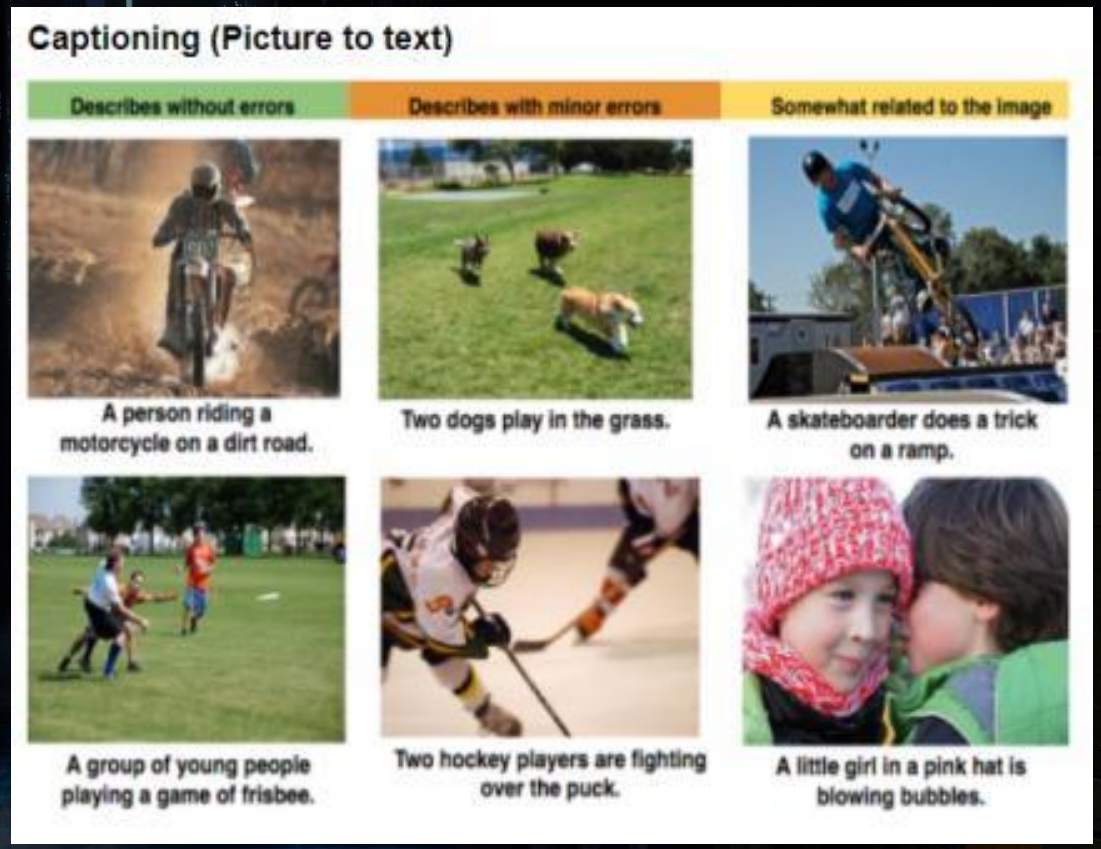
DeepFake

Source: *Amazon Web Services*

V6: Introduction to Deep Learning (DL)

Optical Character Recognition (OCR)

Captioning (Picture to text)



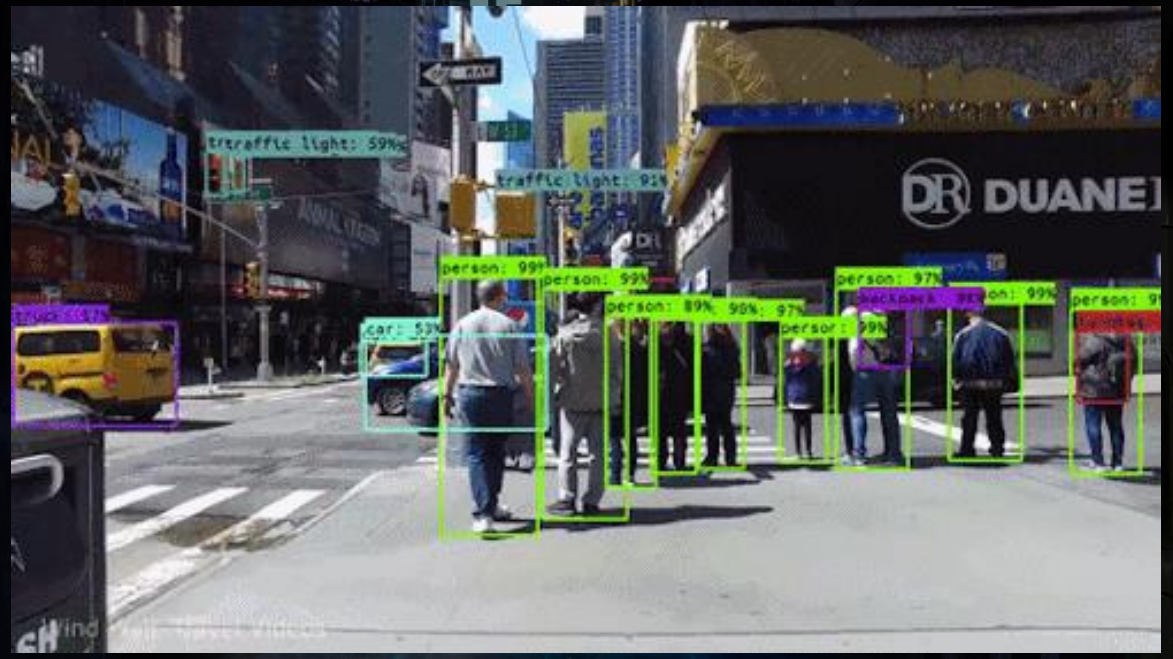
V6: Introduction to Deep Learning (DL)

Pose Estimation

V6: Introduction to Deep Learning (DL)

Image detection & Image recognition (Autonomous Driving)

Autonomous cars, violence detection in public transportation, phone unlocking, person tracking, gesture detection, ...



V7: Hidden Layers and Neurons

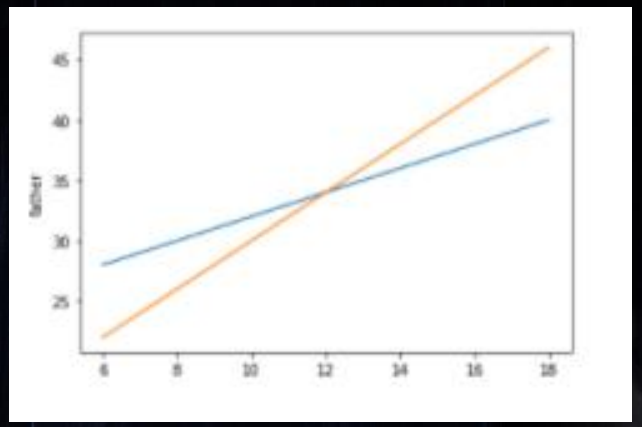
Agenda

- Appreciate neural network architecture
- Appreciate optimal architecture best practices
- Sample codes

V7: Hidden Layers and Neurons

Recap of Simultaneous Equations

- A father is 22 years older than his son
- In 10 years, the father will be twice as old as his son.
- Let Y be the father's age, X be the son's age



- `Y= X + 22`
- `Y + 10 = 2(X+10)`

X = 12
Y = 34

Source: *IBM*

V7: Hidden Layers and Neurons

Recap of Simultaneous Equations

- What if you have more?

$$\begin{cases} 3x + 2y - z = 1 \\ 2x - 2y + 4z = -2 \\ -x + \frac{1}{2}y - z = 0 \end{cases}$$

- `x = 1, y = -2, z = -2`

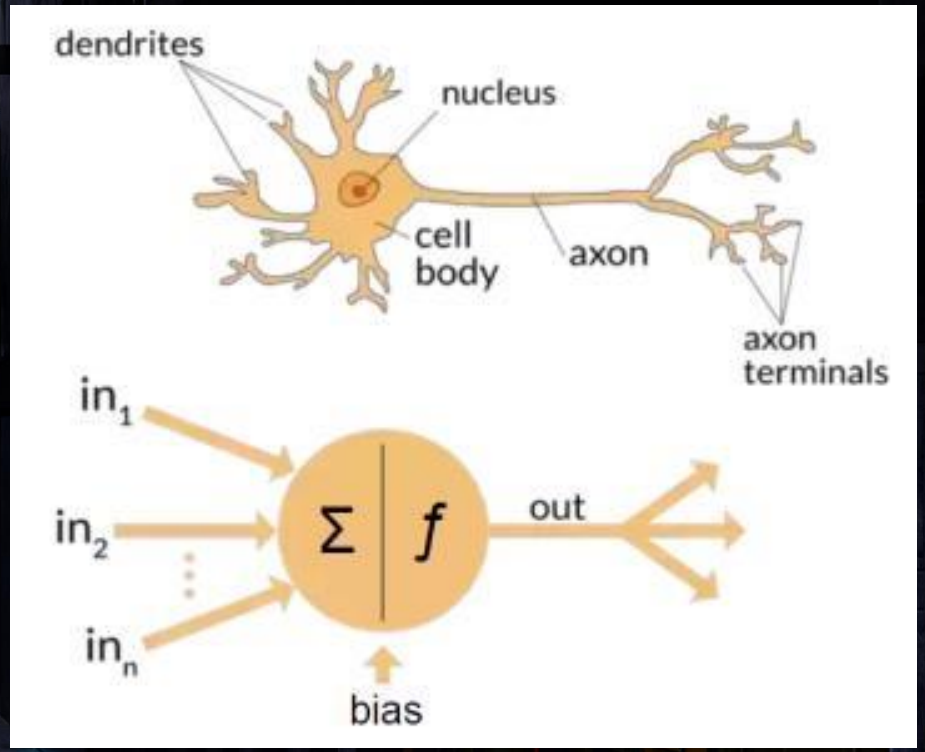
- If weights are x, y and z, weights can be solved

Source: *IBM*

V7: Hidden Layers and Neurons

NN neuron

- Each neuron has some number of weighted inputs.

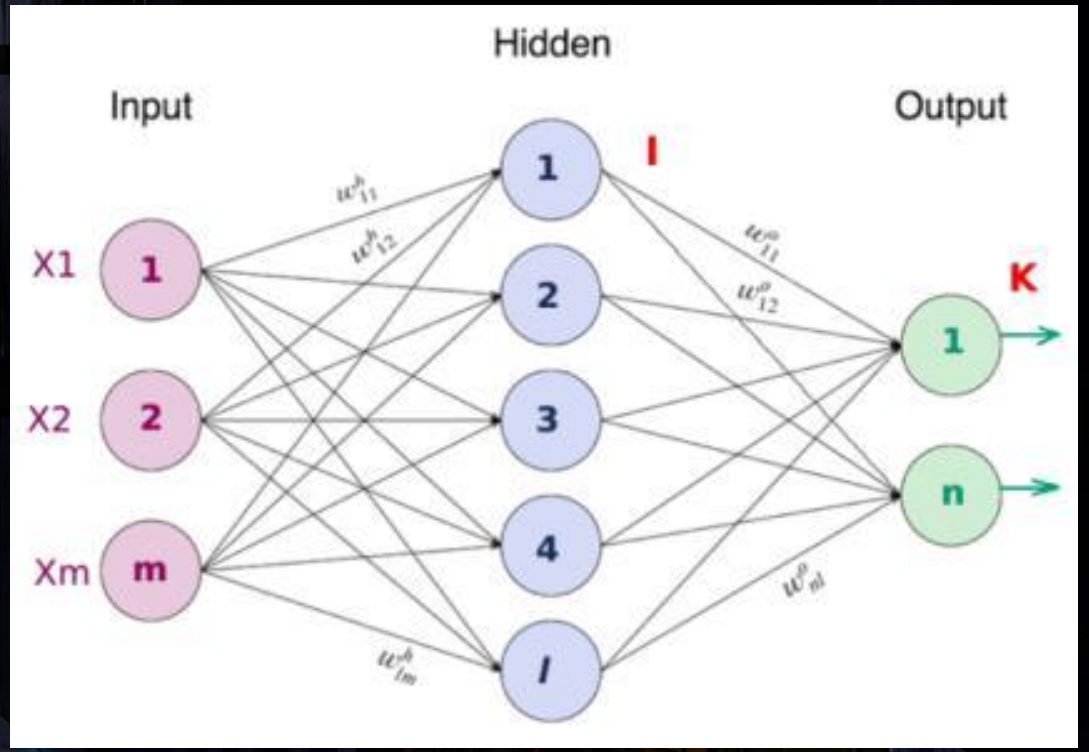


Source: *IBM*

V7: Hidden Layers and Neurons

NN network

- Each neuron has some number of weighted inputs.
- These weighted inputs are summed together (a linear combination)
- Then passed through an activation function to get the unit's output.
- Feed forward NN



Source: *Ahmed, 2020, Medium.com*

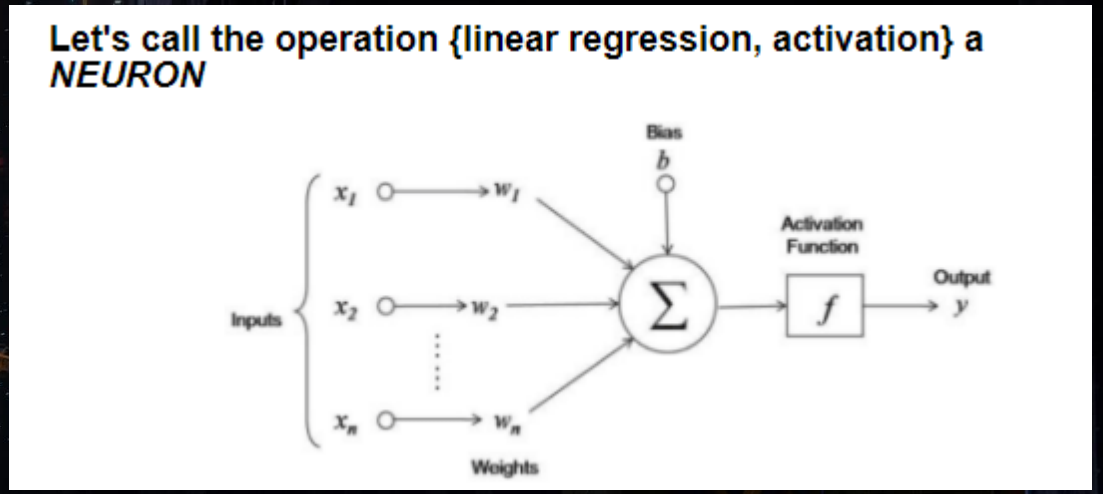
DSTA
Defence Science & Technology Agency

V7: Hidden Layers and Neurons

brainhack

NN network linear operations

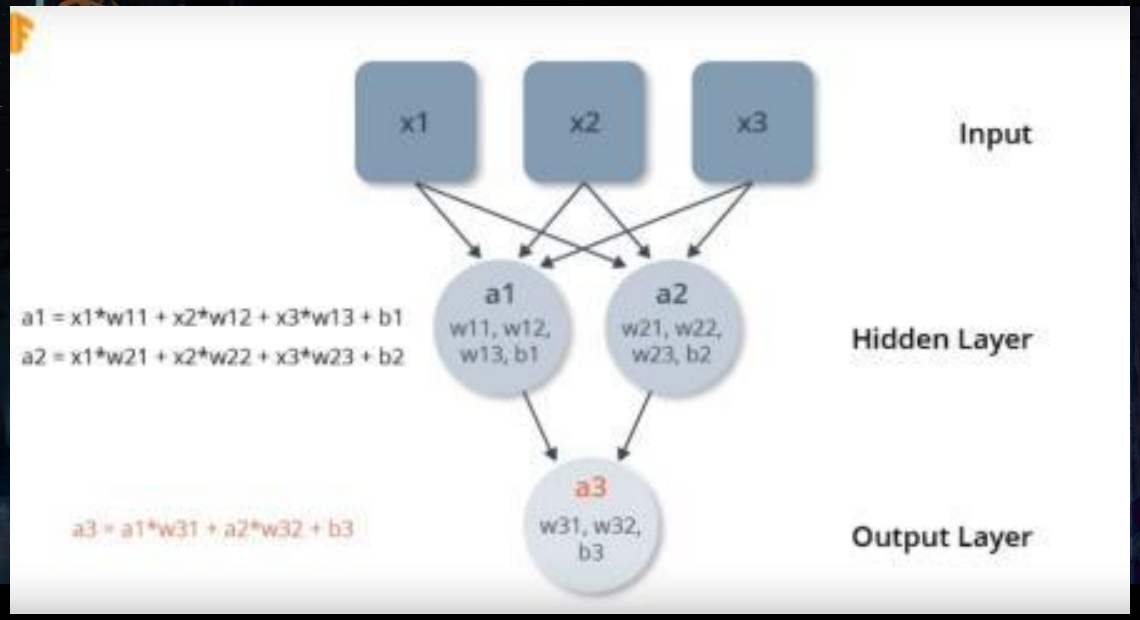
- Data input is entering to a neuron
- `X = {x1,x2,…}`
- `Weights = {w1, w2, …}`
- `Bias = b` (default = 0)
- Regression Outputs = `w1*x1+w2*x2+…+b`
- Activation Function transform the output = `sin(w1*x1+w2*x2+…+b)`



V7: Hidden Layers and Neurons

NN network nonlinear operations

- Expand from 1 neuron computation to multiple computations and a FCNN would be achieved
- Multiple neurons in a dense layer increase dimensional complexity the problems can solve
- Non-linear problems can be solved with non-linear activation functions e.g. Sigmoid



Source: *Tensorflow*

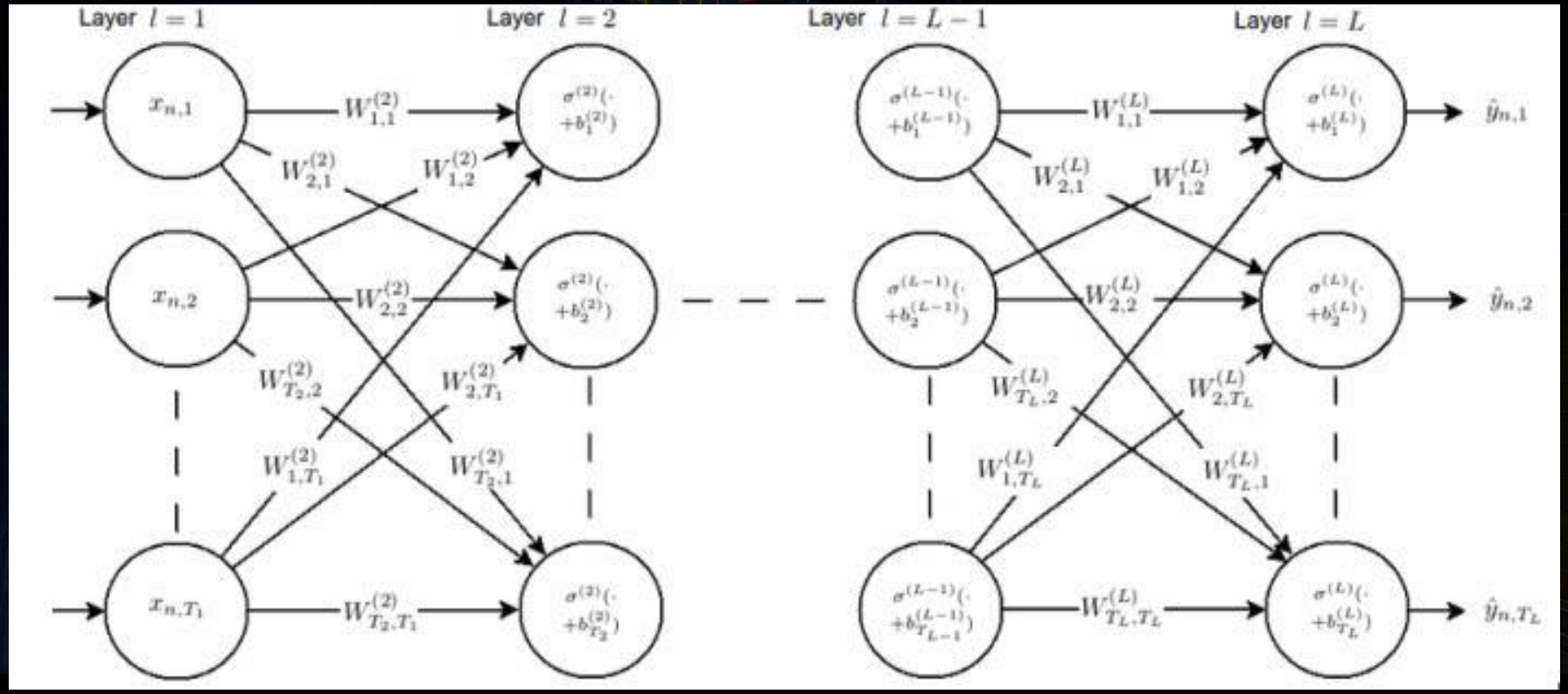
DSTA
Defence Science & Technology Agency

V7: Hidden Layers and Neurons

brainhack

NN

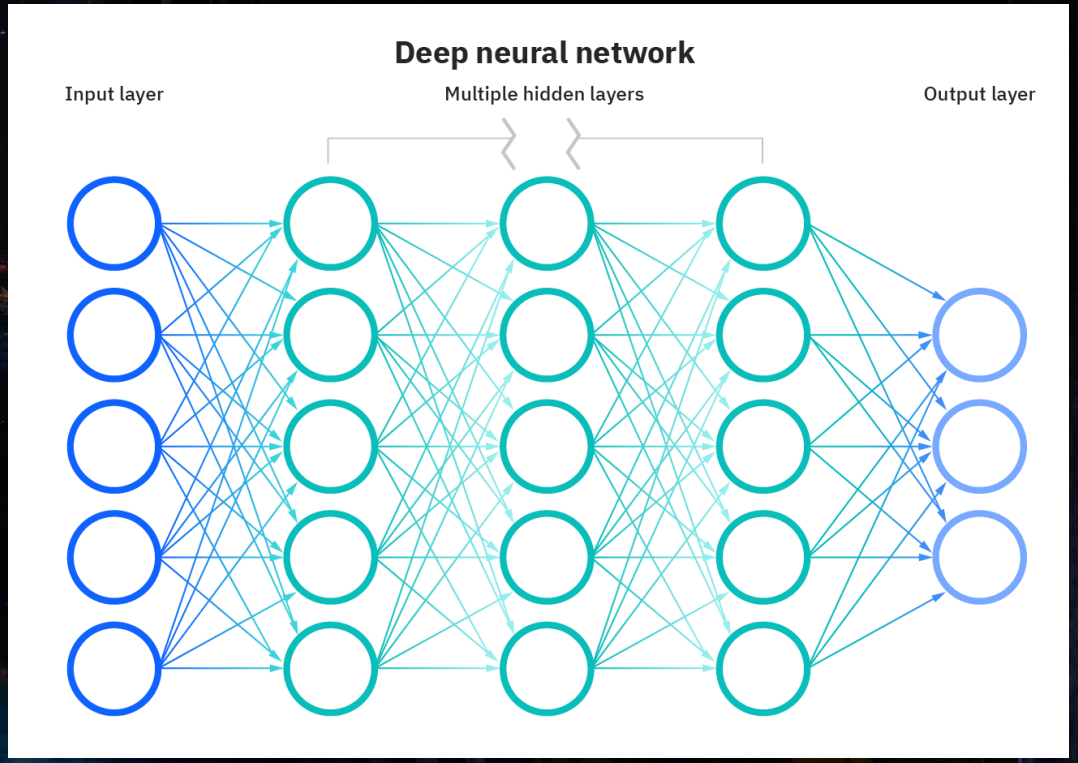
Dense [2] | Dense [L-1] | Dense [L]



V7: Hidden Layers and Neurons

NN

- Increase layers increase abstract representations of the data input to solve problems
- Increase neurons to increase the features extracted from data input



Source: *IBM*

V8: Activation and Optimizer

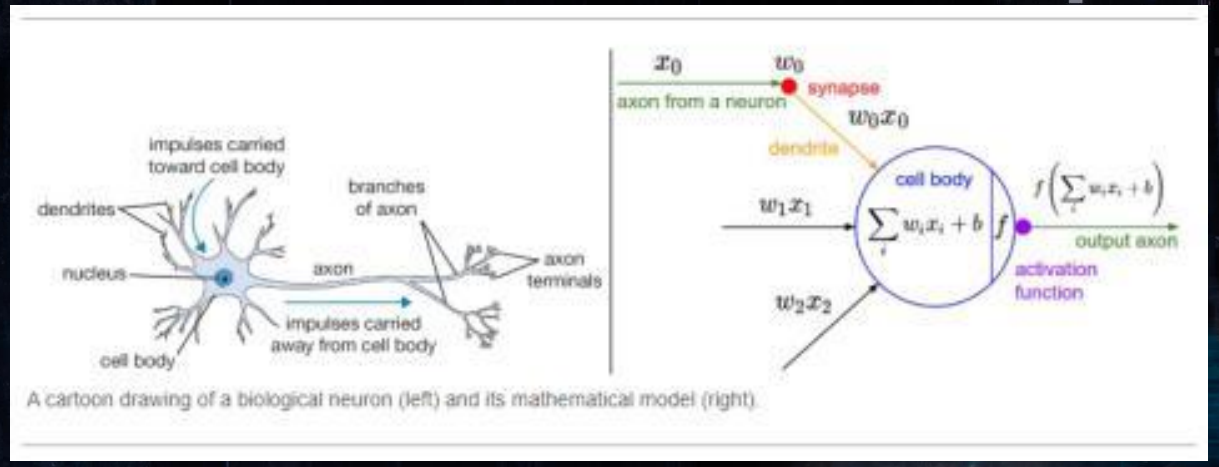
Agenda

- Appreciate the key parts of a neural network
- Appreciate the relationship between the key parts

V8: Activation and Optimizer

Activation Functions

- After the linear combinations, the output will be transformed by an activation function
- The final output will be decided by the activation function



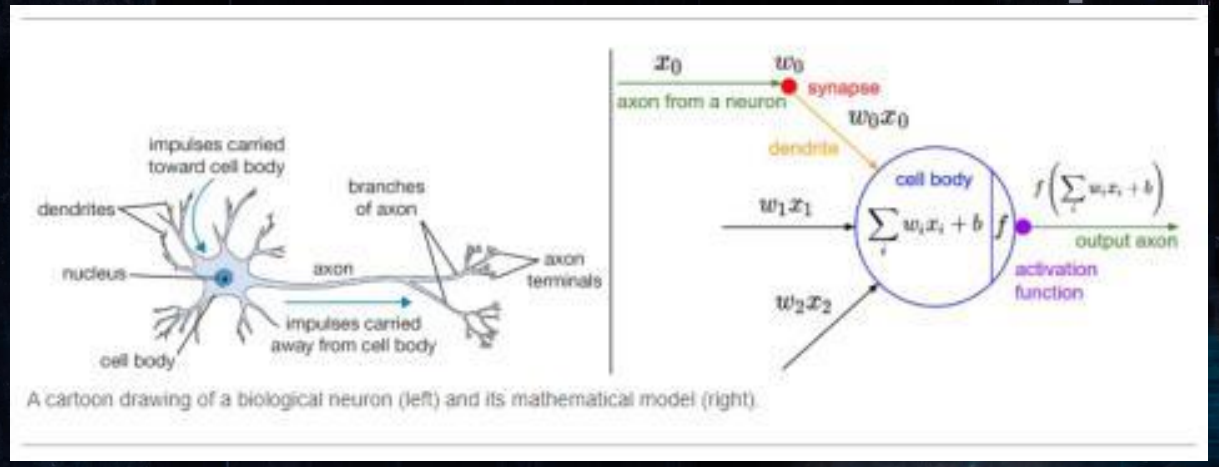
A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Source: *Stanford CS231n*

V8: Activation and Optimizer

Activation Functions

- After the linear combinations, the output will be transformed by an activation function
- The final output will be decided by the activation function



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

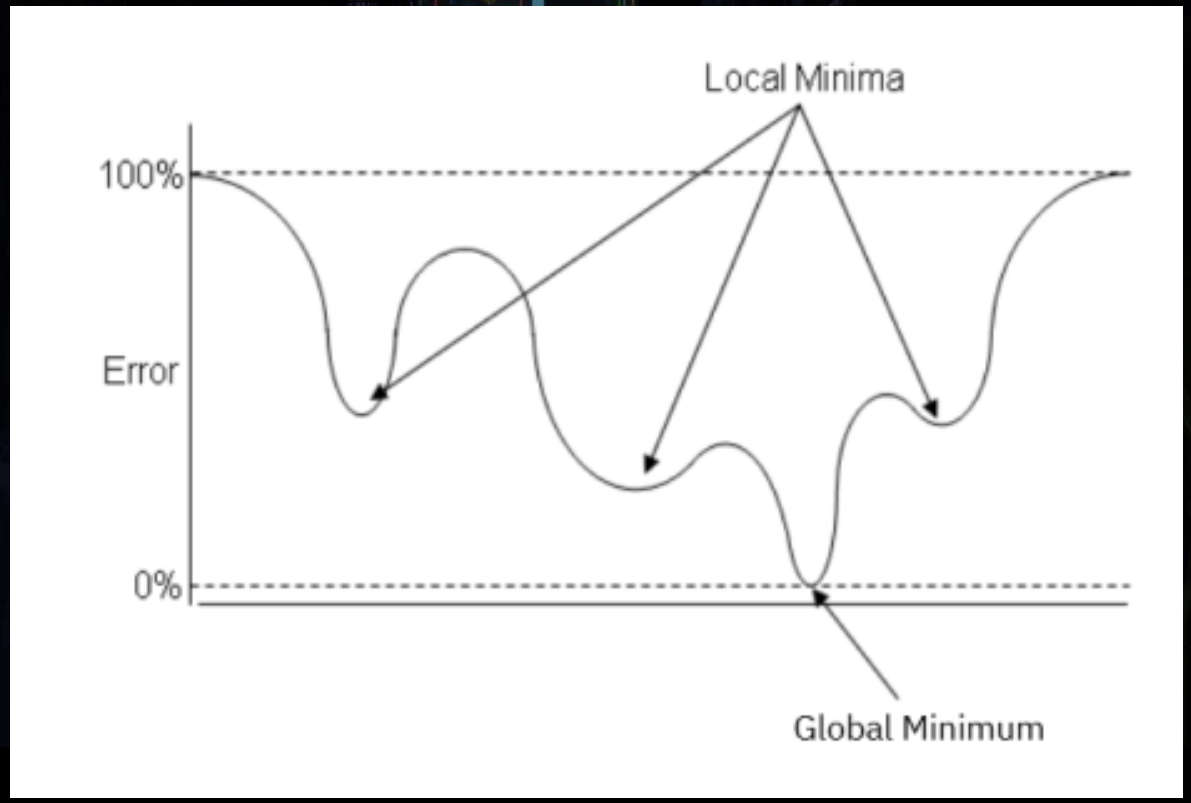
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Source: *Stanford CS231n*

V8: Activation and Optimizer

Gradient Descent

- Algorithm (formula) that minimizes the loss/error/cost function

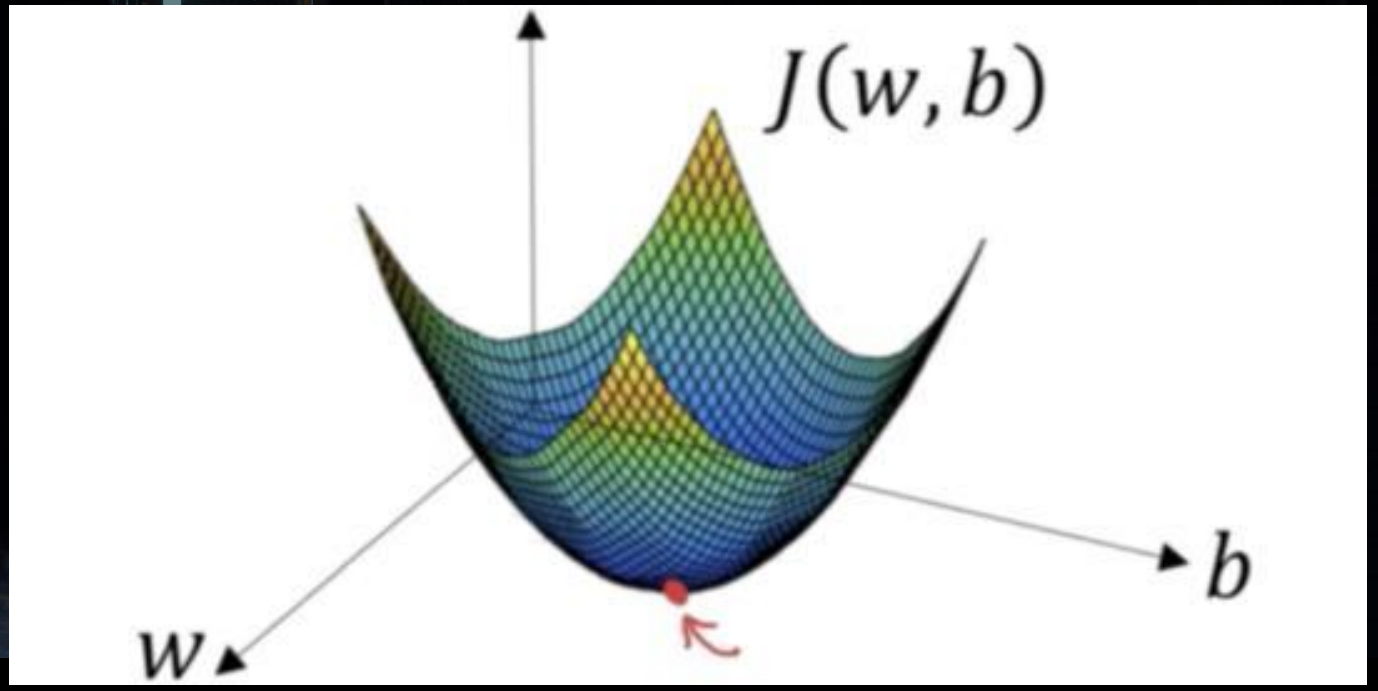


Source: *IBM*

V8: Activation and Optimizer

Gradient Descent

- Blind
  - No (or very little) memory of the space it has traversed
  - Get stuck in local minima
  - No guarantee of minimization

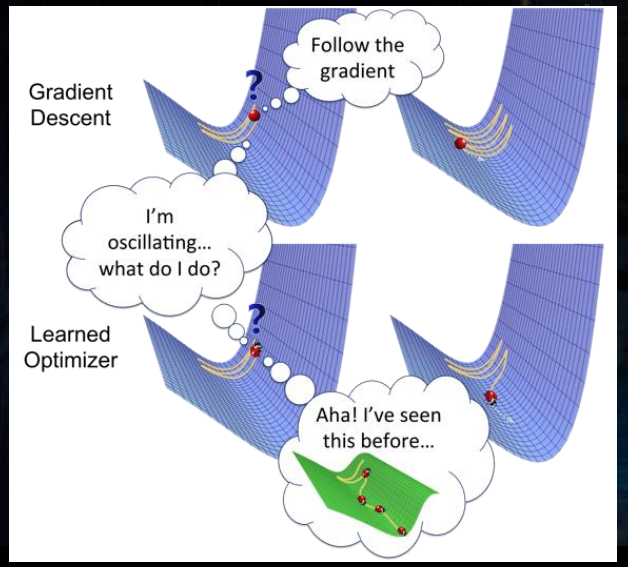


Source: *IBM*

V8: Activation and Optimizer

Optimizer

- As the model learns, the optimizer is a formula that changes the weights to reduce errors
- Similar to "solver" in excel, or the process of solving the simultaneous equations
- Optimizer = 'gradient descent', 'adam', 'stochastic gradient descent', 'adagrad'



Source: *Sanket, 2019, Medium.com*

V9: How Model Learns?

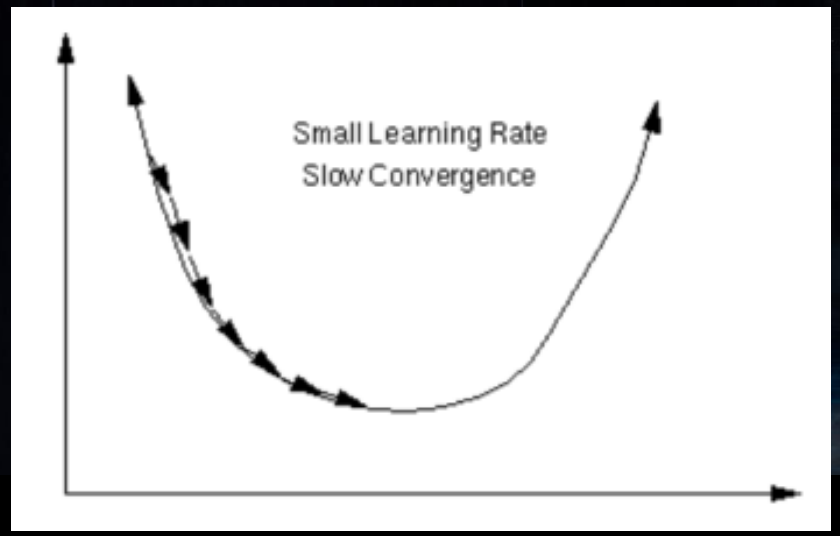
Agenda

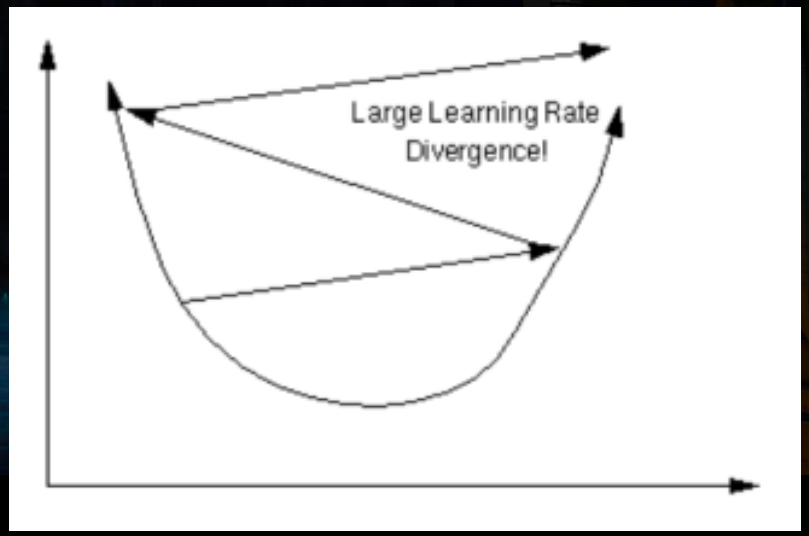
- Appreciate parameters such as learning rate
- Appreciate fitting for model to learn
- Sample codes

V9: How Model Learns?

Learning Rate

- The learning rate in gradient descent is basically how big of a “step” the algorithm takes.
- The appropriate rate depends on the data and solution space.
- But problems can arise if the rate is too large or too small.
- If the learning rate is too large, the algorithm might step over the global minimum.





Source: *IBM*

V9: How Model Learns?

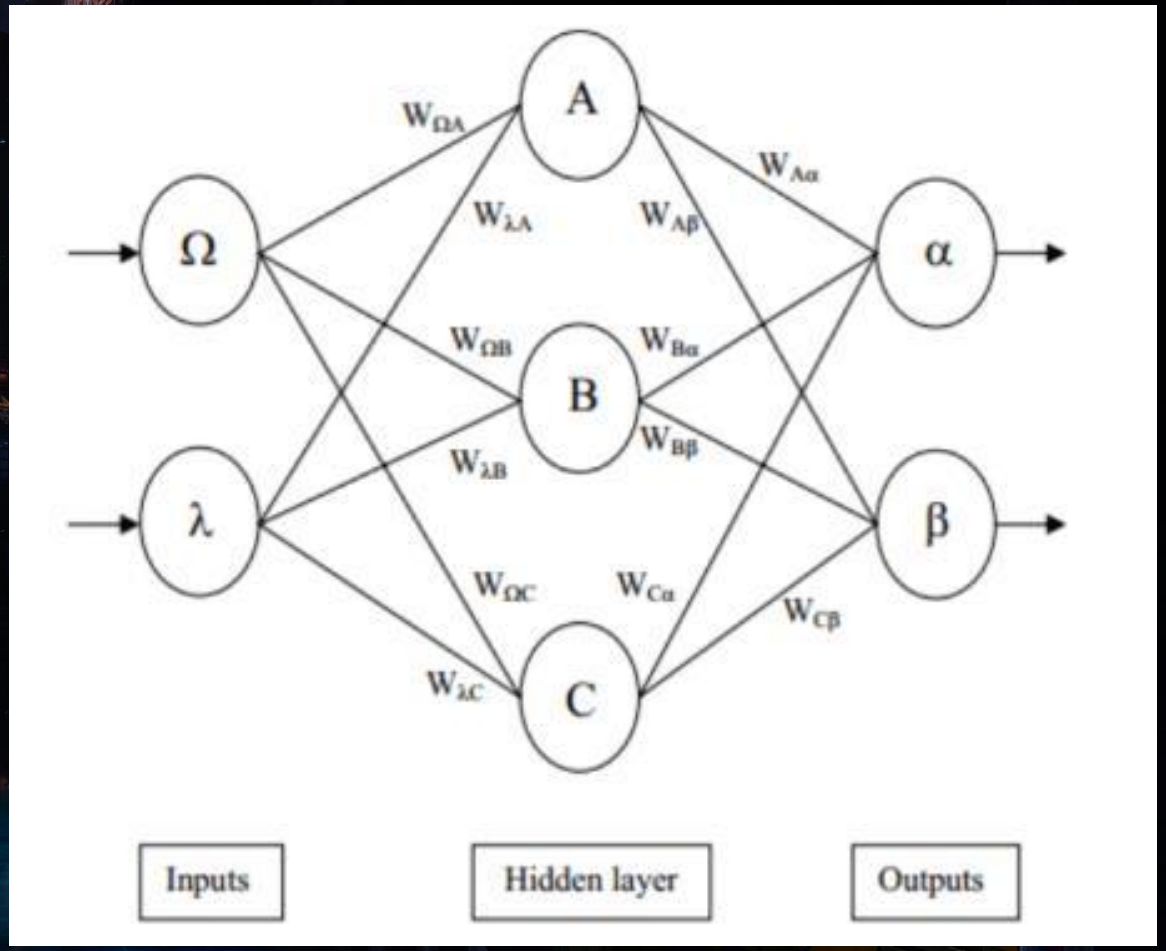
Fitting

- It tries to find the weight that minimizes the loss function.
- The learning phase is **iterative** and **stochastic**.
- The intuition is that you give a subset of data, and the algorithms updates the parameters

V9: How Model Learns?

Backpropagation

1. Calculate errors of output neurons
2. Change output layer weights
3. Calculate (backpropagate) hidden layer errors
4. Change hidden layer weights



Source: *IBM*

DSTA
Defence Science & Technology Agency

V10: Encoding and Normalization

brainhack

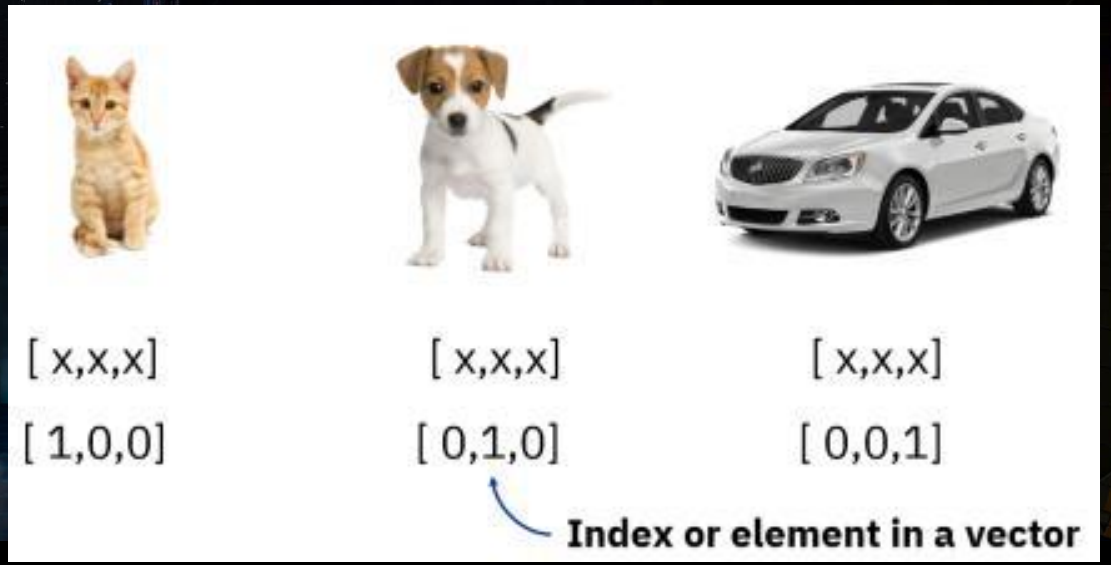
Agenda

- Appreciate data encoding
- Appreciate data normalization
- Sample codes

V10: Encoding and Normalization

One-Hot Encoding

- Create a matrix of 0's and 1's (DataSets and DataLoaders)
- Make each category a column in a table
- Binary classification will only have one output node at the end

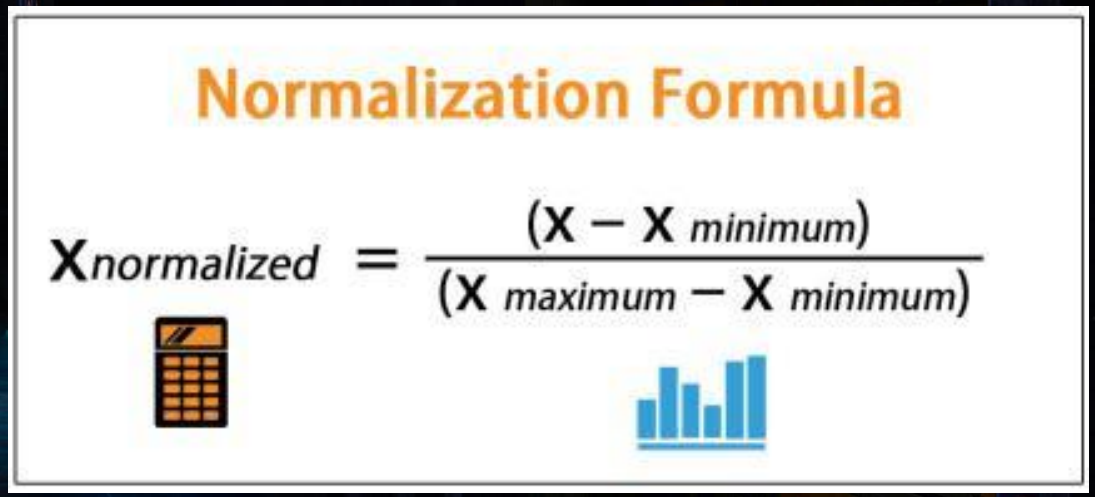


Source: *IBM*

V10: Encoding and Normalization

Data Normalization

- Gradient descent is scale sensitive so if there is one input field in the millions and one in the tens, the larger field will dominate. (Derivative of absolute larger values will be relatively larger)
- Images are often scaled to be between 0 and 1 (Minmax Scaling) to reduce the pixels into proportionate comparisons.



Source: *Madhuri, WallStreetMojo*

V11: Bringing it all together

Agenda

- Appreciate computer vision use cases
- Appreciate computer vision pipeline
- Appreciate computer vision limitations
- Appreciate computer vision developments

DSTA
Defence Science & Technology Agency

V11: Bringing it all together

brainhack

Neural Network Development

- Many moving parts
- All parts are interrelated
- Experimentation helps
- Computation limitation exists

DSTA
Defence Science & Technology Agency

V11: Bringing it all together

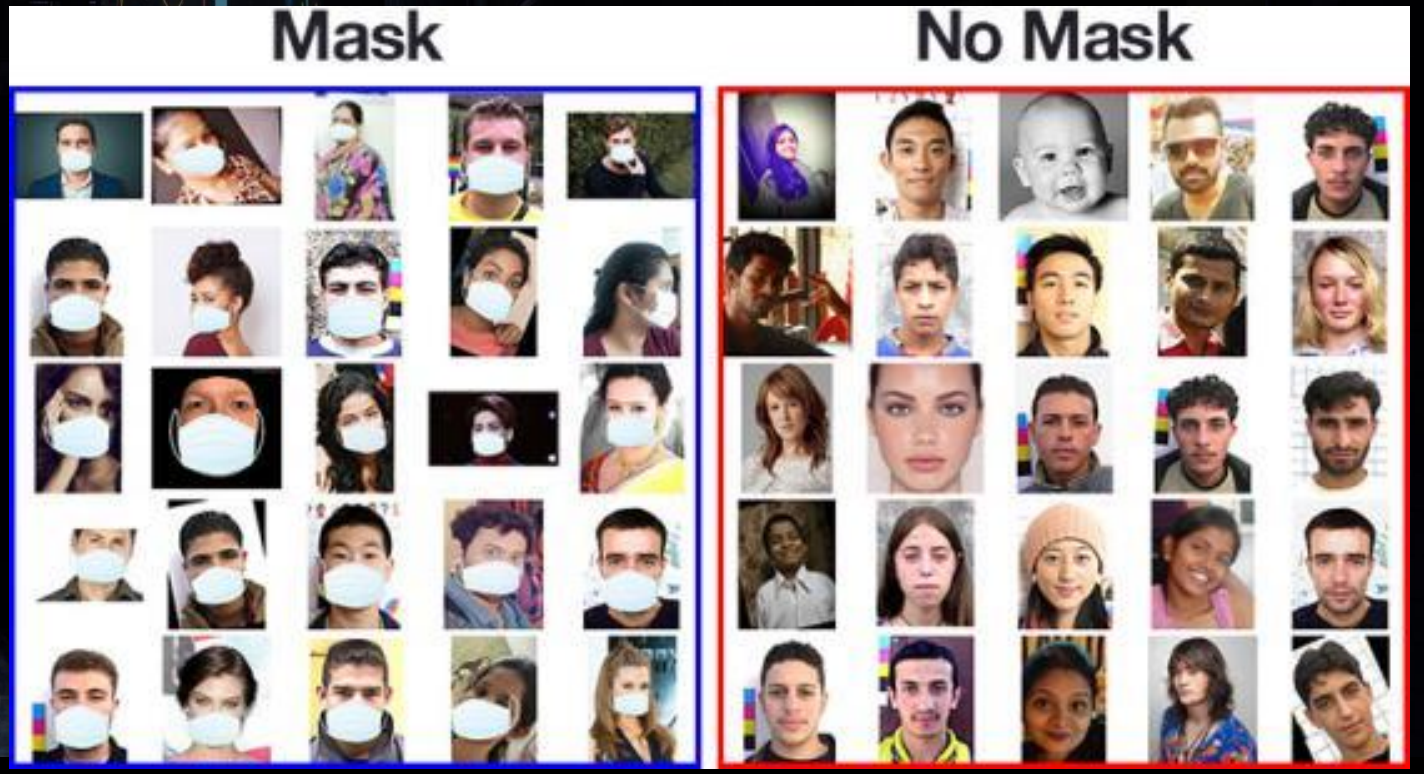
brainhack

CV use cases

V11: Bringing it all together

CV developments

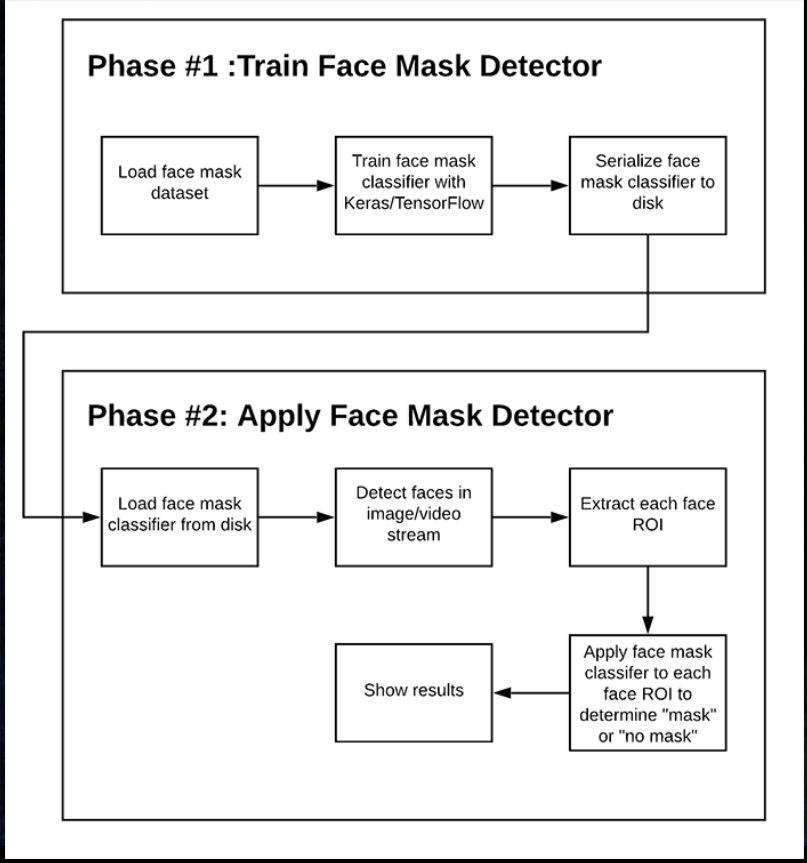
- Emotional recognition (Realeyes etc.)
- Embedded Vision
- Extended Reality (VR and AR)
- Edge Computing



Source: *Adrian, 2020, PyImage.com*

V11: Bringing it all together

CV pipeline

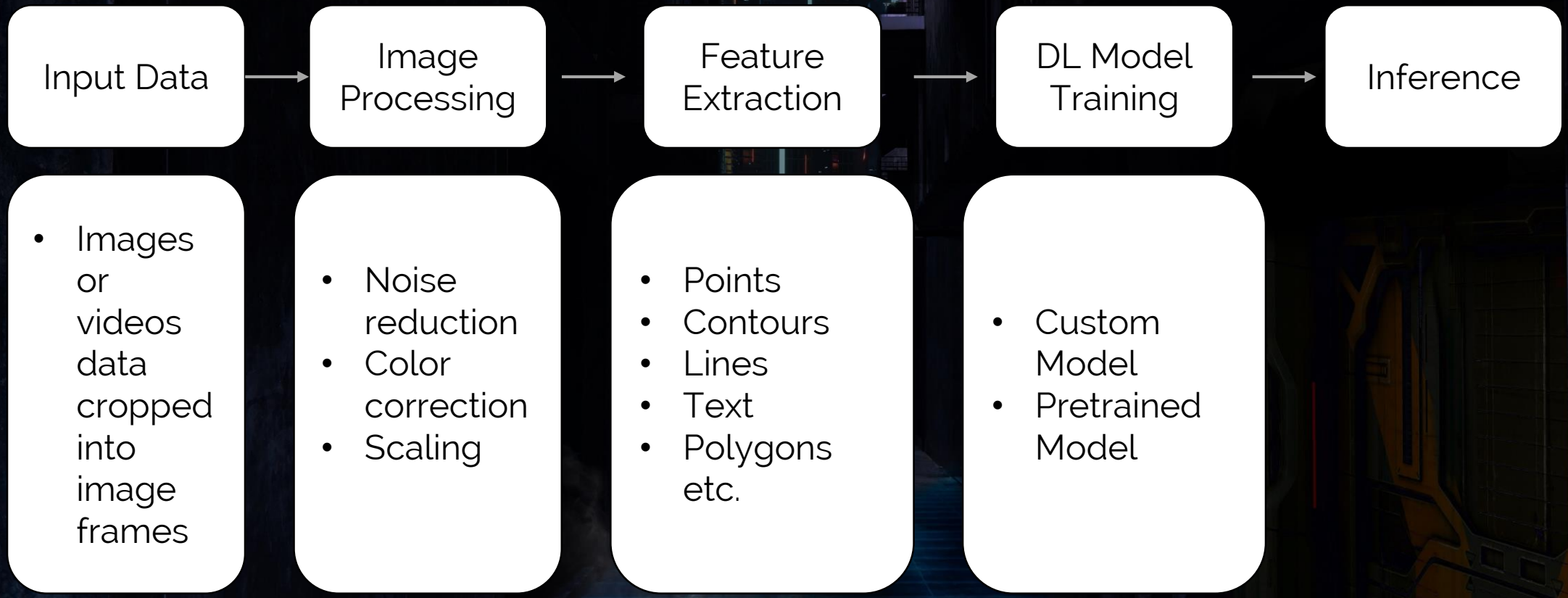


1. Training:
   - Focus on loading face mask detection dataset from disk
   - Training a model (using Keras/TensorFlow) on this dataset
   - Then serializing the face mask detector to disk
2. Deployment:
   - Performing face detection
   - Classifying each face as with_mask

Source: *Adrian, 2020, PyImage.com*

V11: Bringing it all together

CV pipeline



V11: Bringing it all together

CV limitations

- Ethics
- Privacy
- Accuracy
- Google's image recognition is still not perfect, there has been a scenario where the computer vision algorithm did mistake by tagging a picture of two dark-skinned people as "gorilla" hence causing embarrassment for the company (Huawei Forum, 2020)

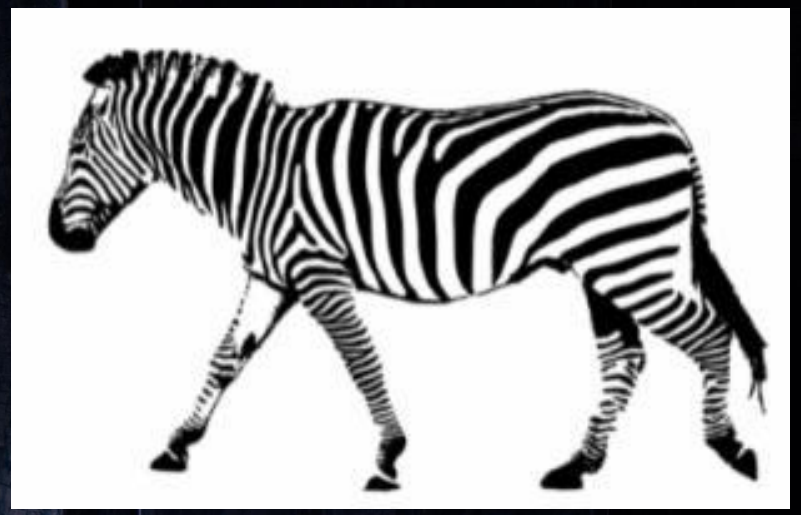
V12: Image Processing

Agenda

- Appreciate image processing
- Appreciate image processing techniques

V12: Image Processing

Image



Height

Width

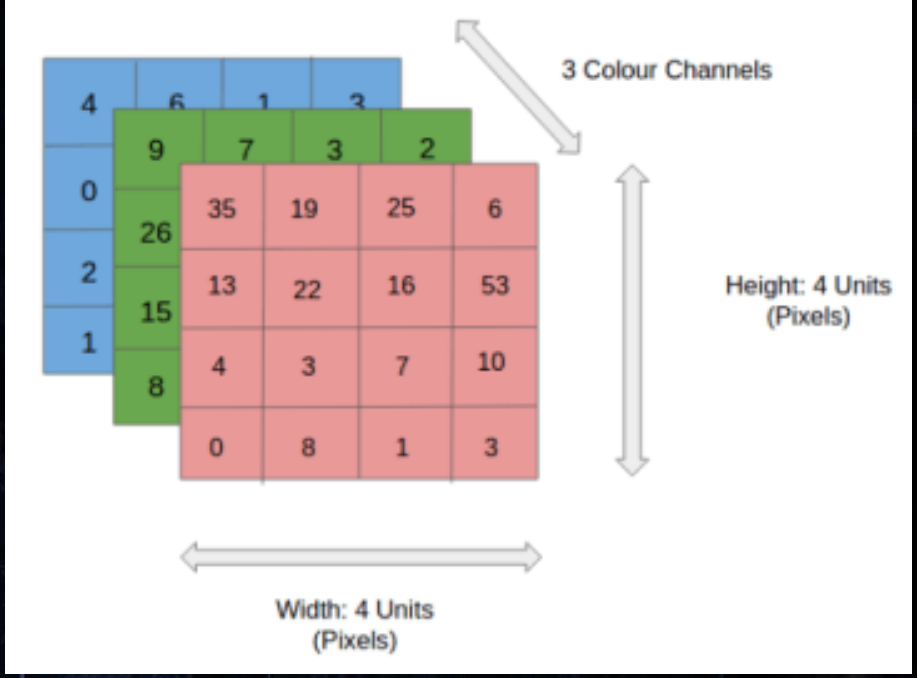
- It can be represented by a matrix of size (height, width).

DSTA
Defence Science & Technology Agency

V12: Image Processing

brainhack

Image



- It can be represented by a matrix of size (height, width).
- If it is in color, it can be represented as a **tensor** of size (height, width, channels), where the **channels** correspond to the RGB colors

DSTA
Defence Science & Technology Agency

V12: Image Processing

brainhack

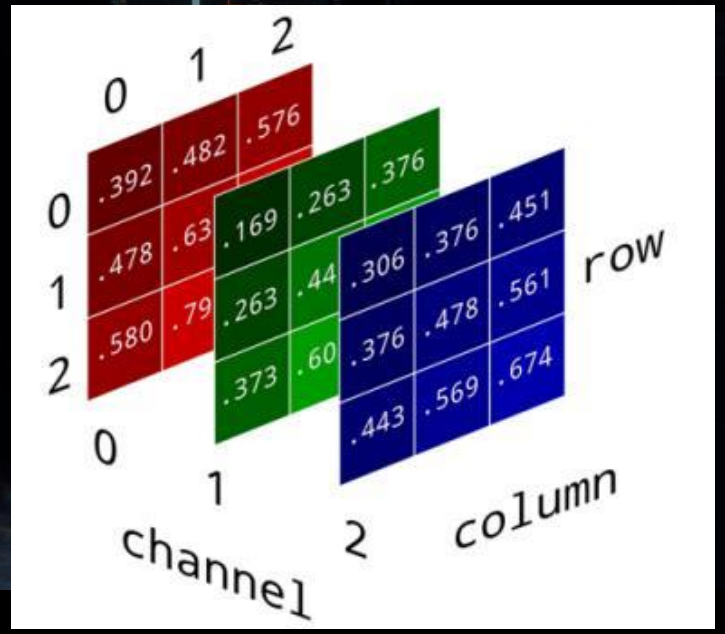
Image Processing Need

- As in any ML algorithm, your inputs should have the same size. The choice of the resolution is very important.
- Why too large images are not that important?

You can still see the zebra, can't you?

Image Processing - Normalization

- You remember that neural networks converge faster if the inputs are *somewhat* normalized (typically around -1 and 1)?
- Well. You have to do the same with the image pixels whose values are between 0 and 255 (for each color) initially.
- Divide the data by 255



V12: Image Processing

Image Processing – Mirroring

- Data augmentation is the process of creating some additional data from your initial dataset. And it's actually very simple with images.

V12: Image Processing

Image Processing – Cropping

- Data augmentation is the process of creating some additional data from your initial dataset. And it's actually very simple with images.

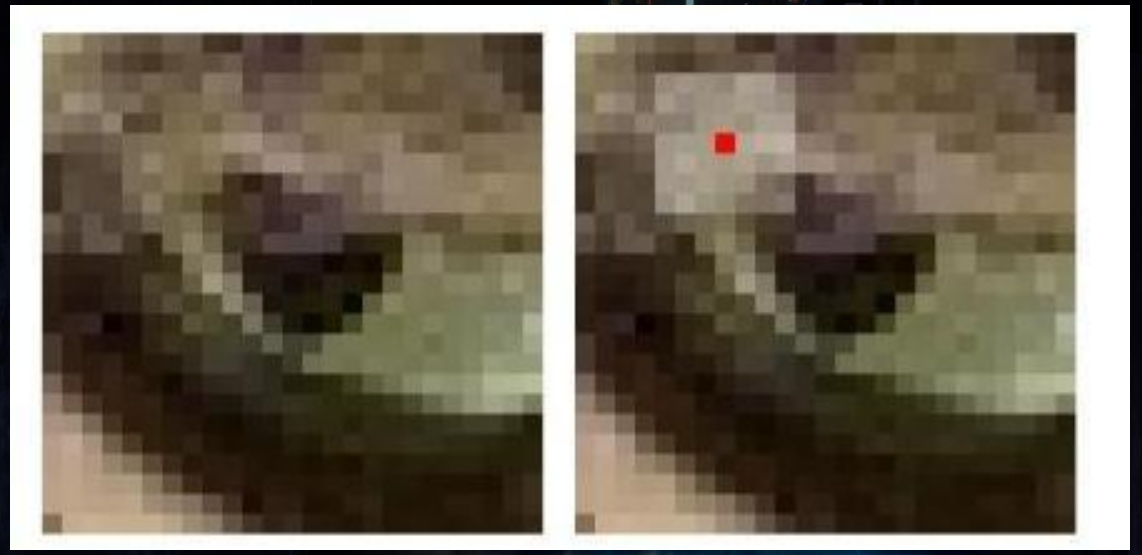
DSTA
Defence Science & Technology Agency

V12: Image Processing

brainhack

Image Processing – Blur/Sharpen Filter

- Apply a blur/sharpen filter
- Alter the pixels in the matrix



Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

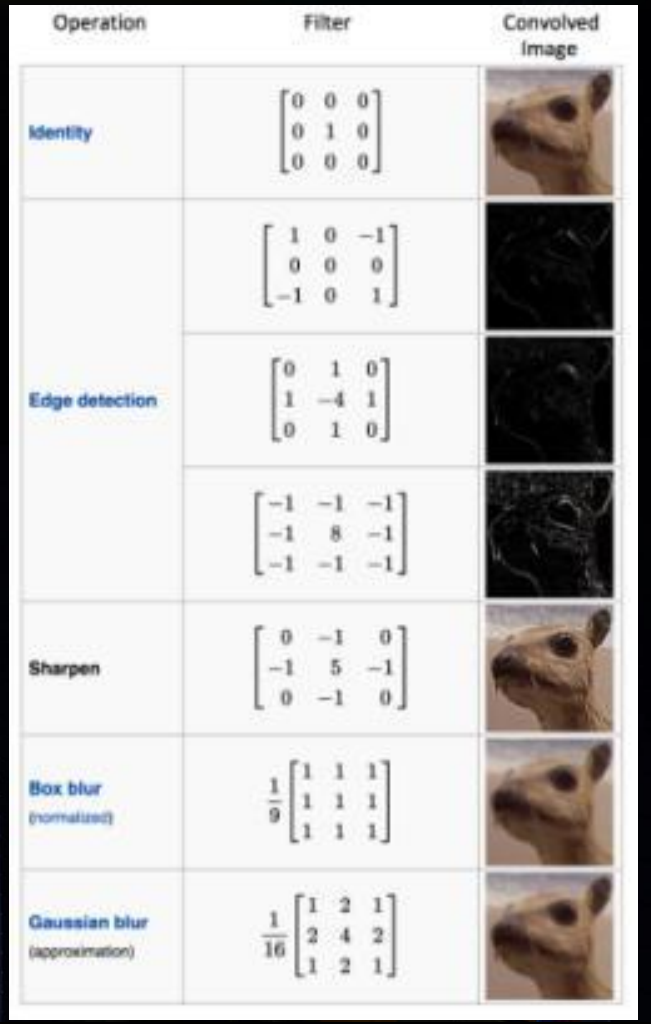
DSTA
Defence Science & Technology Agency

V12: Image Processing

brainhack

Image Processing – Kernel

- In a blur, we consider a rectangular group of pixels surrounding the pixel to filter.
- This group of pixels, called the *kernel*, moves along with the pixel that is being filtered.
- To apply this filter to the image pixel, a weighted sum of the color values of the pixels in the kernel is calculated.
- Larger kernels have more values factored in, and this implies that a larger kernel will blur the image more than a smaller kernel.



V12: Image Processing

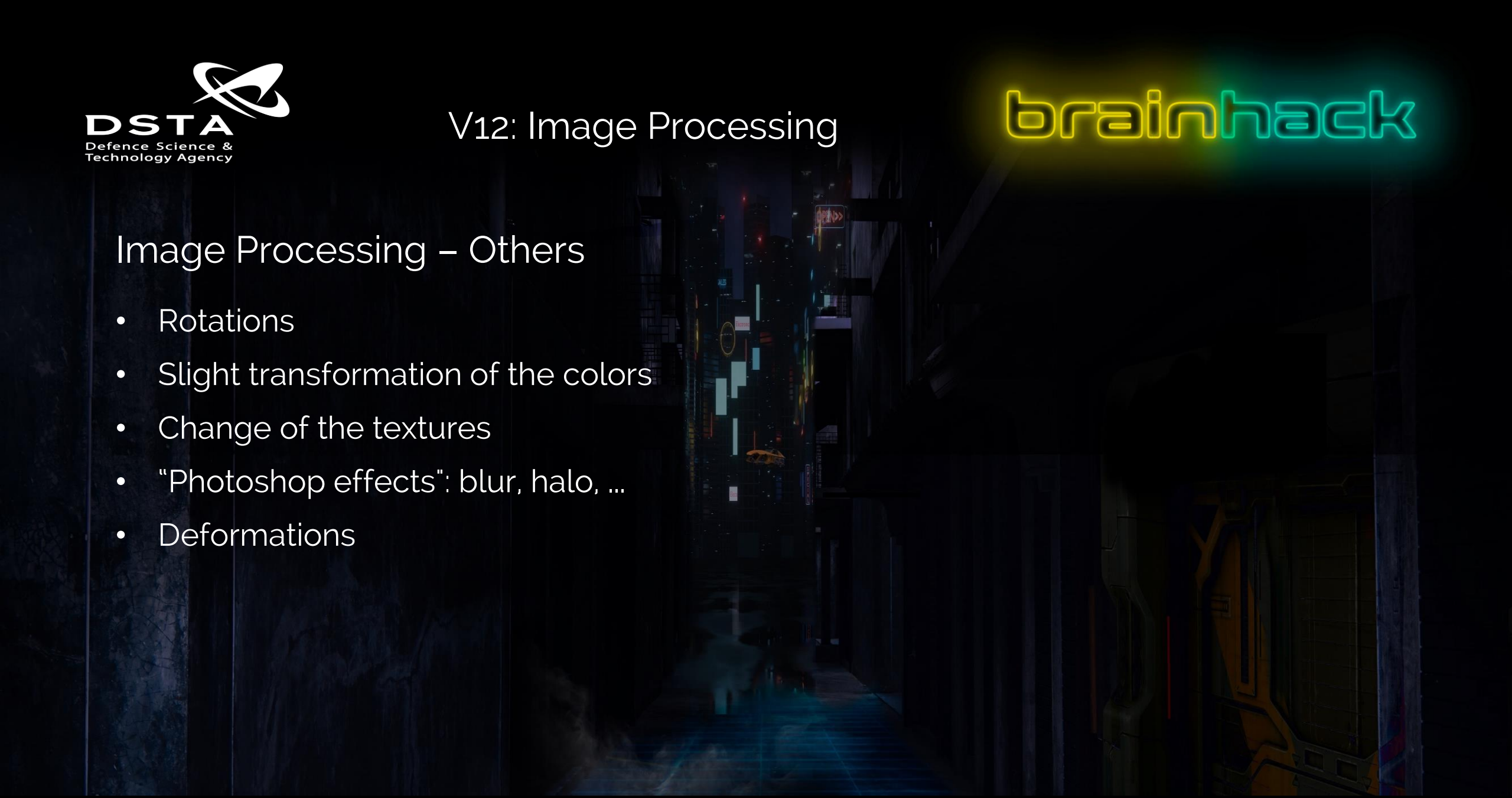


Image Processing – Others

- Rotations
- Slight transformation of the colors
- Change of the textures
- “Photoshop effects”: blur, halo, …
- Deformations

V13: Convolutional Neural Network I

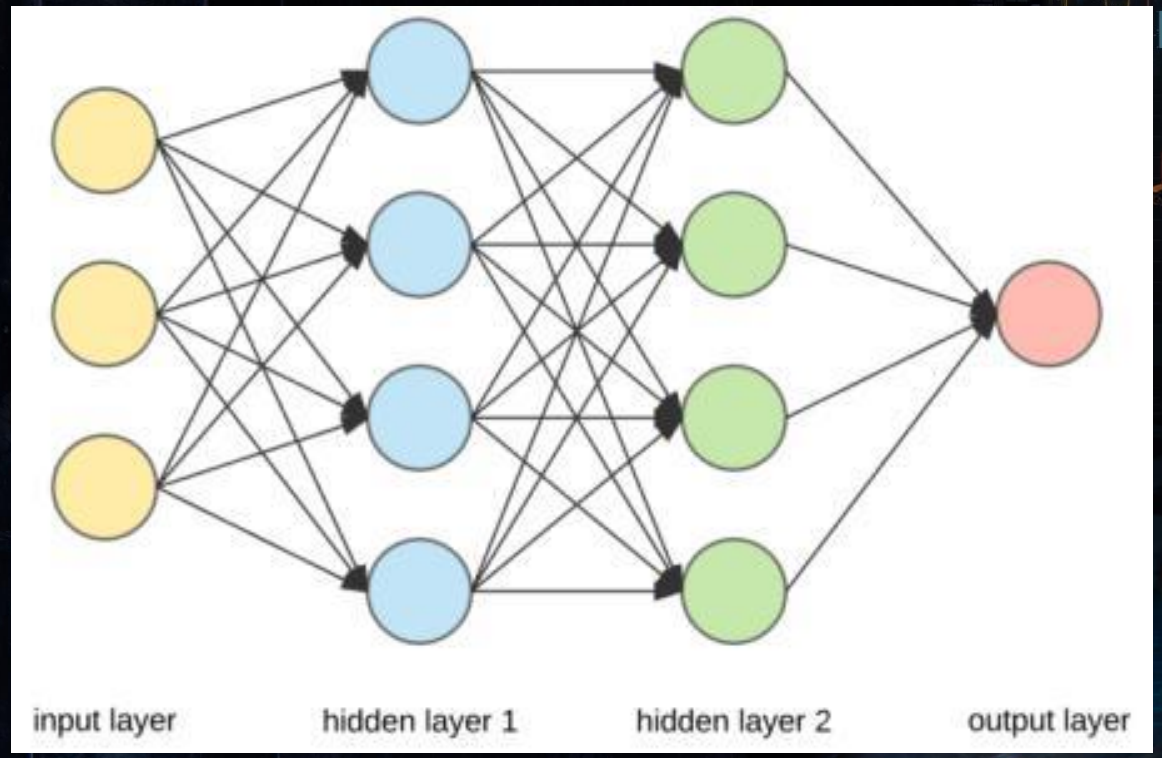
Agenda

- Appreciate multi-class and binary classification
- Appreciate class probabilities output
- Appreciate CNN's retainment of spatial features

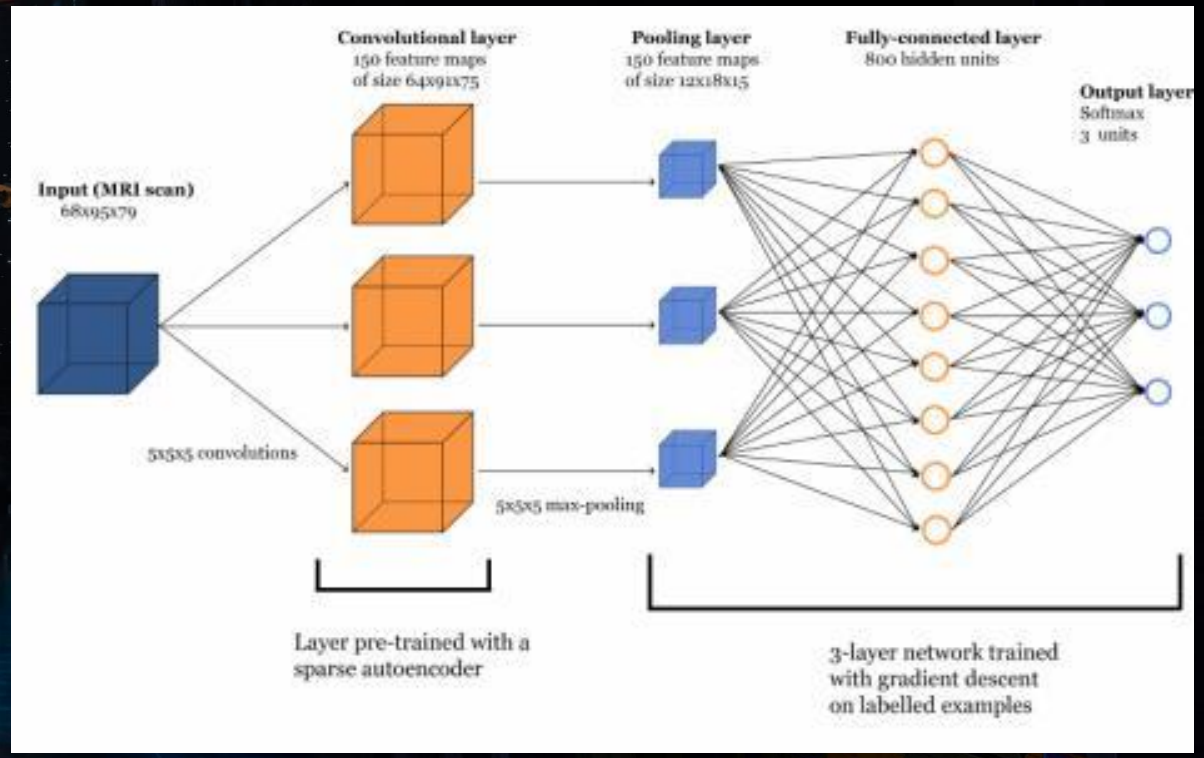
V13: Convolutional Neural Network I

CNN architecture

Binary Class



Multi-Class



V13: Convolutional Neural Network I

Class probabilities due to Softmax

Class 1 --- 0.10

Class 2 --- 0.11

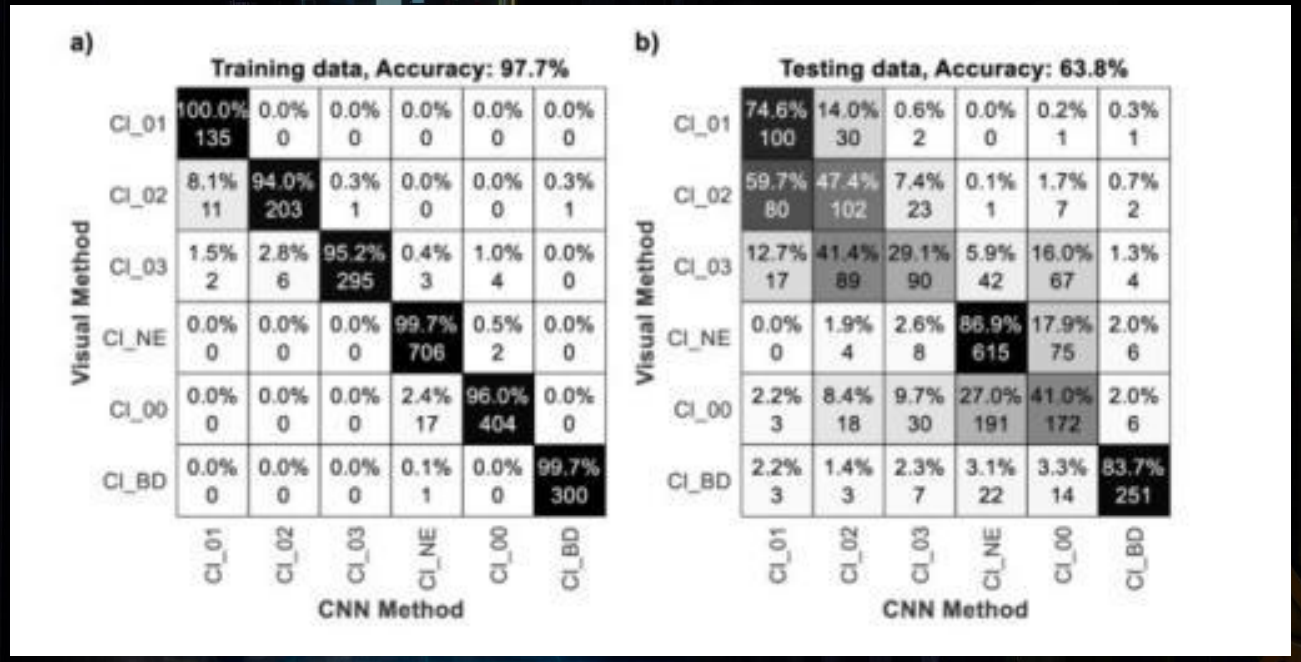
Class 3 --- 0.12

Class 4 --- 0.13

Class 5 --- 0.14

Class 6 --- 0.15

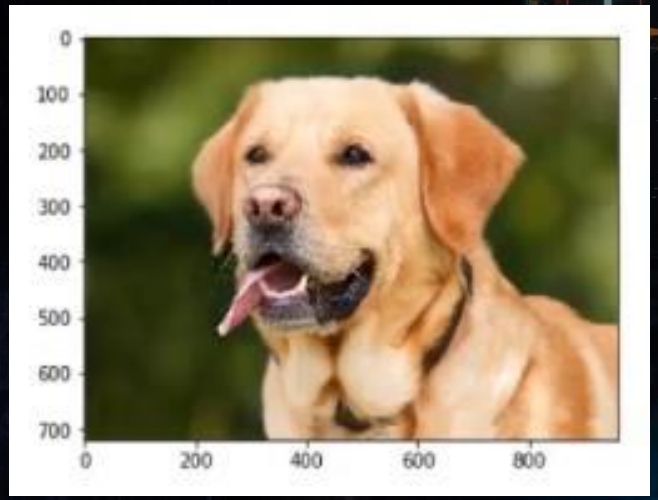
Confusion Matrix



V13: Convolutional Neural Network I

CNN most popular for image problems

1D image



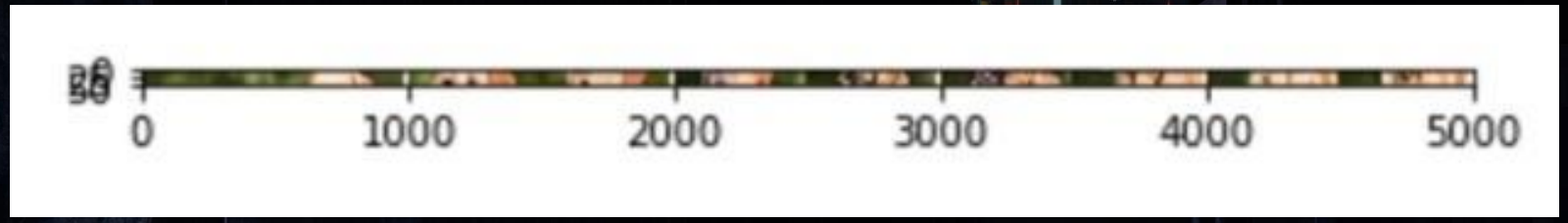
2D image

Are they the same? *Yes, they are*

V13: Convolutional Neural Network I

CNN most popular for image problems

1D image A

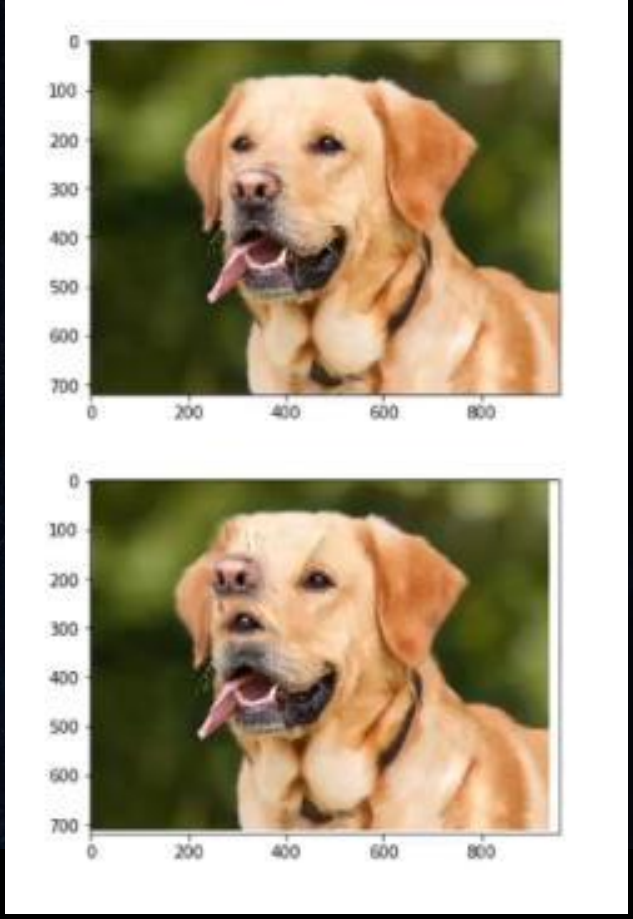


1D image B

Are they the same? *No, they not but can you tell?*

V13: Convolutional Neural Network I

CNN most popular for image problems



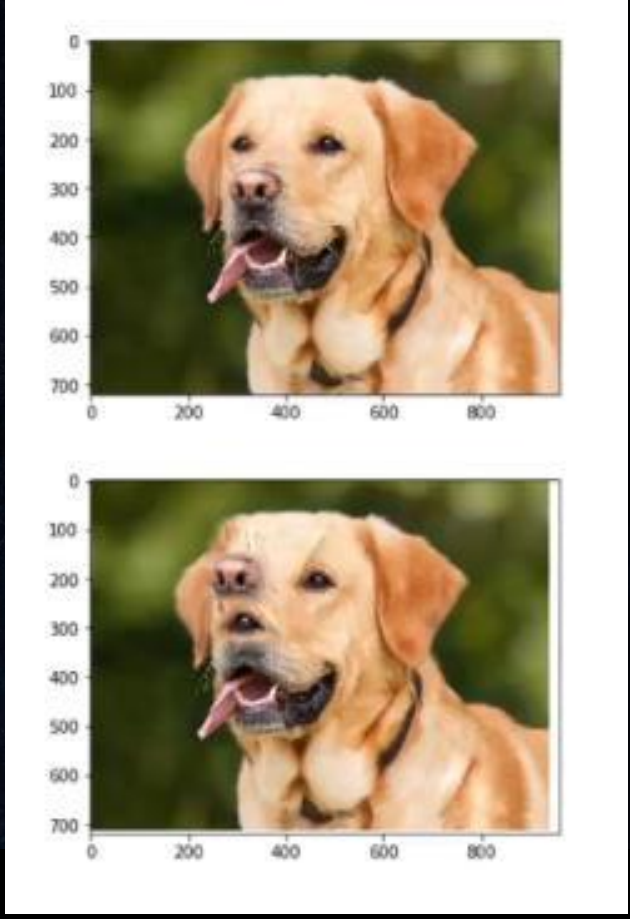
2D image A

2D image B

Are they the same? *No, they not but you can tell*

V13: Convolutional Neural Network I

CNN most popular for image problems



2D image A

2D image B

Are they the same? *No, they not but you can tell*

- CNN keeps the spatial orientation as it extracts the features within the layers
- CNN extract the images with layers that aggregate the weighted operations on the pixels thereby reducing the parameters drastically
- Main advantage of CNN is that it exploits the local connectivity and parameter sharing. This results in spatial and translational invariance, and allows us to go deeper with less parameters.

V14: Convolutional Neural Network II

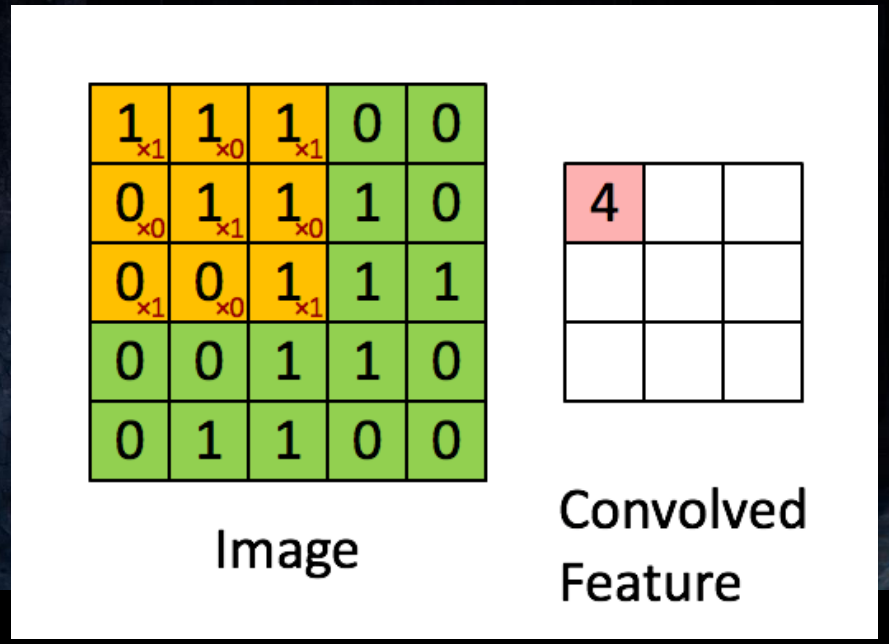
Agenda

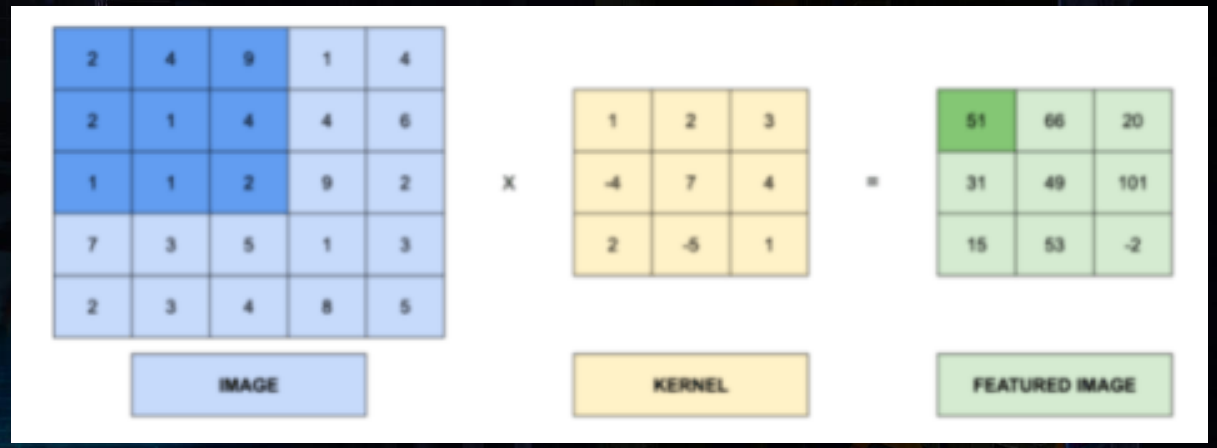
- Appreciate the components of CNN
- Appreciate the optimizer and loss function

CNN – Convolutional Layers

A **convolution** corresponds to a mathematical operation where a **subpart** of the image is **convolved** with a **kernel**.

- This operation is an element-wise *dot multiplication* (not a matrix multiplication)



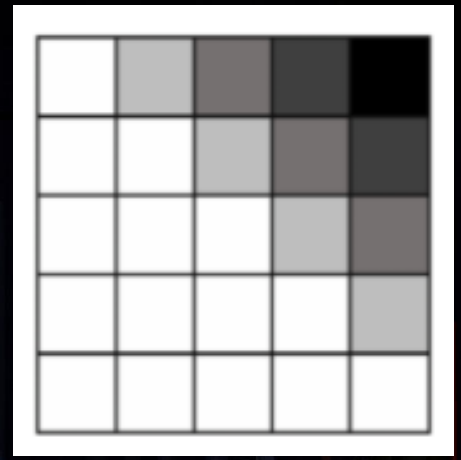


CNN – Kernel Weights

- Optimizer learns and decides on the kernel weights
- Extract key features from image

$$K = \begin{pmatrix} 0 & 0.25 & 0.5 & 0.75 & 1 \\ 0 & 0 & 0.25 & 0.5 & 0.75 \\ 0 & 0 & 0 & 0.25 & 0.5 \\ 0 & 0 & 0 & 0 & 0.25 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

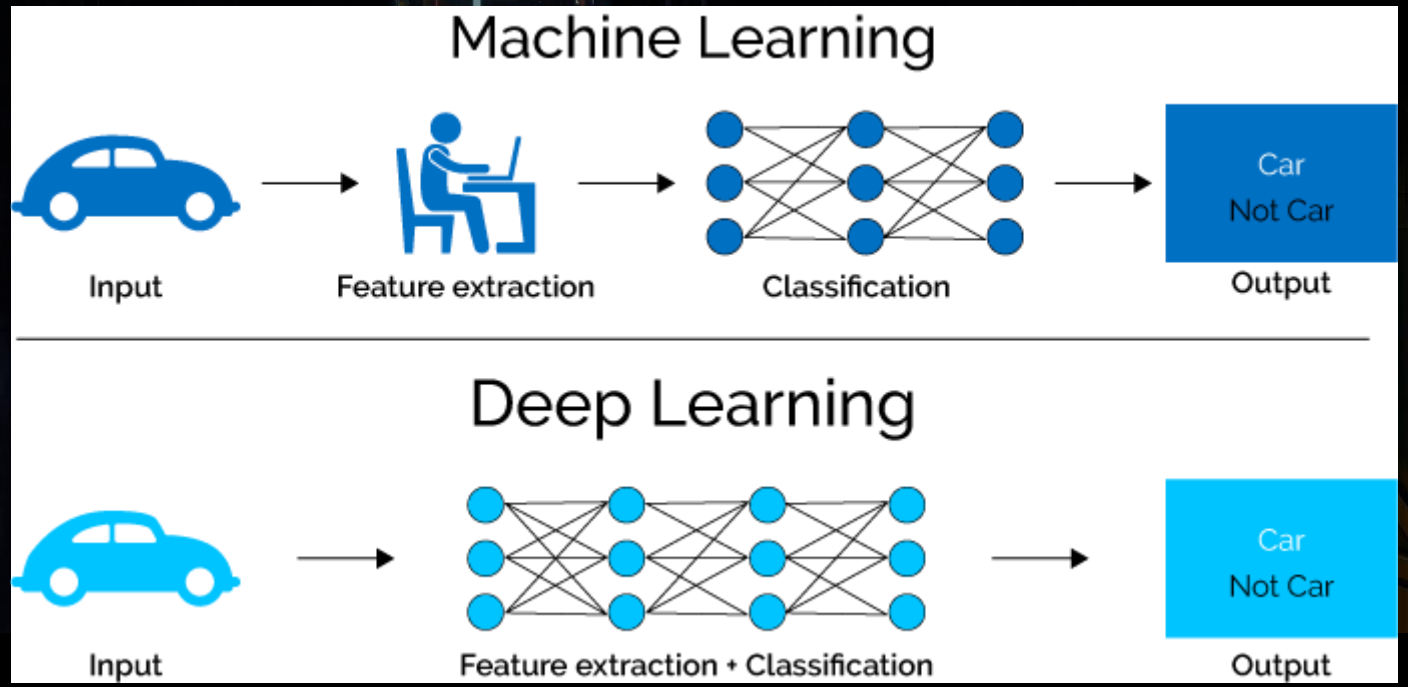
$$K = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{pmatrix}$$



V14: Convolutional Neural Network II

CNN != Machine Learning

- Deep Learning differs from machine learning in that the CNN itself is used to extract features, breaking images down into smaller matrix before finally predicting the correlation to the final output classes

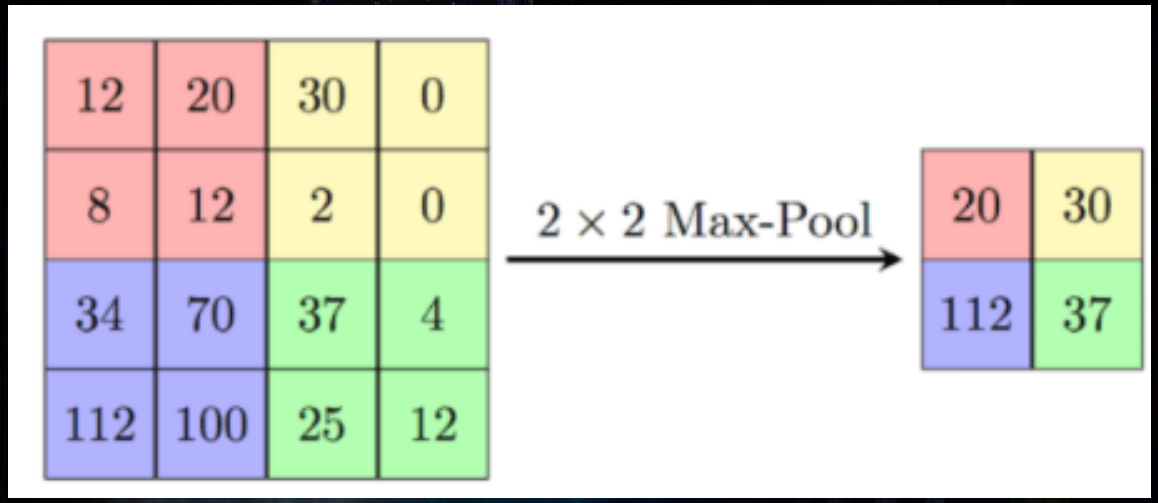


Source: *Sourish, 2018, TowardsDataScience*

CNN – Pooling Layers

A **pooling** corresponds to a reduction of the size of the output

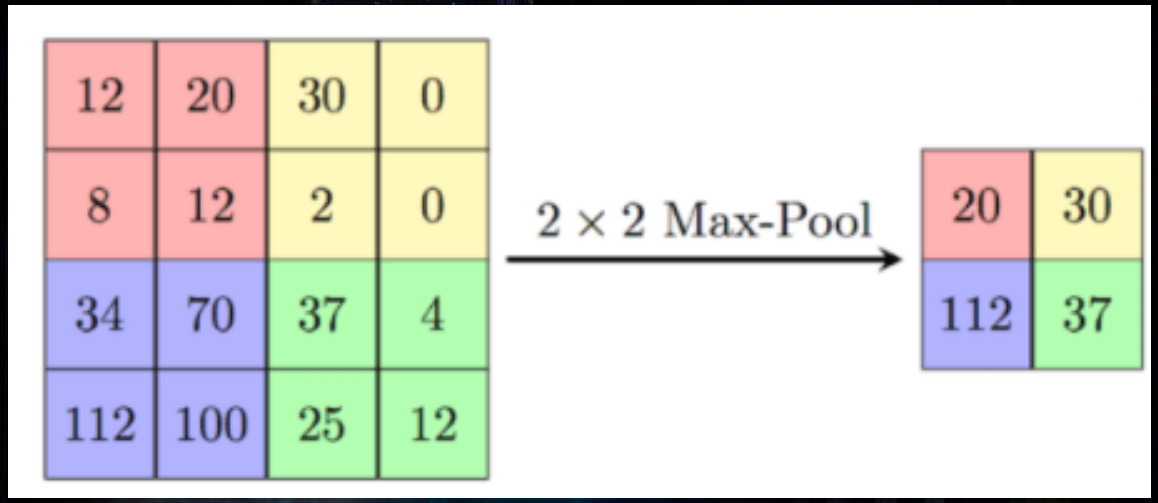
- **`MaxPool`** selects the maximum intensity value of the pixel with a given subpart of the image
- Similar to a kernel but has no convolution operation, just take the max of the pixels fragmented into the kernel size
- Zero parameters to train



V14: Convolutional Neural Network II

CNN – Pooling Layers

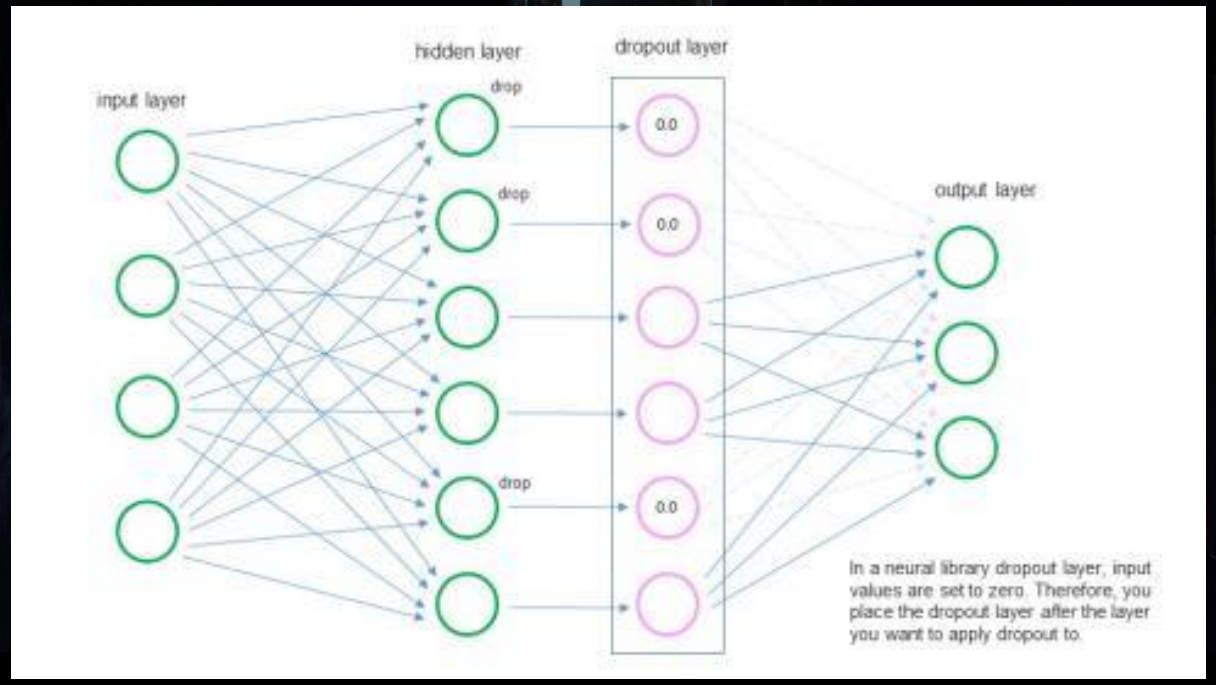
- Reduces dimensional complexity and cost of training
- Makes assumptions about sub-regions of the image
- Can prevent model from over-learning the data



V14: Convolutional Neural Network II

CNN – Dropout layer

- Simplest way to help prevent model from over-learning the data
- Entirely random with no assumptions



Source: *James, 2018, McCaffrey*

V15: Convolutional Neural Network III

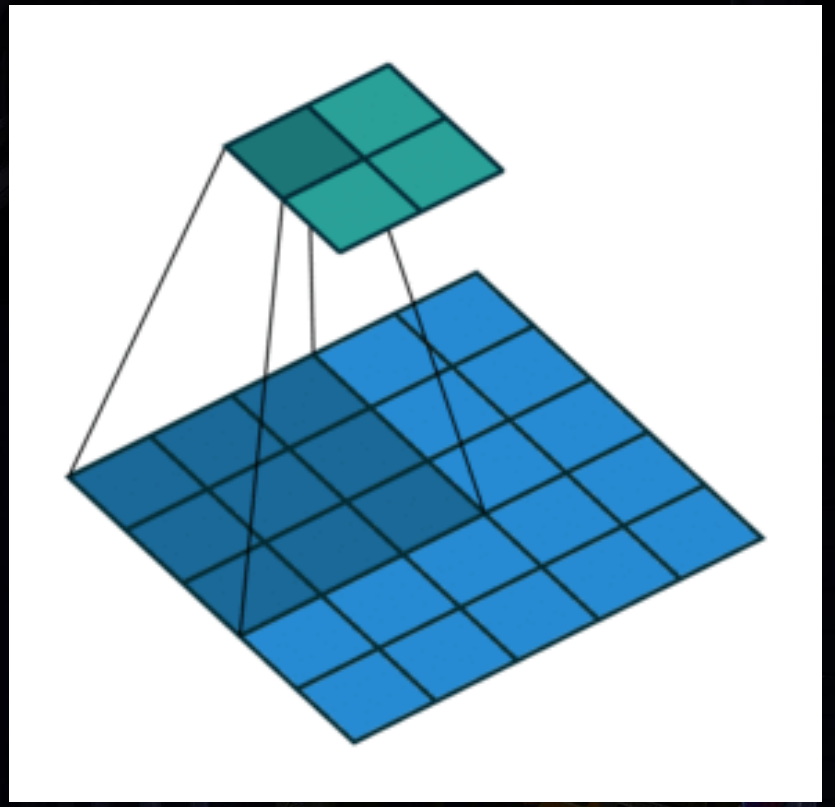
Agenda

- Appreciate hyperparameter tuning
- Appreciate model evaluation

V15: Convolutional Neural Network III

Convolutional Hyperparameters - Striding

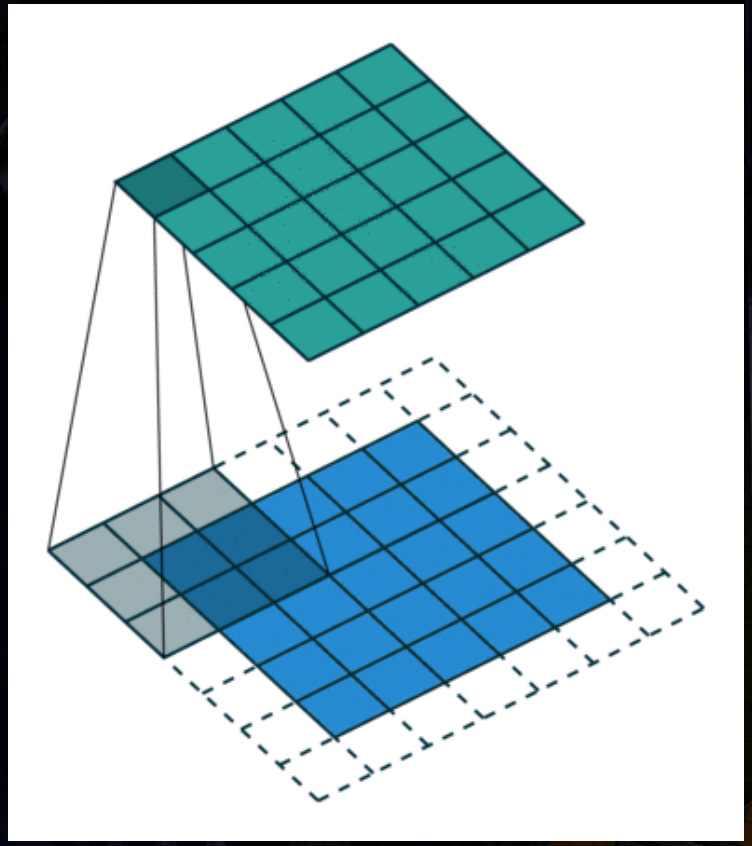
- The amount by which the filter shifts is the stride.
- Increasing the stride will result in less parameter outputs and also smaller data output
- Intuitively, the image sub-regions has less overlap and smaller spatial dimensions is achieved with striding
- 7x7 input data with stride of 1 will get 5x5 output
- 7x7 input data with stride of 2 will get 3x3 output
- Stride controls how the kernel filter in the convolutional layer convolves around the input data.



V15: Convolutional Neural Network III

Convolutional Hyperparameters - Padding

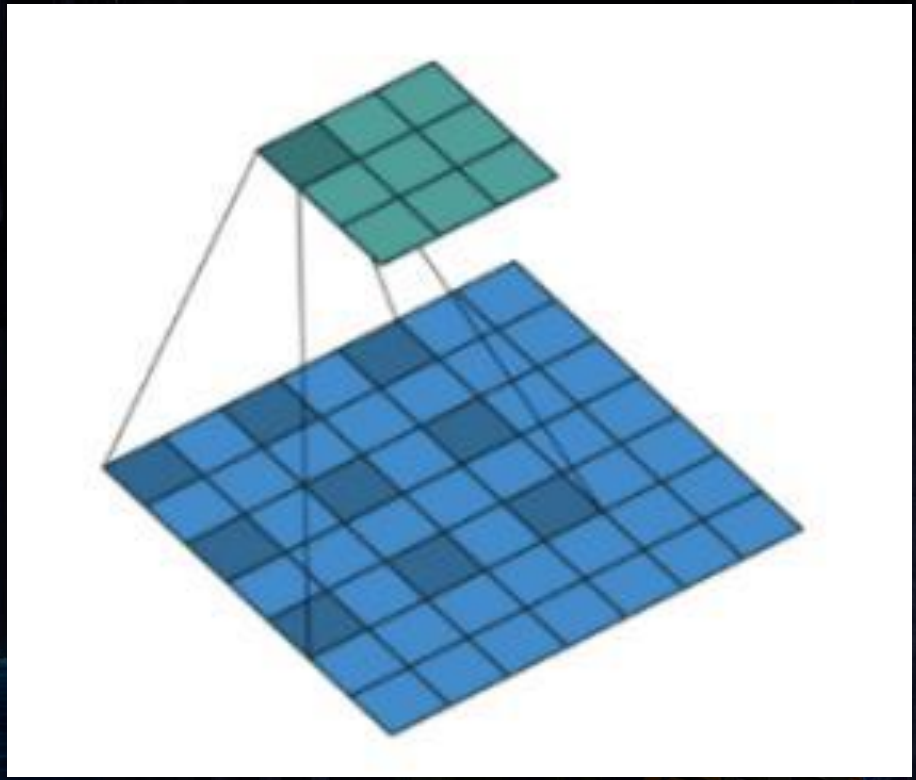
- As convolutional layers always reduce the output dimension, the way to keep dimension intact is padding
- Increasing the padding will add more white spaces at the ends of the image
- Padding intuitively increases the importance of the borders



V15: Convolutional Neural Network III

Convolutional Hyperparameters - Dilation

- More common in image segmentation where each pixel is labelled by its corresponding class
- Dilation helps to increase a "global view" of the data exponentially at a lower cost
- Increasing the dilation increases the spacing between kernel points



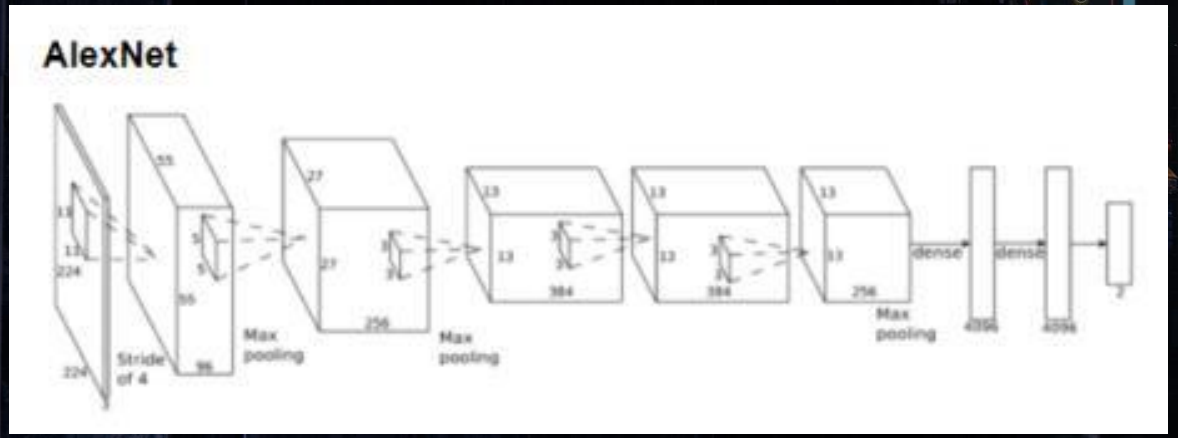
V16: Convolutional Neural Network IV

Agenda

- Appreciate pretrained model
- Appreciate benefits of transfer learning
- Sample codes

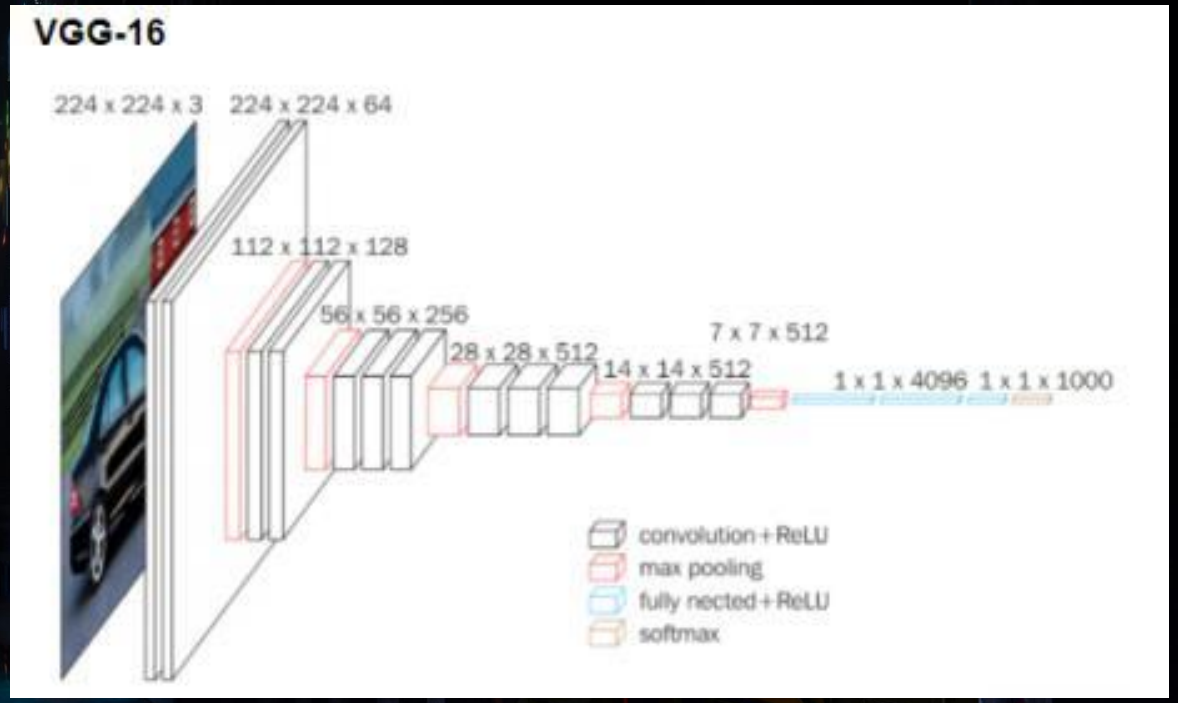
V16: Convolutional Neural Network IV

SOTA – State Of The Art



8 layers

- First to implement ReLU activation functions
- Stacked on LeNet



13 layers

- Stacked on AlexNet

DSTA
Defence Science & Technology Agency

V16: Convolutional Neural Network IV

brainhack

SOTA – State Of The Art

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
VGG16	528 MB	0.713	0.901	138,357,544	23
InceptionV3	92 MB	0.779	0.937	23,851,784	159
ResNet50	98 MB	0.749	0.921	25,636,712	-
Xception	88 MB	0.790	0.945	22,910,480	126
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
ResNeXt50	96 MB	0.777	0.938	25,097,128	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

V16: Convolutional Neural Network IV

Transfer Learning

- It can be seen that the kernels are learning some patterns: edges, outlines, shapes, gradients, ... which seems to be independent from any task.
- You can load the same CNN, remove the last layer and replace them by the one suited to your problem, and then learn the weights of these last layers.

- Hence, transfer learning!

- Most classical architectures are available in PyTorch, with pretrained weights. This allows us to do transfer learning i.e. using the knowledge contained in the convolutions of the network to perform another task on the images

V17: Object Detection I

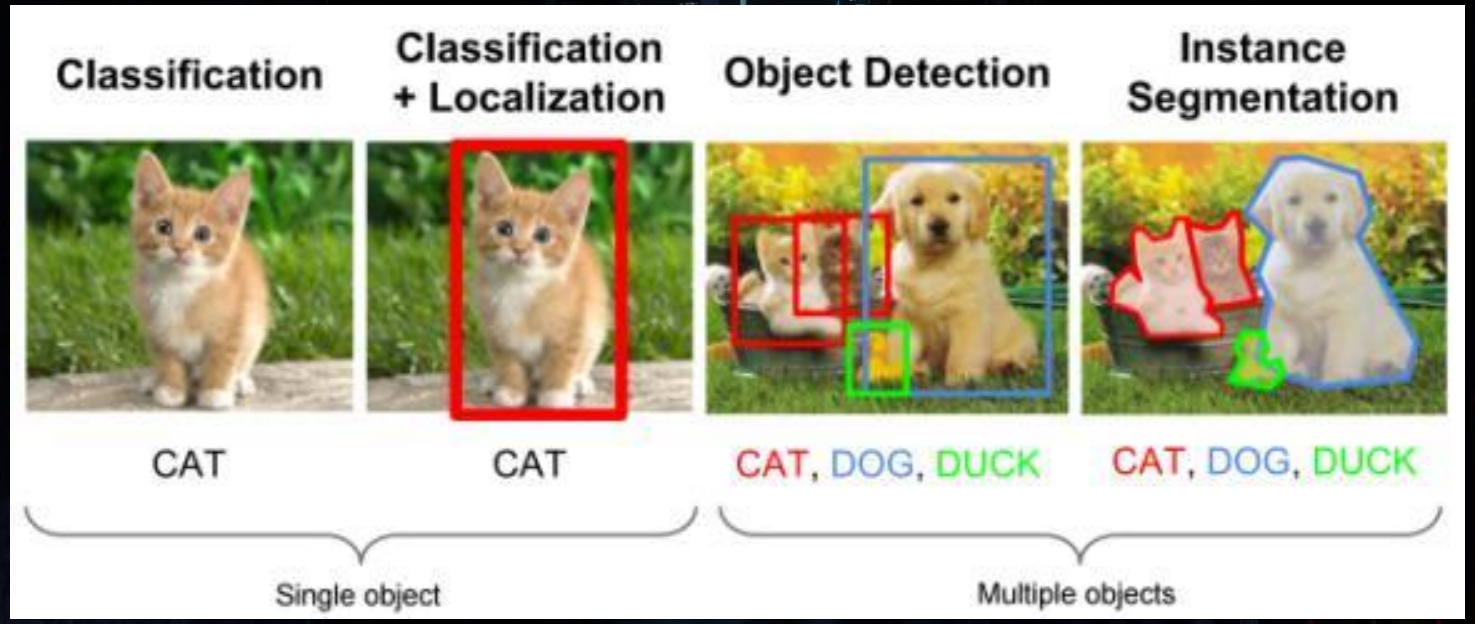
Agenda

- Appreciate object detection
- Appreciate procedures and annotations
- Appreciate object detection versus image classification

V17: Object Detection I

Image Classification

- 1 image 1 label



Object Detection

- 1 image multiple labels

Source: *Michal, KDNuggests*

DSTA
Defence Science & Technology Agency

V17: Object Detection I

brainhack

Object Detection vs. Image Classification

- Classification can be said to be easier than object detection.
- In classification all objects in a single image are grouped or categorized into a single class.
- On the other hand, in object detection, each object of a single class in an image has to be matched to a certain class and also localized.

DSTA
Defence Science & Technology Agency

V17: Object Detection I

brainhack

Instance Segmentation

- Instance segmentation assigns a label to every pixel in an image in such a way that pixels with the label share certain characteristics

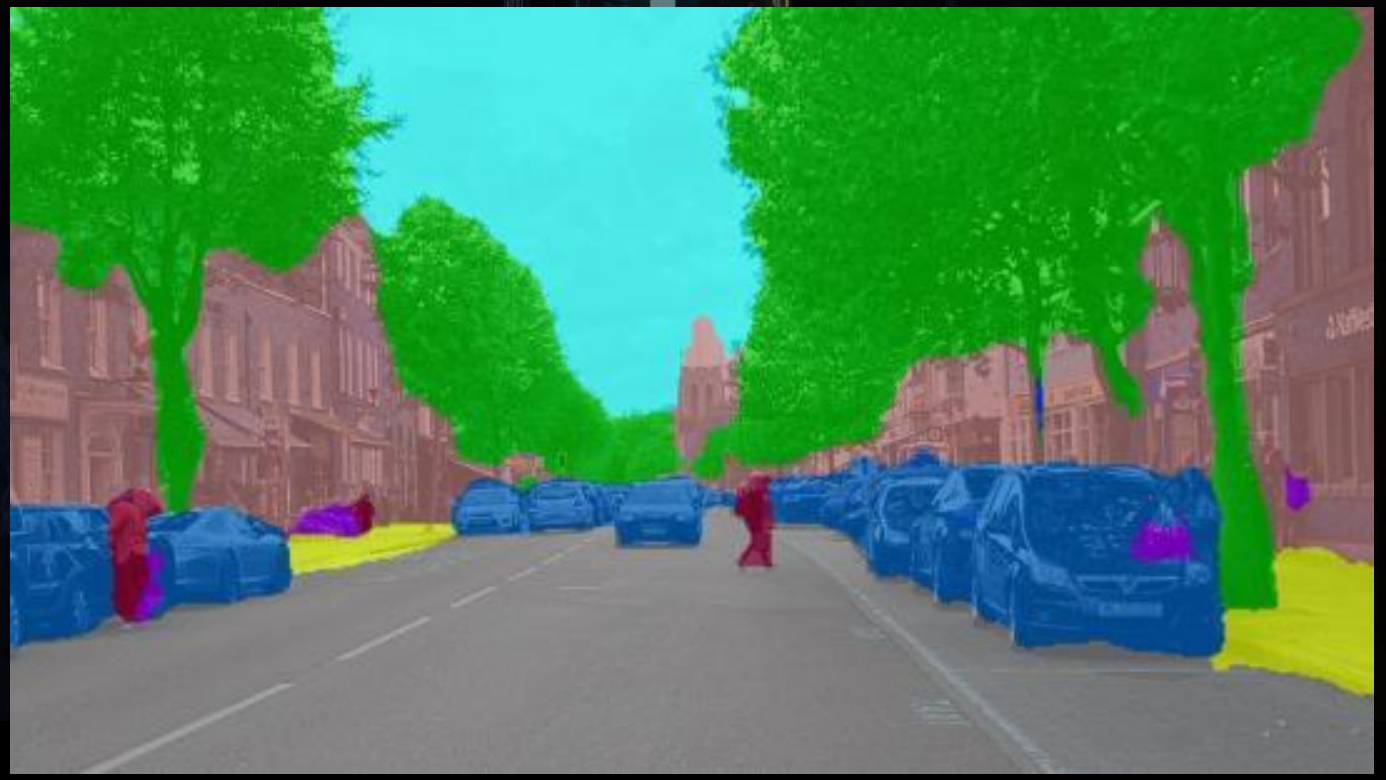
DSTA
Defence Science & Technology Agency

V17: Object Detection I

brainhack

Semantic Segmentation

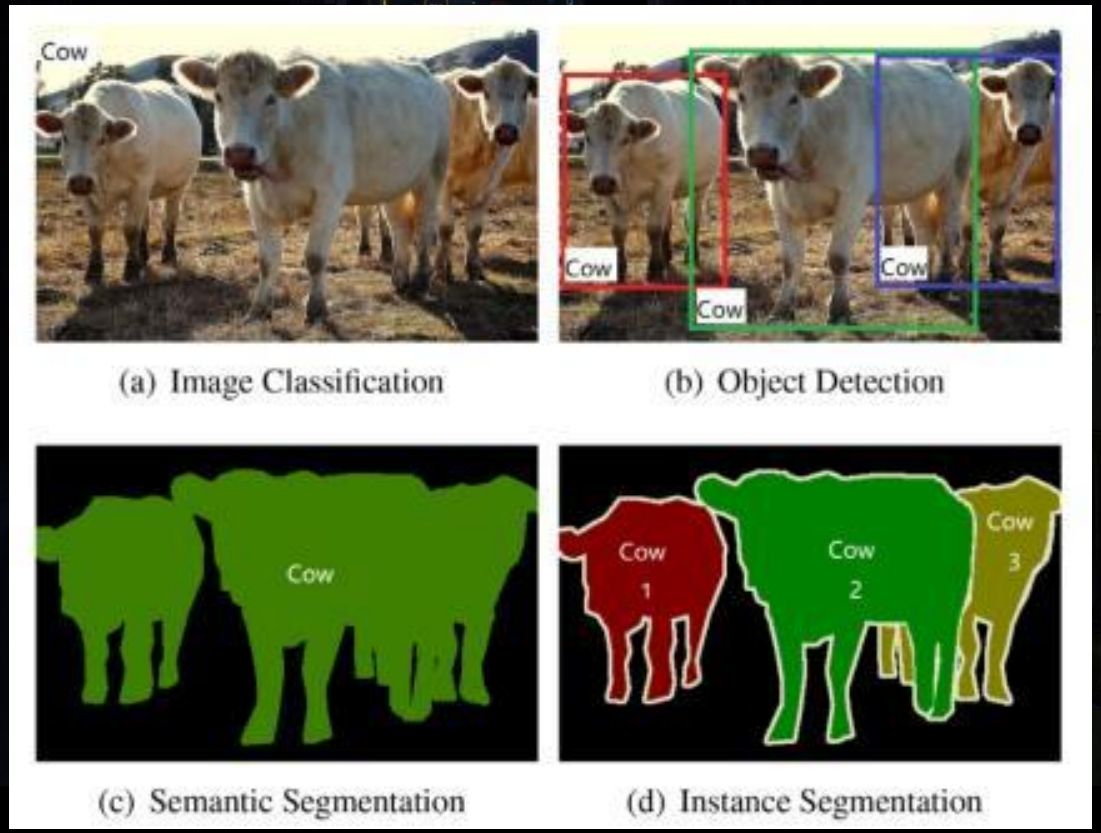
- Semantic segmentation detects and classify all objects as the same class in the image.



V17: Object Detection I

Object Detection, Instance Segmentation, Semantic Segmentation

In summary!



Source: *Pentagonspace, 2020, Medium.com*

V17: Object Detection I

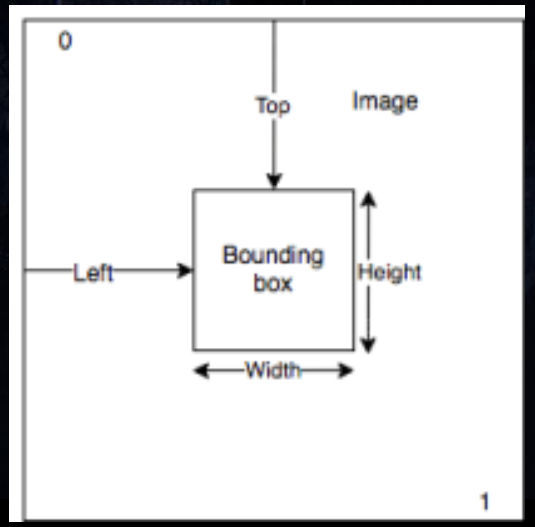
Object Detection Annotation

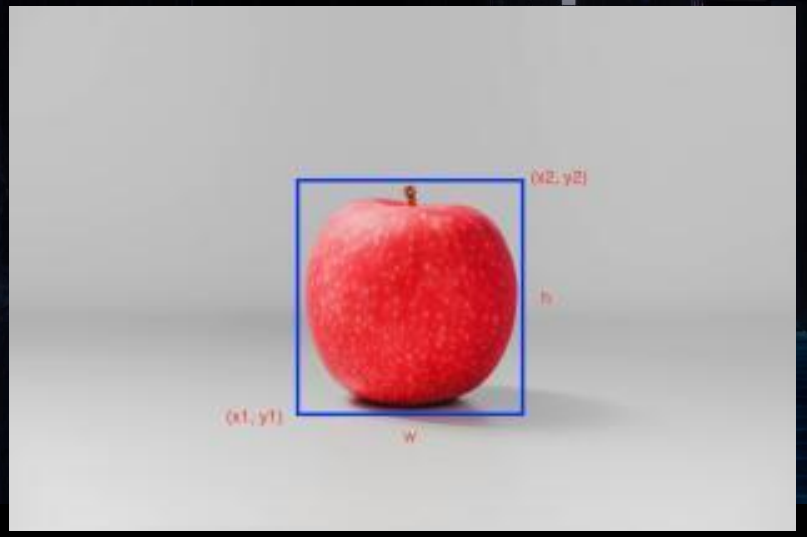
- Objects are annotated either by bounding boxes (coordinates, height, width etc.) or that of masking
- Masking enables semantic and instance segmentation
- Any tools
  - MakeSense (Bounding box, polygon, point annotation)
- Stored in formats of XML, VGG JSON, CSV or even YOLO format

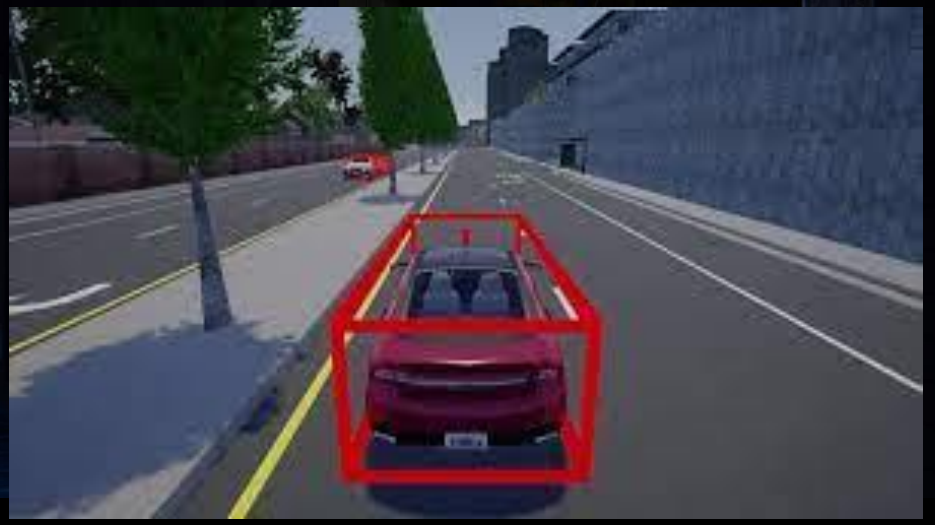
V17: Object Detection I

Object Detection

- To detect objects, a box or any shape can be drawn up to highlight the location of the objects of interest.
- Bounding boxes are usually represented by either two coordinates (x1, y1) and (x2, y2) or by one co-ordinate (x1, y1) and width (w) and height (h) of the bounding box.







DSTA
Defence Science & Technology Agency

V17: Object Detection I

brainhack

Object Detection

- Predict object type and the position in the image
- SOTA models commonly used i.e. YOLO [You Only Look Once], Mask R-CNN [Region-Based CNN]
- Involves localizing objects in the image, sometimes also indicates which pixels in the image belong exactly to the object (segmentation)

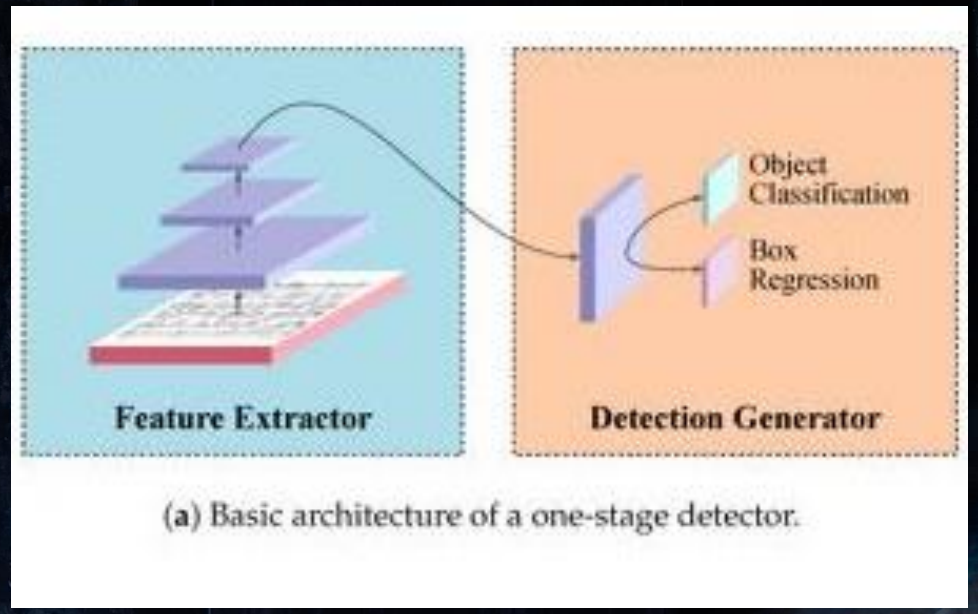
DSTA
Defence Science & Technology Agency

V17: Object Detection I

brainhack

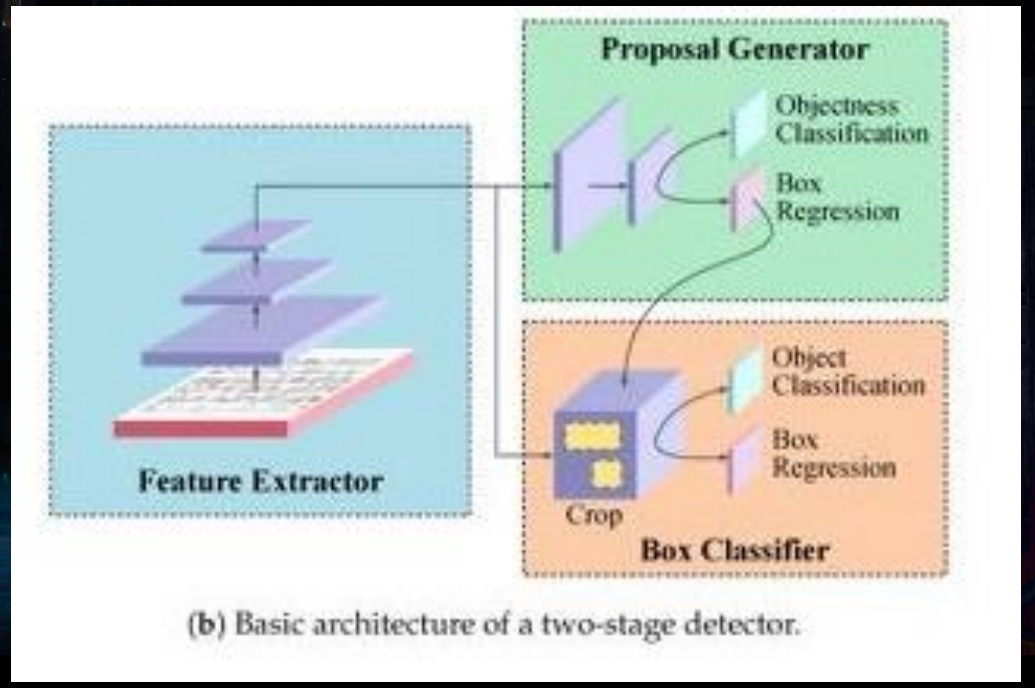
SOTA models commonly used

YOLO [You Only Look Once]
(One-Stage Detector)



(a) Basic architecture of a one-stage detector.

Mask/Faster R-CNN [Region-Based CNN]
(Two-Stage Detector)



(b) Basic architecture of a two-stage detector.

Object Detection

One-Stage Detector	Two-Stage Detector
• Directly calculates the class probabilities of bounding box coordinates via regression	• Generate sub-regions • Send sub-regions for object classification and bounding-box regression
• Accuracy tends to be lower	• Accuracy tends to be higher
• Training tends to be faster	• Training tends to be slower

V17: Object Detection I

General Procedures

1. Prepare object detection dataset ready for modeling
2. Use transfer learning or custom model to train model
3. Evaluate model on test

DSTA
Defence Science & Technology Agency

V18: Object Detection II

brainhack

Agenda

- Parsing Annotation File
- Sample codes tutorial on PyTorch
- Preparing for model learning

V18: Object Detection II

General Procedures

XML data sample

```
<annotation>
        <folder>Kangaroo</folder>
        <filename>00001.jpg</filename>
        ……
        <size>
                <width>450</width>
                <height>319</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>kangaroo</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>233</xmin>
                        <ymin>89</ymin>
                        <xmax>386</xmax>
                        <ymax>262</ymax>
                </bndbox>
        </object>
        <object>
                <name>kangaroo</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>134</xmin>
                        <ymin>105</ymin>
                        <xmax>341</xmax>
                        <ymax>253</ymax>
                </bndbox>
        </object>
</annotation>
```

Kangaroo1

Kangaroo2

Prepare object detection dataset ready for modeling

V18: Object Detection II

Prepare object detection dataset ready for modeling

General Procedures

```
# load and parse the file
tree = ElementTree.parse(filename)

# extract each bounding box
for box in root.findall('.//bndbox'):
        xmin = int(box.find('xmin').text)
        ymin = int(box.find('ymin').text)
        xmax = int(box.find('xmax').text)
        ymax = int(box.find('ymax').text)
        coors = [xmin, ymin, xmax, ymax]

# extract image dimensions
width = int(root.find('.//size/width').text)
height = int(root.find('.//size/height').text)
```

Parse the annotation as a ElementTree object

Extract all the details of the bounding boxes

Extract dimensional values, may calculate diagonal values

DSTA
Defence Science & Technology Agency

V19: Object Detection III

brainhack

Agenda

- Model Loading
- Finetuning
- Sample codes tutorial on PyTorch

DSTA
Defence Science & Technology Agency

V19: Object Detection III

brainhack

Transfer Learning

1. Fast/Faster R-CNN model pre-fit on the MS COCO object detection dataset
2. Train and improve on own custom dataset

DSTA
Defence Science & Technology Agency

V19: Object Detection III

brainhack

Transfer Learning Operating Procedures

1. Change the number of classes for your own problem
2. Change the hyperparameters (epoch, batches etc.)
    - 1 epochs is when ENTIRE dataset pass through the NN once
    - 1 batch size is the number of training examples in a single batch to input to the model
    - Iteration means the number of batches needed to complete one epoch.
        - 1 iteration means only 1 batch is needed to complete 1 epoch, in this case the batch size is the entire dataset
        - Iteraction = epochs / batchsize
    - Batch size is useful because entire dataset cannot be passed into the neural network just like how you cannot read an entire newspaper article in 1 shot

V19: Object Detection III

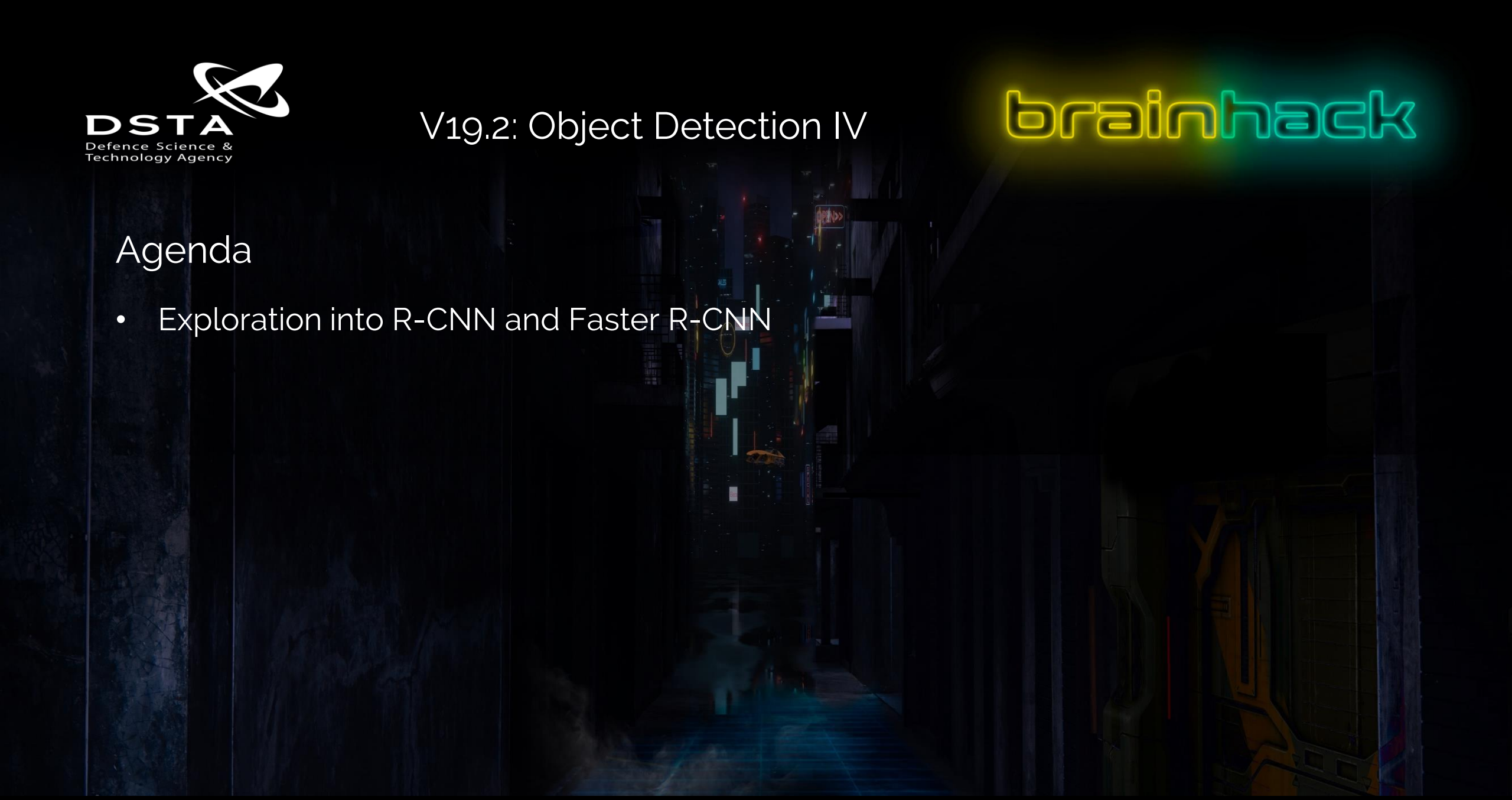
Transfer Learning Operating Procedures

3. Specify which layers to train. Usually only the heads (classifier prediction output layer)
    - SOTA layers are too many, it is unlikely to re-train entire huge deep network
    - What has learnt well, leave it. *See further from the shoulders of giants*
    - The past layers have solved the weights, so the new layers or last few layers can be optimized for your own data

DSTA
Defence Science & Technology Agency

V19.2: Object Detection IV

brainhack



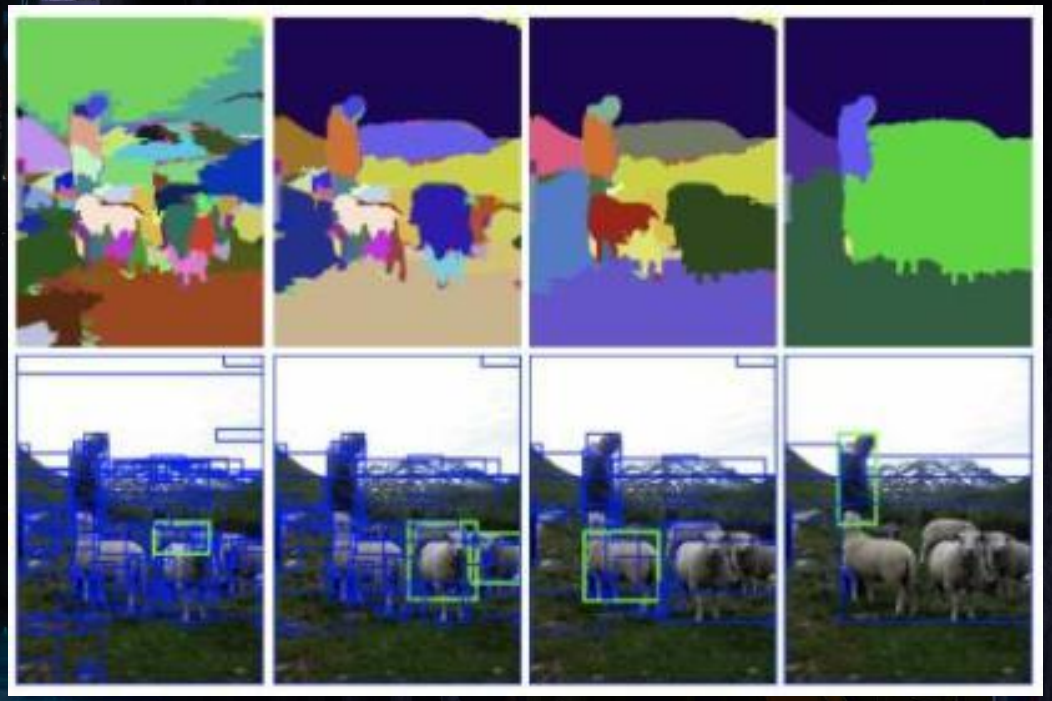
Agenda

- Exploration into R-CNN and Faster R-CNN

V19.2: Object Detection IV

How does R-CNN work?

- Find sub-regions of interest in the image using selective search by combining regions with similar pixels and textures into several rectangular boxes.
- Boxes are passed to a CNN model to classify into one of the many classes
- Background classes for removing bad proposals (Negative Examples)
- Compute the regression between predicted boxes and ground-truth boxes

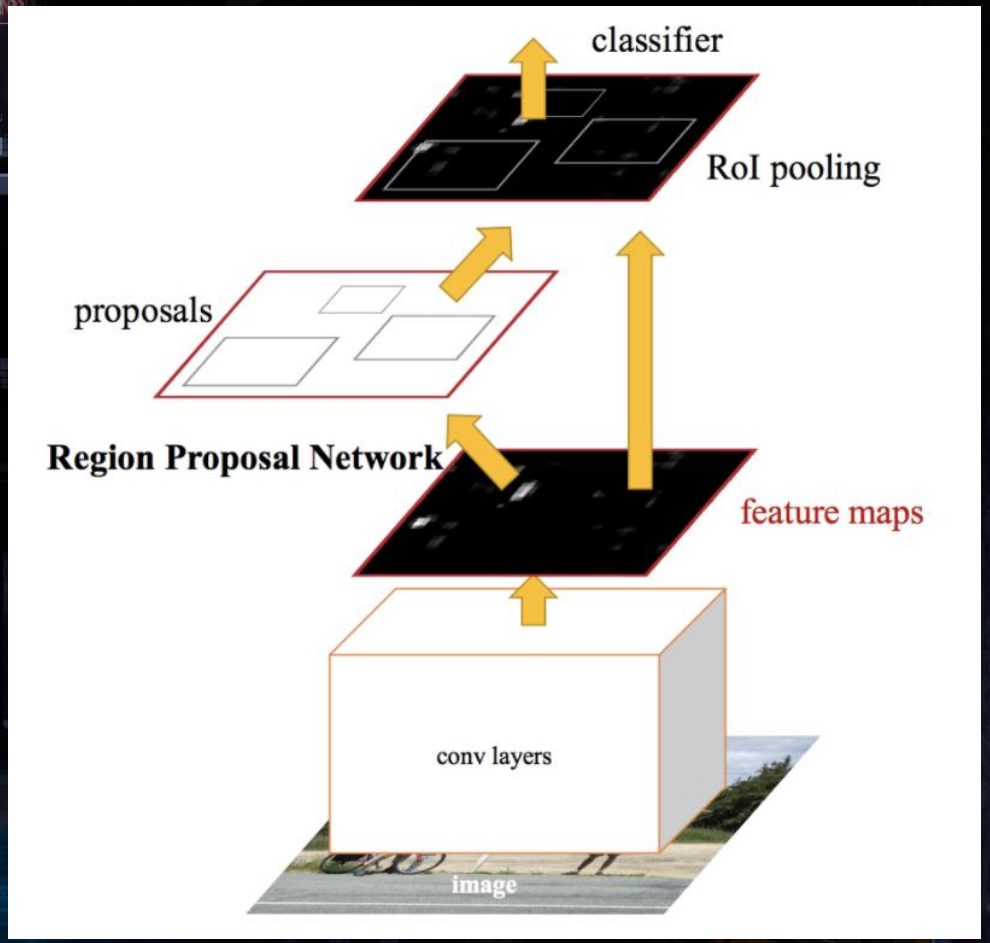


Source: *Yinghan, TowardsDataScience.com*
Javier, Tryolabs

V19.2: Object Detection IV

How does Faster R-CNN work?

- Replace selective search with Region Proposal Network
- Predicted proposals are reshaped to a fixed size using Region of Interest (ROI) Pooling
- Classifier predicts class of proposal and offset values for bounding boxes

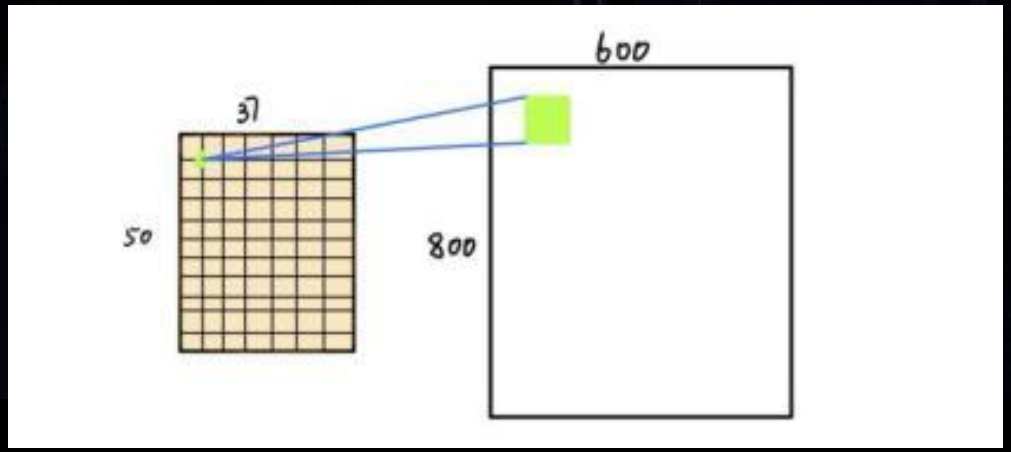


Source: *Yinghan, TowardsDataScience.com*
Javier, Tryolabs

V19.2: Object Detection IV

Regions are proposed using the Region Proposal Network (RPN)

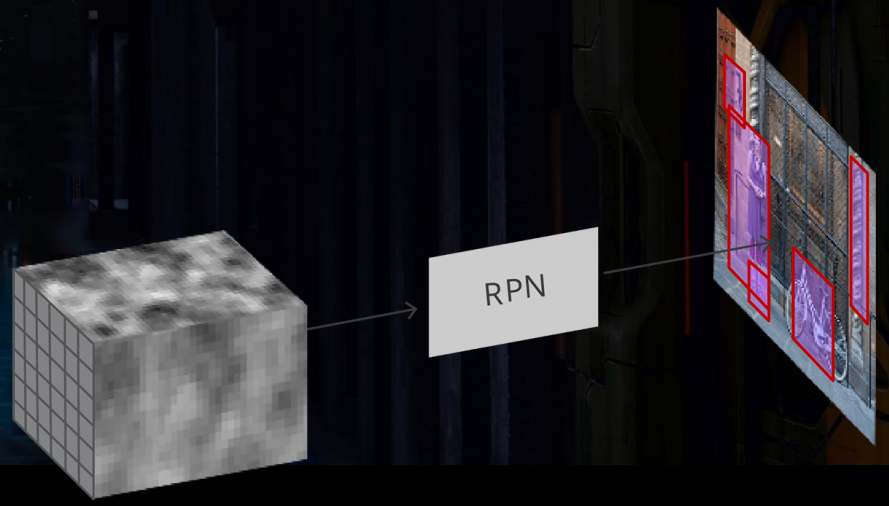
- Input image fed into backbone CNN
- Place a set of anchors on the input image for each location on the output feature map from the backbone network.
- Anchor may contain relevant object – which then drives the question of whether the anchor can be adjusted to better fit the object



Source: *Yinghan, TowardsDataScience.com*
Javier, Tryolabs

RPN Anchors

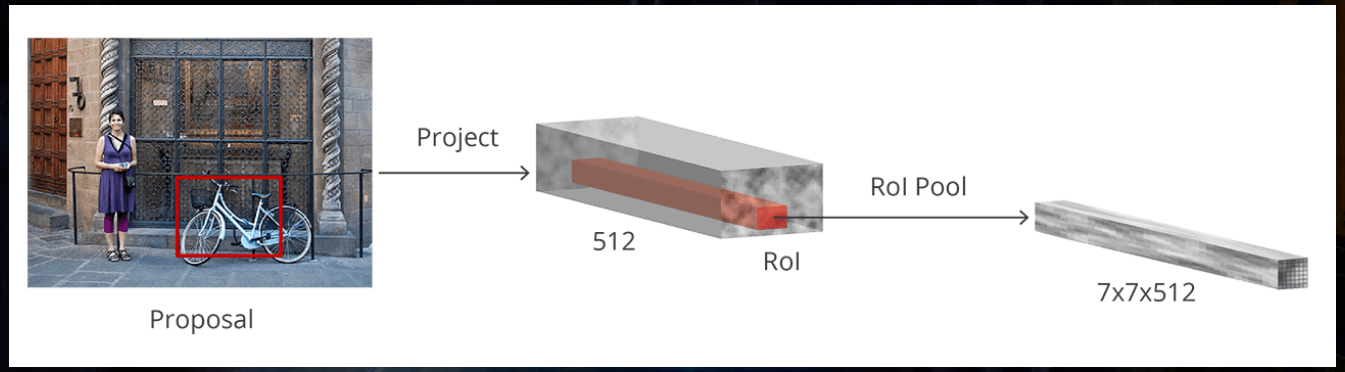
- Each anchor contains 2 key information
  - Objectness Score: Probability that an anchor contains an object (Background 0, Object 1)
  - Regression Scores: Bounding Box regression to adjust anchors to better fit the object (Delta Change in X_center, Delta Change in Y_center, Delta Change in Width and Delta Change in Height)



Source: *Yinghan, TowardsDataScience.com*
Javier, Tryolabs

ROI Pooling

- Reuse existing convolutional feature map by extracting fixed-size feature maps for each proposal.
- Common to crop the feature map by the proposal then resize to a fixed size 14*14*cd using usually bilinear interpolation then max pool with 2x2 kernel to get final 7x7*cd feature map. (cd refers to convolutional depth)



Source: *Yinghan, TowardsDataScience.com*
Javier, Tryolabs

V20: Object Detection V

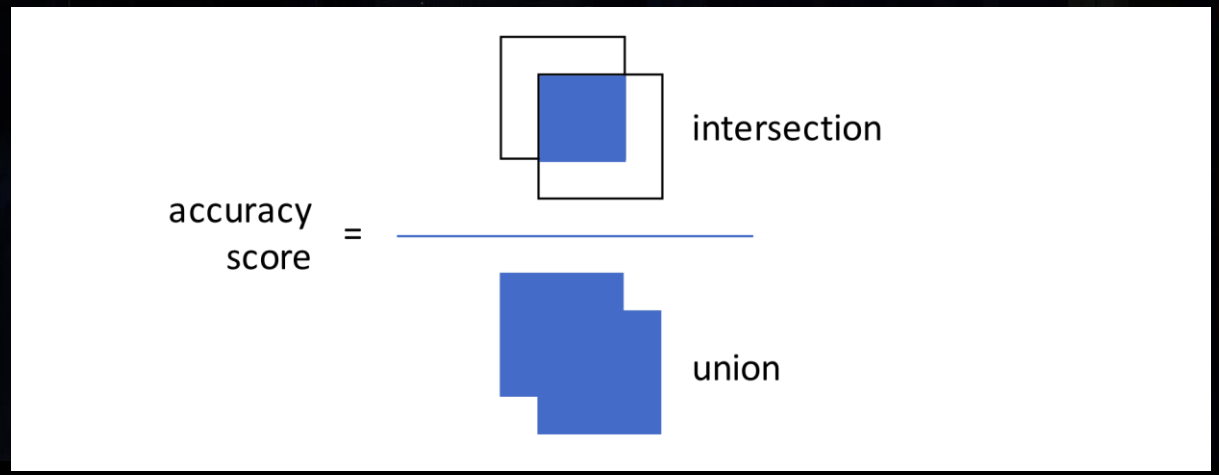
Agenda

- Evaluation
- Sample codes tutorial on PyTorch

V20: Object Detection V

Know what to see

- Determining how good the predictions are on the bounding boxes means determining how good the predicted bounding boxes and actual boxes overlap.
- Calculation: the intersection divided by the union, referred to as "*intersection over union*," or IoU. A perfect bounding box prediction will have an IoU of 1.

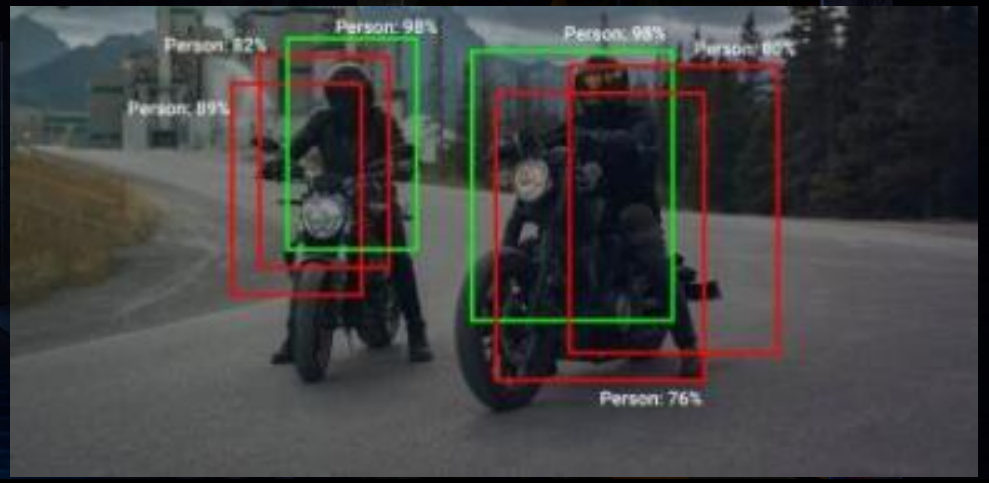


Source: *Jonathan Hui, TowardsDataScience.com*

V20: Object Detection V

Classification problem: Precision and Recall

- Precision refers to the percentage of the correctly predicted bounding boxes (IoU > 0.5) out of all bounding boxes predicted.
  - Precision falls when false positives increase
- Recall is the percentage of the correctly predicted bounding boxes (IoU > 0.5) out of all objects in the photo.
  - Recall falls when false negatives increase

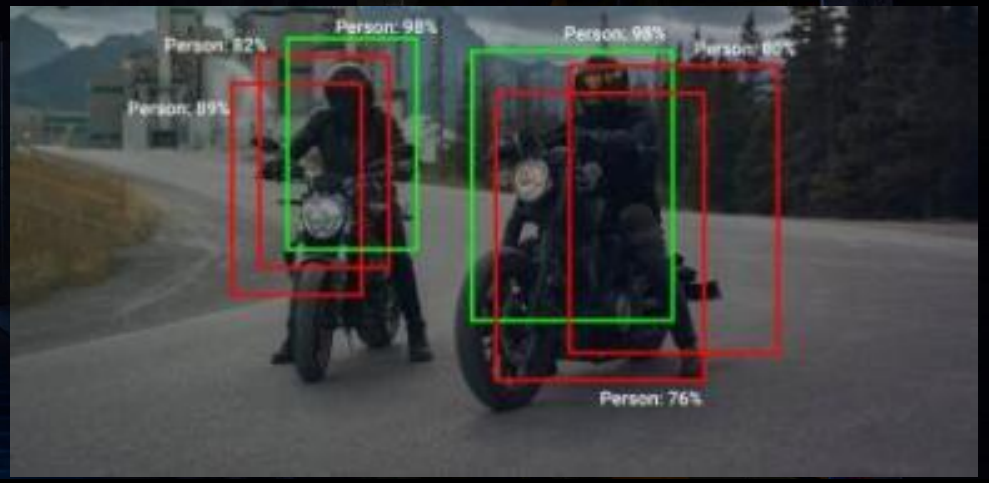


Source: *Jonathan Hui, TowardsDataScience.com*

V20: Object Detection V

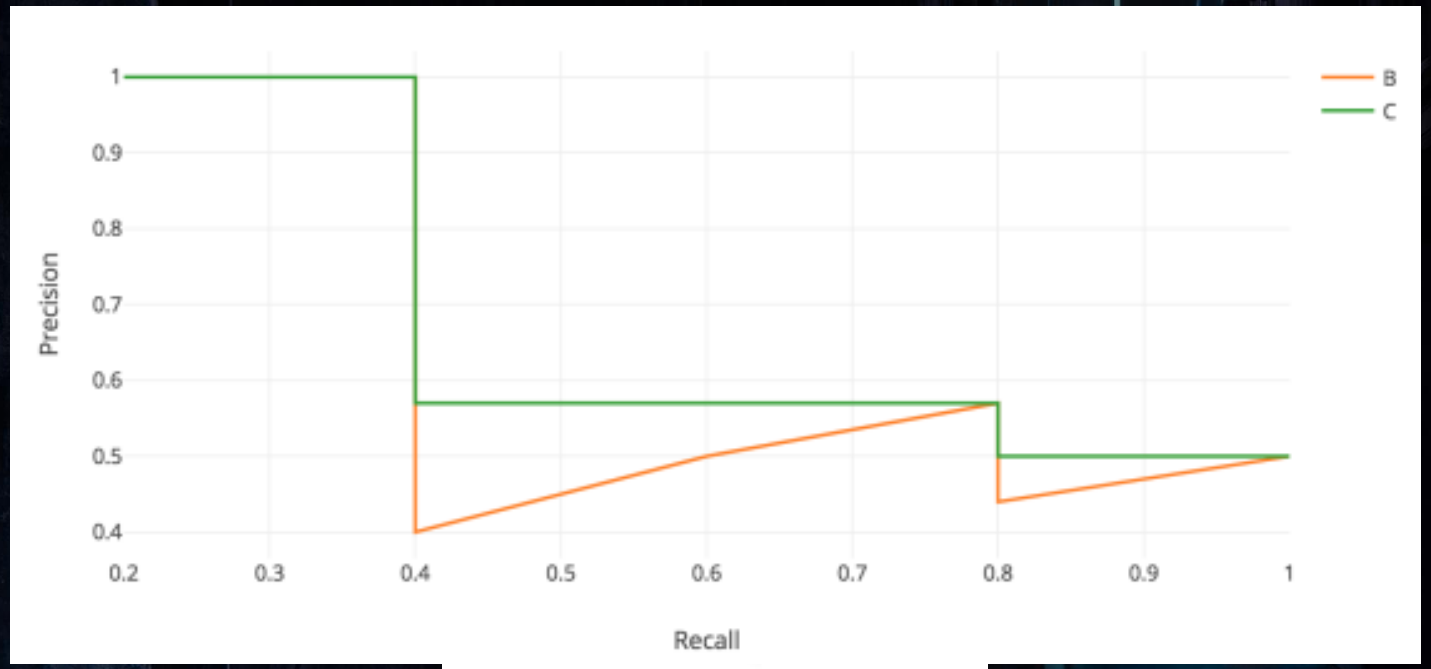
Mean Average Precision (mAP)

- **Mean Average Precision (mAP)** is usually used to compare between models
- Results vary given the stochastic nature of the algorithm, training and evaluation procedure.
- mAP is the average of the AP for all categories



Source: *Jonathan Hui, TowardsDataScience.com*

Average Precision (AP) Computation



$$\mathrm{AP} = \int_0^1 p(r)dr$$

- Smooth out the precision and recall curve
- AP is less sensitive to small variations in the ranking of detections
- Replace precision value for given recall with maximum precision value to the right of that recall level

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r})$$

Source: *Jonathan Hui, TowardsDataScience.com*

V20.2: Object Detection VI

Agenda

- Exploration into Post-Processing Steps i.e. Non-Maximum Suppression

V20.2: Object Detection VI

Resolving multiple detections of the same object

- One problem in Object Detection is that the algorithm may find multiple detections of the same object
- Non-maximum suppression is a way to make sure that the algorithm detects each object only once
- Filter the huge number of proposals generated by the Proposal Generation Network

DSTA
Defence Science & Technology Agency

V20.2: Object Detection VI

brainhack

3 inputs for Non-Maximum Suppression

To build a non-maximum suppression, there are 3 ingredients:

1. The list of proposal box generated (List A)
2. Corresponding confidence scores for each proposal box
3. An overlap threshold decided by the Scientists

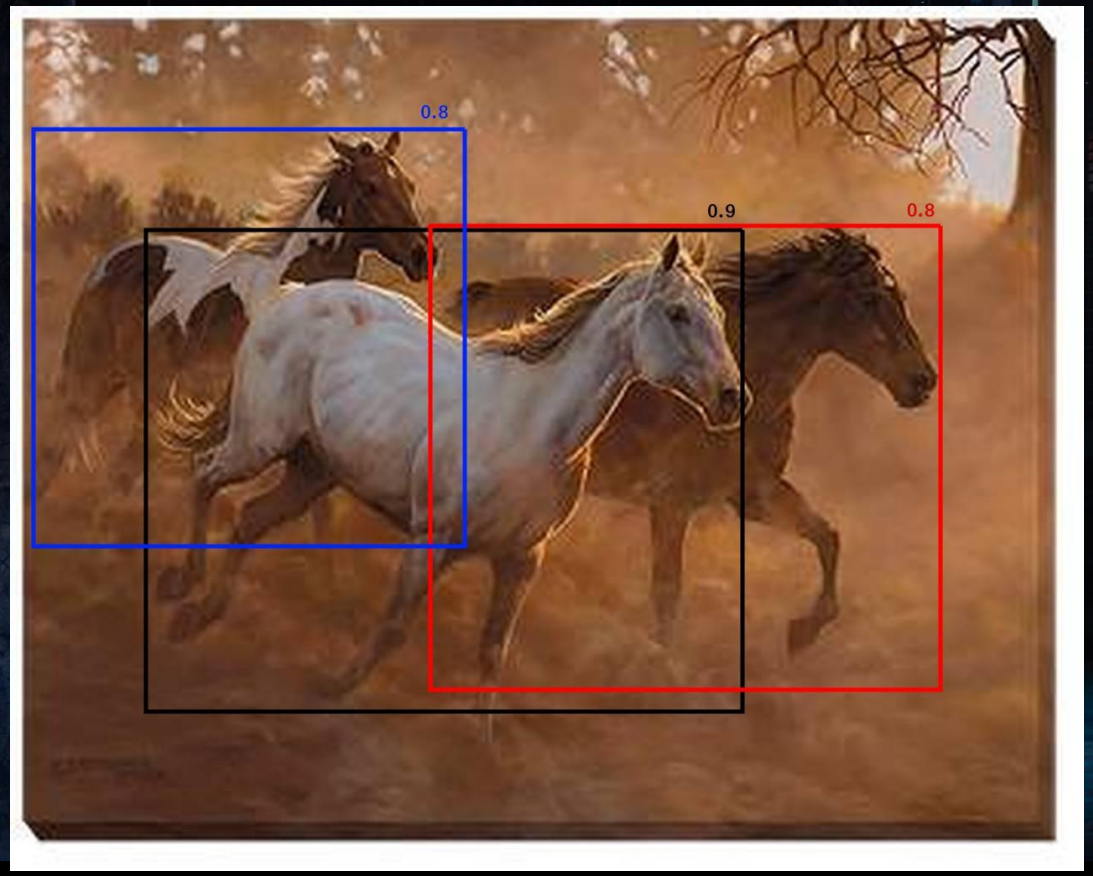
Process of the Algorithm

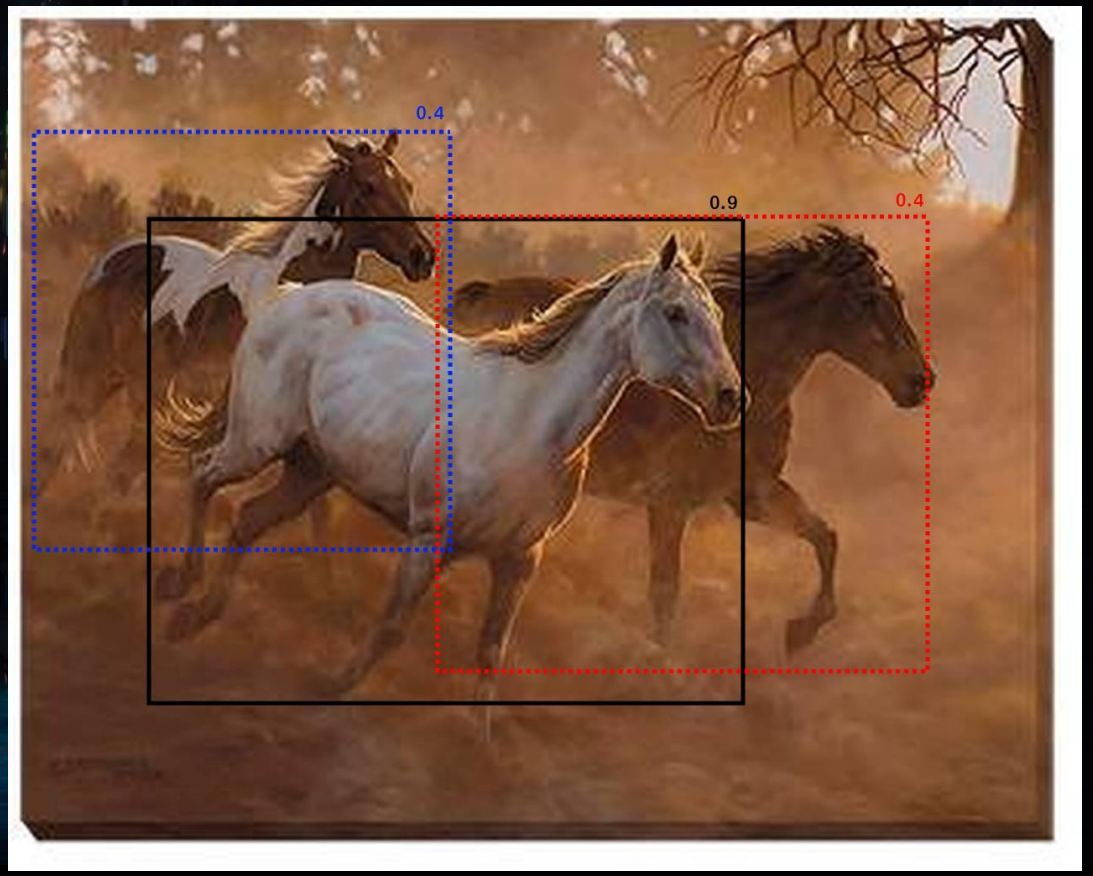
The method solves the problem via the following process:

1. Select the box with the highest confidence score, remove it from List A and add to a new final proposal list say B.
2. Calculate the IoU this box with all other proposed box
3. For each remaining boxes, if IoU > threshold set earlier, remove those boxes also from List A
4. Repeat Step 1 to 3 repetitively and final List B will contain all the boxes that has the highest probabilities
5. Stop when List A has no more boxes

V20.2: Object Detection VI

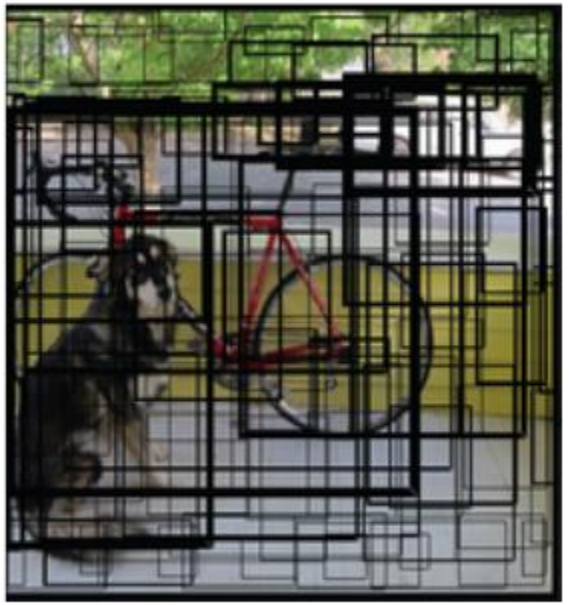
Before Suppression to After Suppression



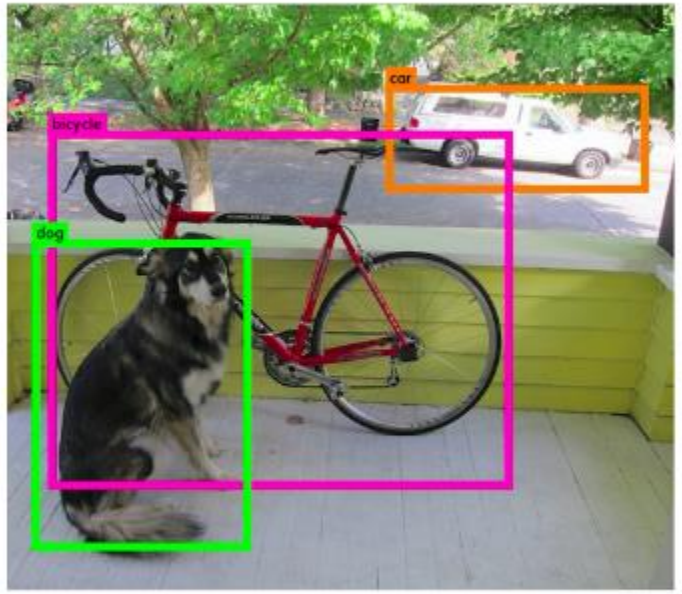


V20.2: Object Detection VI

Multiple Bounding Boxes



Multiple Bounding Boxes



Final Bounding Boxes

Independently carry out non-maximum suppression for each of the classes.

V21: Tips on Model Training and Improvements

Agenda

- Data-Centric Approaches
- Model-Centric Approaches

V21: Tips on Model Training and Improvements

Data Centric Approaches

Data cleaning is essential

- Check through wrong labels
- Check through noisy labels (Similarity Comparison)

Impact of noisy labels depend on the size of the data

	Steel defect detection
Baseline	76.2%
Model-centric	+0% (76.2%)
Data-centric	+16.9% (93.1%)

Source: *Andrew Ng, 2021, DeepLearning.AI*

V21: Tips on Model Training and Improvements

Data Centric Approaches

Subsampling

- Time-efficient to train
- Prevent over-learning
- Helps to perform hyperparameter tuning with small sample

Cross-Validation

- Estimate the skill of the model on unseen data
- Use a small sample for each parallel run of the model to test
- Helps to perform hyperparameter tuning with small sample

For both training and testing

V21: Tips on Model Training and Improvements

Data Centric Approaches

Systematic data-centric approaches

- Improve data through iteration

E.g. using 2 independent labelers to label a sample (More for real world)

E.g. Complementary labeling

V21: Tips on Model Training and Improvements

Data Centric Approaches

Shift the mindset from Big Data to Good Data

Perform error analysis to identify the types of data the algorithm does poorly on (e.g. which images are predicted wrongly with their labels)

Either get more of that data via data augmentation, data generation or data collection (features – x)

Else give a consistent change to the labels (target – y)

V21: Tips on Model Training and Improvements

Data Centric Approaches

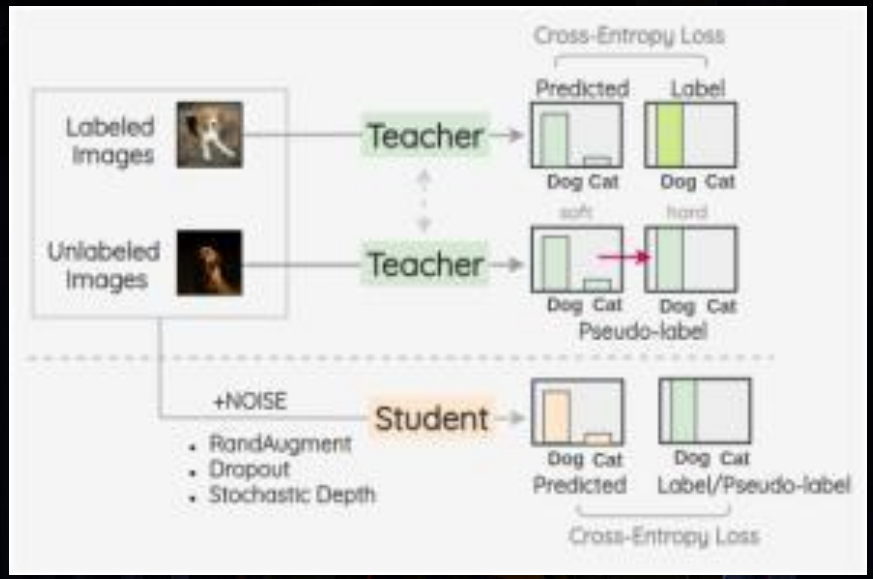
	Steel defect detection	Solar panel	Surface inspection
Baseline	76.2%	75.68%	85.05%
Model-centric	+0% (76.2%)	+0.04% (75.72%)	+0.00% (85.05%)
Data-centric	+16.9% (93.1%)	+3.06% (78.74%)	+0.4% (85.45%)

V21: Tips on Model Training and Improvements

Model Centric Approaches

Pipeline changes

1. Train a supervised classifier on the labeled dataset.
2. Assign a class to another set of unlabeled sample using the previous model (pseudo labeling) to construct a new *pseudo* labeled dataset.
3. Train a bigger (i.e., larger architecture) model with strong data augmentation on the combined labeled and *pseudo* labeled dataset. (Increase generalization)



V21: Tips on Model Training and Improvements

Model Centric Approaches

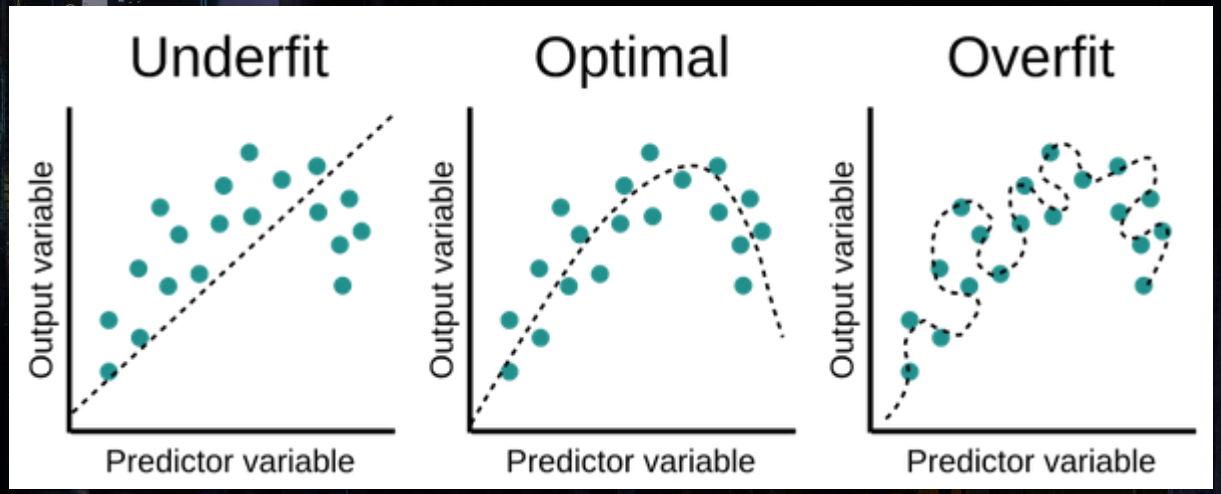
Hyperparameter Tuning

- Learning rate (Dynamic)
- Gradient Descent
- Striding, Padding, Kernel Size etc.
- MaxPooling, AveragePooling, Dropout

V21: Tips on Model Training and Improvements

Model Centric Approaches

Be careful of overfitting



V22: Additional Resources and Links

Agenda

- Sources for additional resources and links

V22: Additional Resources and Links

Resource Links

NumPy - http://docs.scipy.org/doc/numpy/reference/routines.math.html

PyTorch - https://PyTorch.org/tutorials/

TensorFlow - https://www.tensorflow.org/tutorials

Neural Networks - https://towardsdatascience.com/understanding-neural-networks-19020b758230

Convolutional Neural Networks - https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

Neural Networks Playground - https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/playground-exercises

V22: Additional Resources and Links

Resource Links

Image Processing - https://towardsdatascience.com/image-pre-processing-c1aec0be3edf

Object Detection - https://machinelearningmastery.com/object-recognition-with-deep-learning/

COCO dataset - https://towardsdatascience.com/master-the-coco-dataset-for-semantic-image-segmentation-part-1-of-2-732712631047

Hyperparameter Tuning - https://towardsdatascience.com/simple-guide-to-hyperparameter-tuning-in-neural-networks-3fe03dad8594

Public Datasets - https://PyTorch.org/vision/stable/datasets.html

Linearity of problems - https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

V23: Object Detection Xtra Notebook I

Agenda

- More examples of object detection programs in PyTorch

V1: Introduction to Speech Classification

Agenda

- Appreciate Speech Classification
- Appreciate Speech Classification Use-Cases
- Appreciate Procedures of Speech Classification

V1: Introduction to Speech Classification

- Speech Classification refers to a set of tasks or problems of getting a program to automatically classify input utterance or audio segment into categories.

- Real speech and audio recognition systems are much more complex but like classifying images, speech classification tries to classify a one second audio clip as "down", "go", "left", "no", "right", "stop", "up" and "yes".

- Using CNN, instead of directly using the sound file as an amplitude vs time signal, the audio signal will be converted into an image.

V1: Introduction to Speech Classification

- Speech Command Classification (multi-class)
  - Classifying an input audio pattern into a discrete set of classes
  - Objective of command recognition models: small and efficient so that they can be deployed onto low-power sensors and remain active for long durations of time.
- Voice Activity Detection (binary or multi-class)
  - Predict which parts of input audio contain speech versus background noise
- Audio Sentiment Classification (typically multi-class), etc.

V1: Introduction to Speech Classification

- Automatic speech recognition (ASR)
- Digital signal processing
- Classifying music clips to identify the genre of the music
- Classifying short utterances by a set of speakers to identify the speaker based on the voice.
- Tagging
- Detect illness from the tone of an individual
- Audio Generation
- Building blocks of every intelligent conversational tool (Conversational AI)

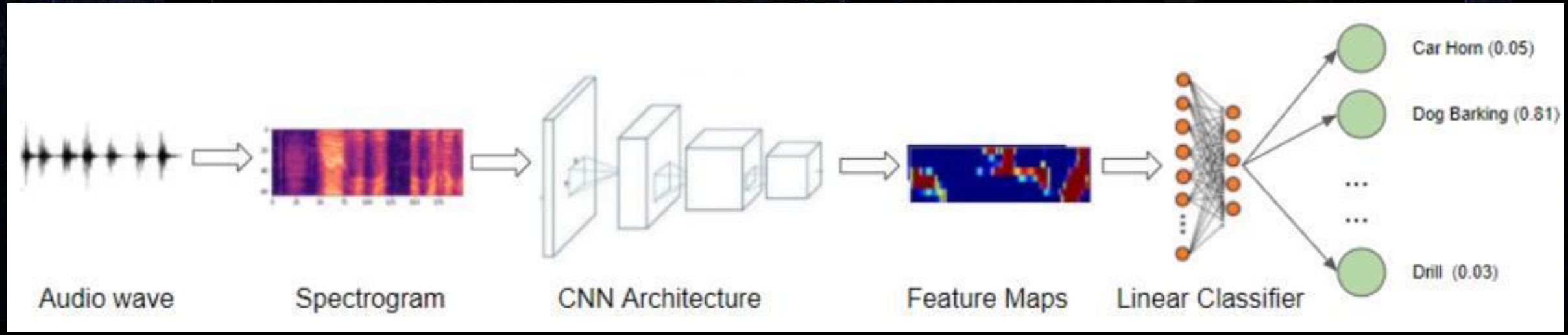
V1: Introduction to Speech Classification

- Voice Assistants (Alexa):

1. First, the assistant must convert the speech to text.
2. The text is run through a natural language processing (NLP) step, which turns the words into numeric data.
3. Finally, there's a classification of the *utterance* - what people say to the *intent* - what they want the voice assistant to do.

V1: Introduction to Speech Classification

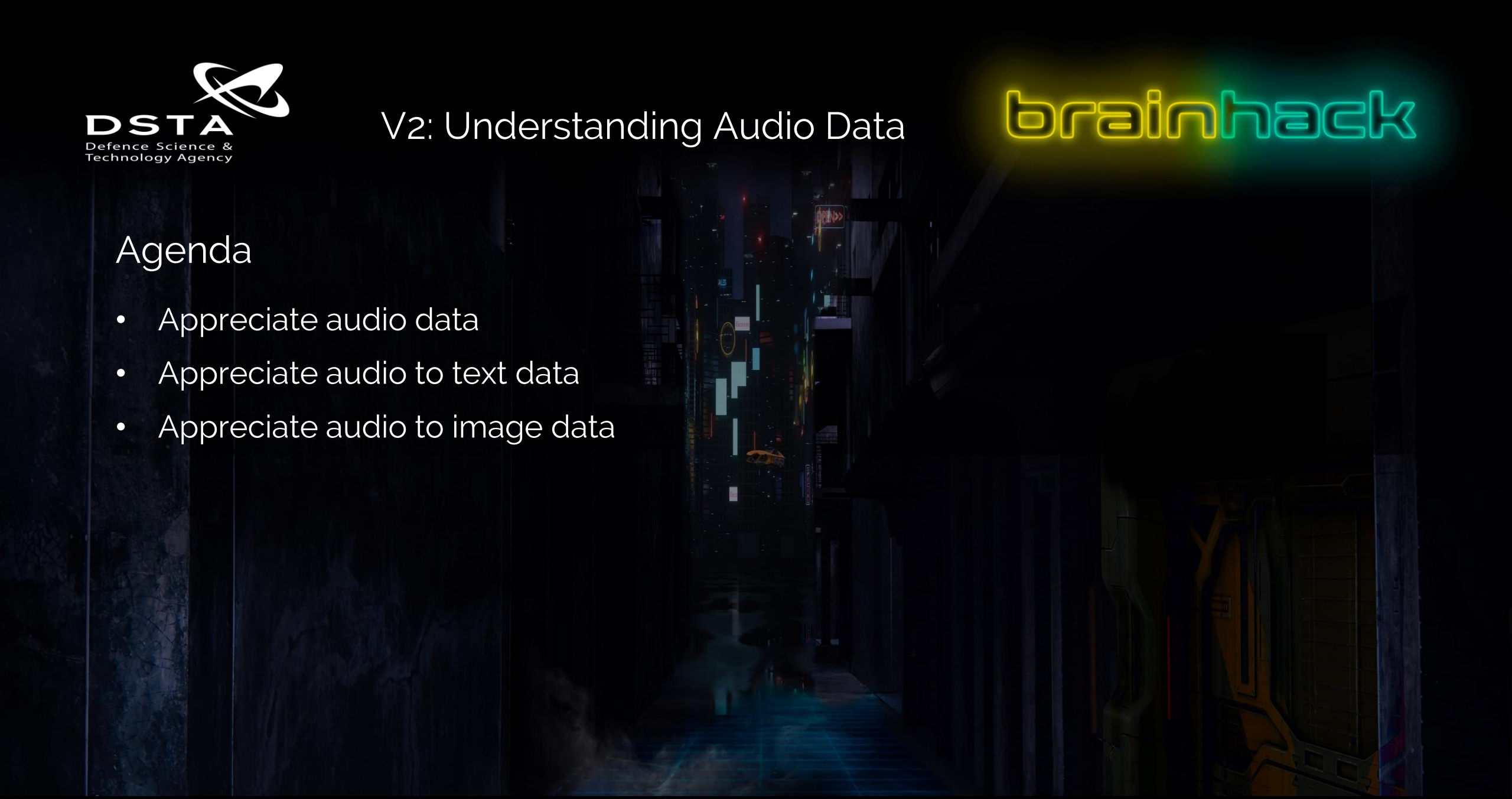
1. Start with sound files
2. Convert them into spectrograms
3. Use signal processing to extract the most important information from the audio file
4. Input them into a model
5. Produce predictions about the class to which the sound belongs.



V2: Understanding Audio Data

Agenda

- Appreciate audio data
- Appreciate audio to text data
- Appreciate audio to image data

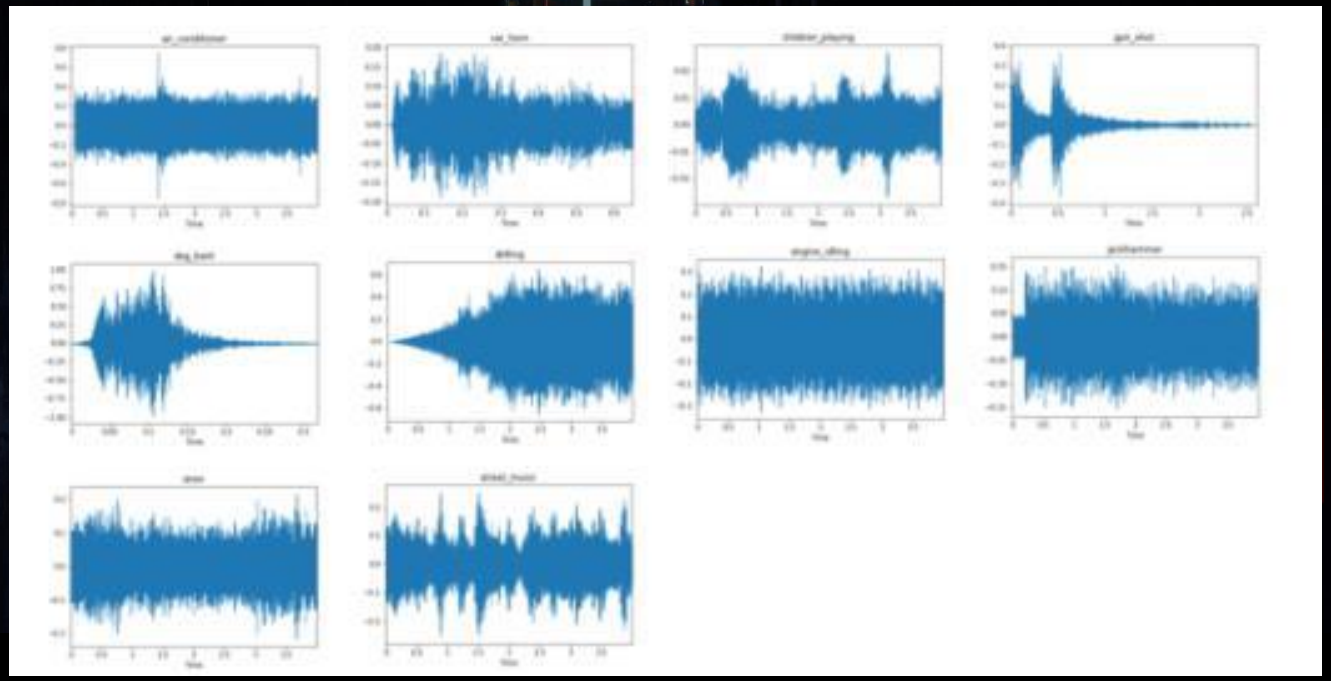


V2: Understanding Audio Data

1. Sample Rate - for audio, a microphone is used to capture the sound and then its converted from analog sound to digital sound by sampling at consistent intervals of time.
    1. The higher the sample rate the higher the quality of the sound.
    2. The average sound sample rate is 48 kHz or 48,000 samples per second.
2. When the audio is sampled its sampling the frequency or the pitch of the sound and the amplitude or how loud the audio is.
3. This signal can be represented as a waveform which is the signal representation over time in a graphical format comparing the relationship between the sample rate and frequency.

V2: Understanding Audio Data

- Audio data is generally visualized in the form of waveplots.
- A waveplot is a graph consisting of two axes. The X axis represents the time, the Y axis represents the displacement from the mean position which refers the amplitude.
- Librosa package from python can be used to illustrate the audio files.



DSTA
Defence Science & Technology Agency

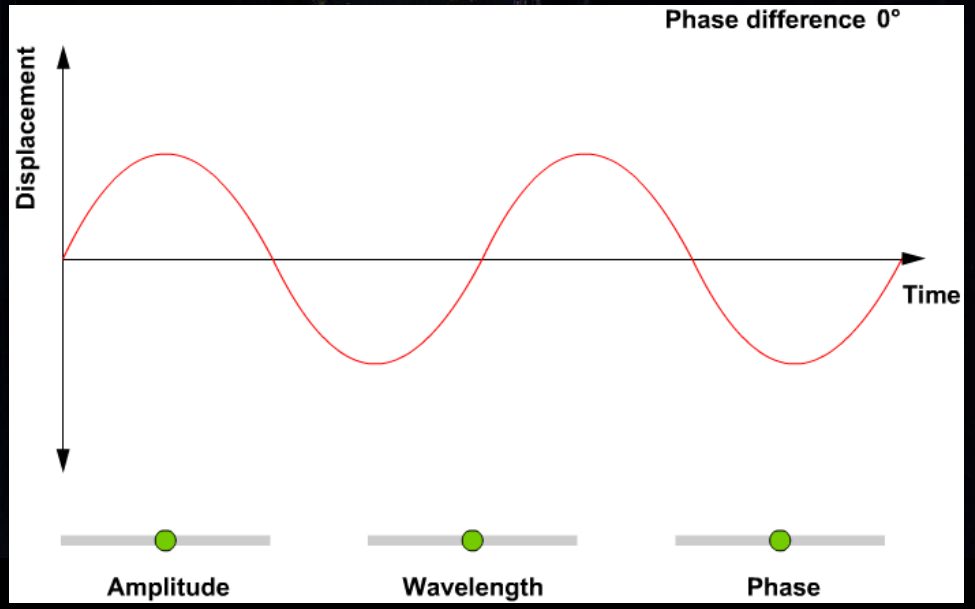
V2: Understanding Audio Data

brainhack

1. Audio data can be converted to text data for NLP and solve text classification
2. Audio data can generate spectrograms for CV and solve as image classification
3. Audio data can extract many other features such as tone, pitch, sound, noises for other creative means to classify them

V2: Understanding Audio Data

- Audio data is essentially a sequence data of sound waves given a sample rate
- Converting audio data to any other forms for classification would remove that sequencing effect and result in static analysis such as using CNN to predict the classes from the spectrograms
- Keeping sequence intact, RNNs would be a good means for modeling audio classification.



DSTA
Defence Science & Technology Agency

V3: Audio Data to Tensors

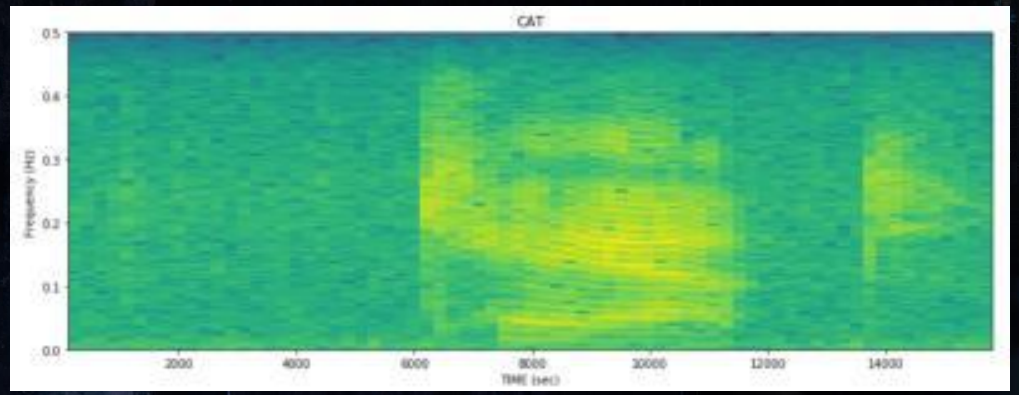
brainhack

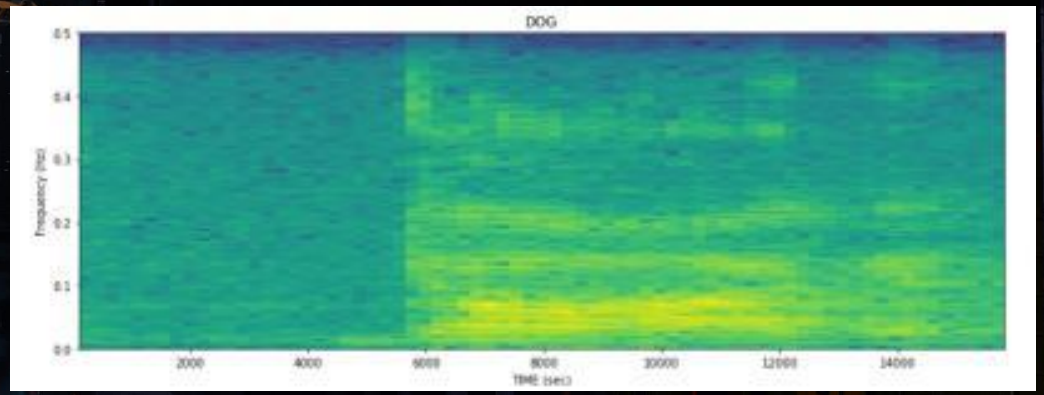
Agenda

- Appreciate audio data conversion to spectrogram
- Appreciate audio data conversion to tensors
- Sample codes

V3: Audio Data to Tensors

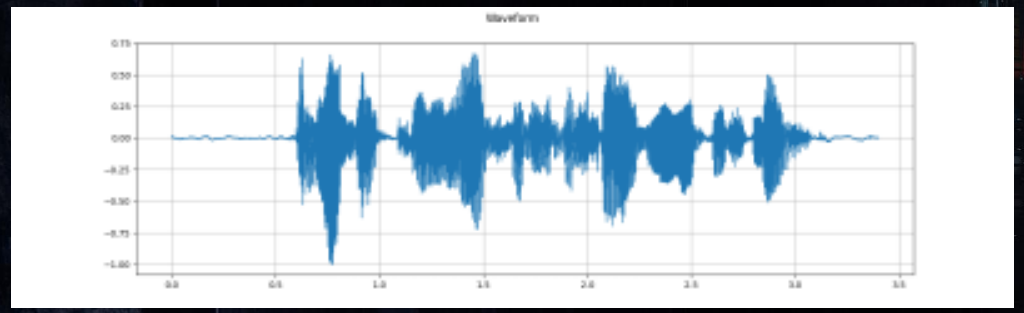
- Spectrogram is just another way to represent wav files
- CAT and DOG are clearly different frequency and wavelength
- Intensity shown by the shades of the color as representing signal strength (loudness)

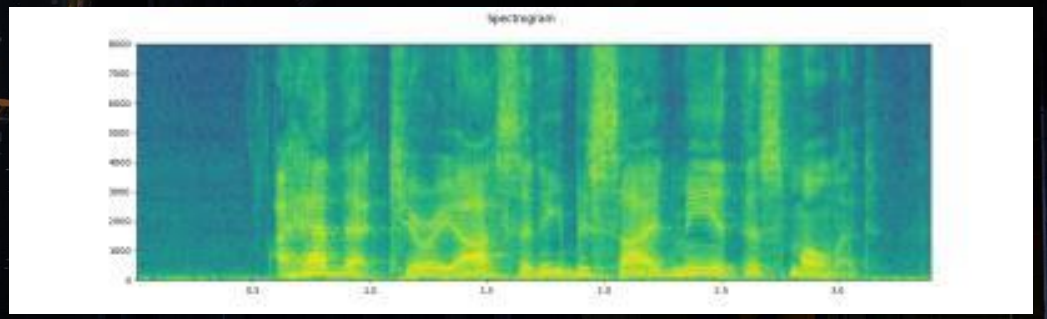




V3: Audio Data to Tensors

```
waveform, sample_rate = torchaudio.load(SAMPLE_WAV_SPEECH_PATH)
print_stats(waveform, sample_rate=sample_rate)
plot_waveform(waveform, sample_rate)
plot_specgram(waveform, sample_rate) play_audio(waveform, sample_rate)
```

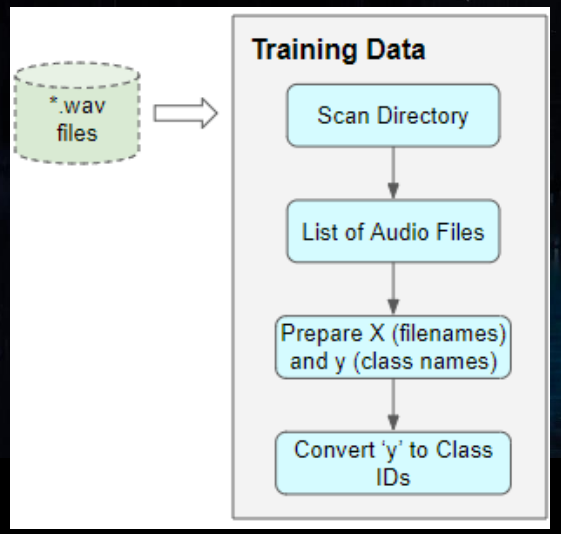




```
Shape: (1, 54400)
Dtype: torch.float32
 - Max:      0.668
 - Min:     -1.000
 - Mean:     0.000
 - Std Dev:  0.122
tensor([[0.0183, 0.0180, 0.0180,  ..., 0.0018, 0.0019, 0.0032]])
<IPython.lib.display.Audio object>
```

V3: Audio Data to Tensors

- Hence, simply read and load the audio file in .wav format using **`torchaudio`**
- **`torchaudio`** loads audio file into a **`torch.Tensor`** object
- TorchAudio has abstracted the load functions for different audio back ends so there is no need to worry about the implementation.
- Otherwise, **`librosa`** work as well (returns a numpy)
- Do note that some pre-processing of audio may be needed such as defining the transforms function



Scan audio directory when metadata not available

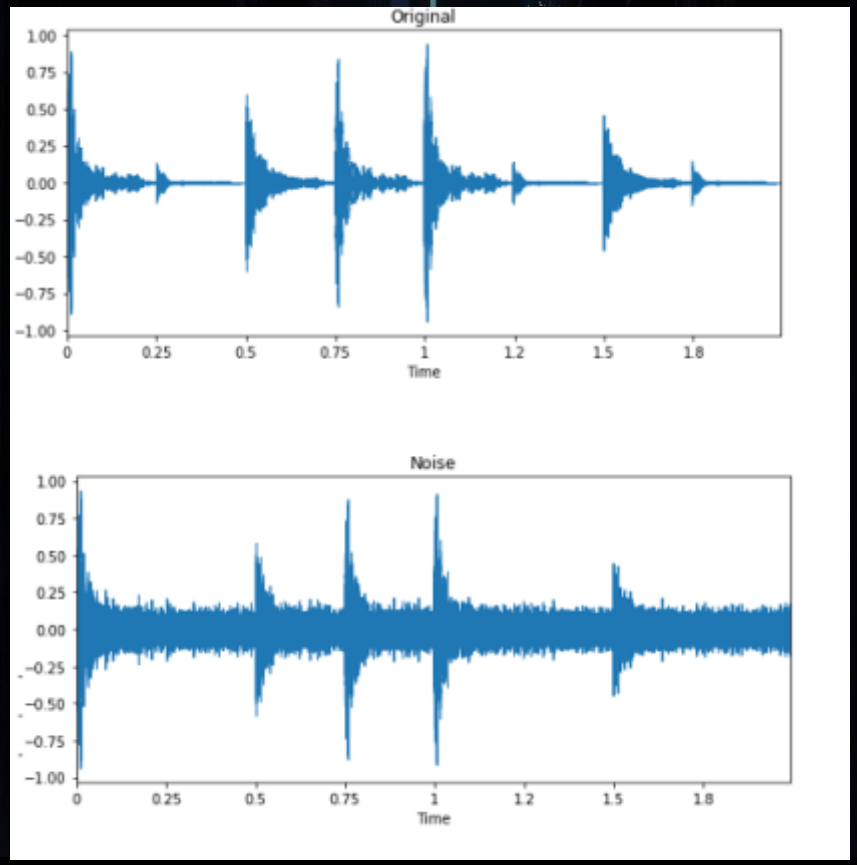
V4: Data Augmentation

Agenda

- Appreciate audio data augmentation benefits
- Appreciate audio data augmentation techniques
- Sample codes

V4: Data Augmentation

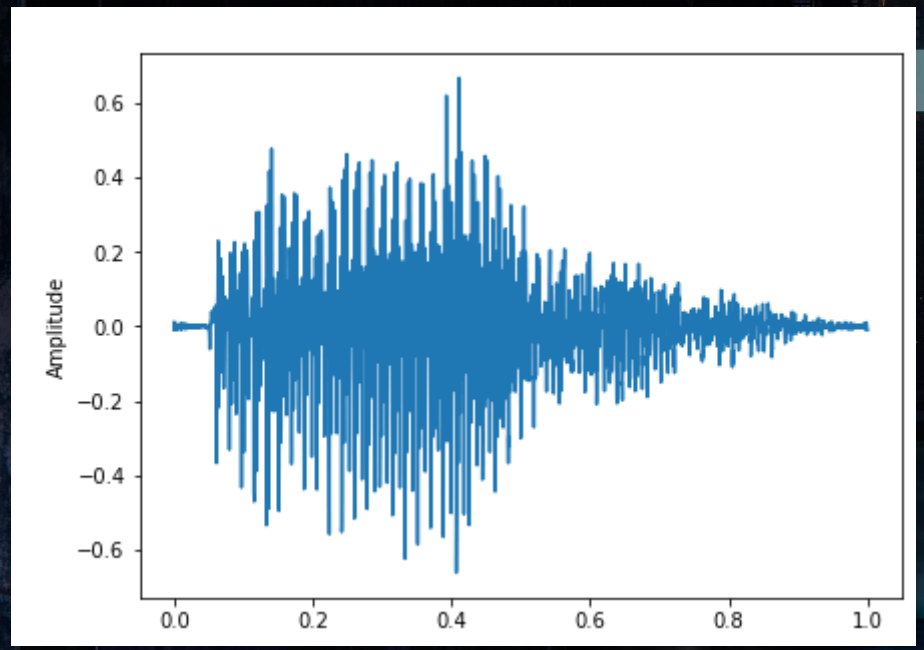
- Noise Injection – randomly add value into tensor array



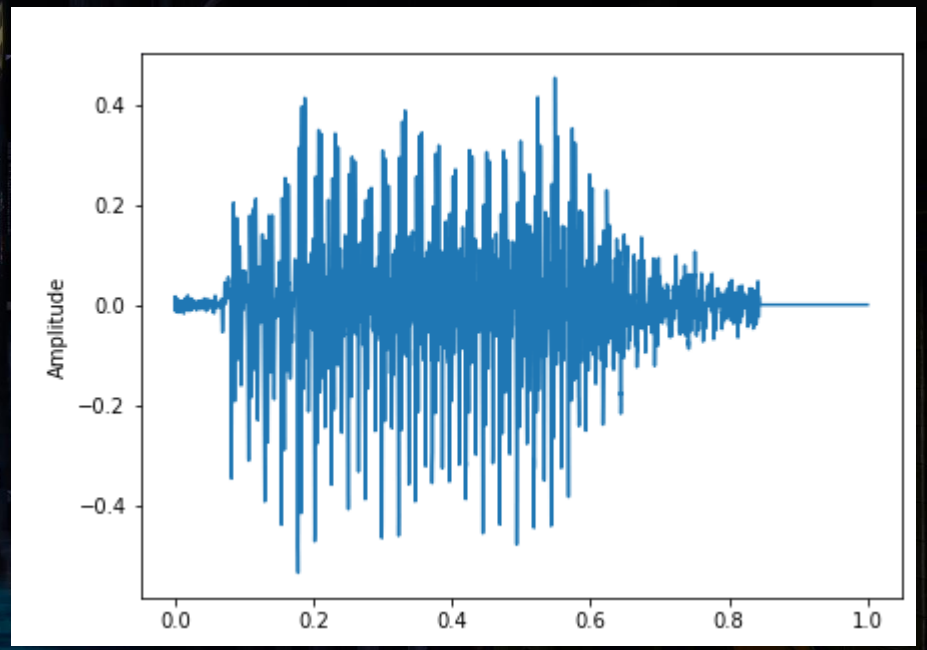
V4: Data Augmentation

- Pitch Shift – shift audio by a random amount

Original



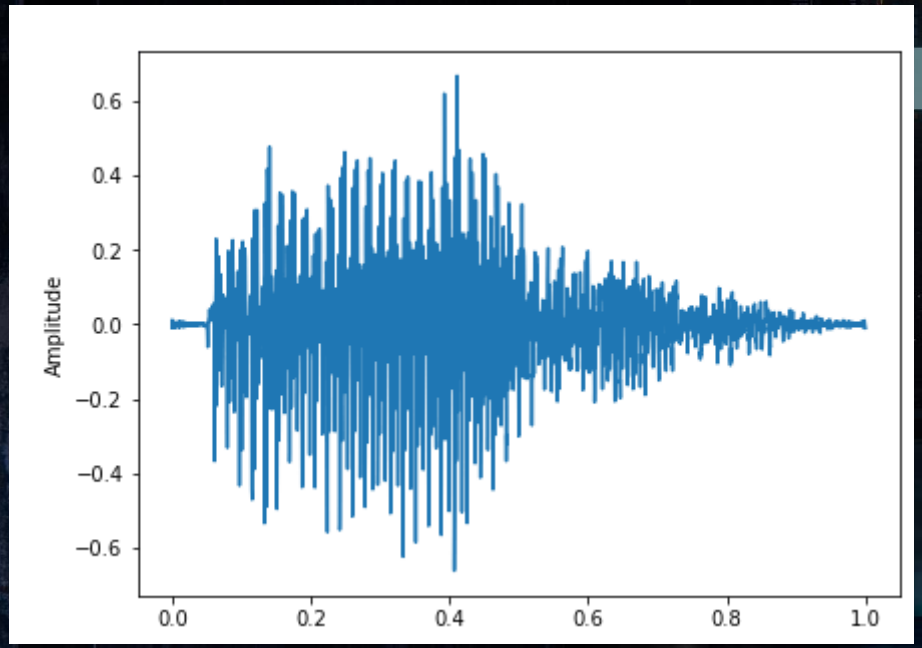
Changed



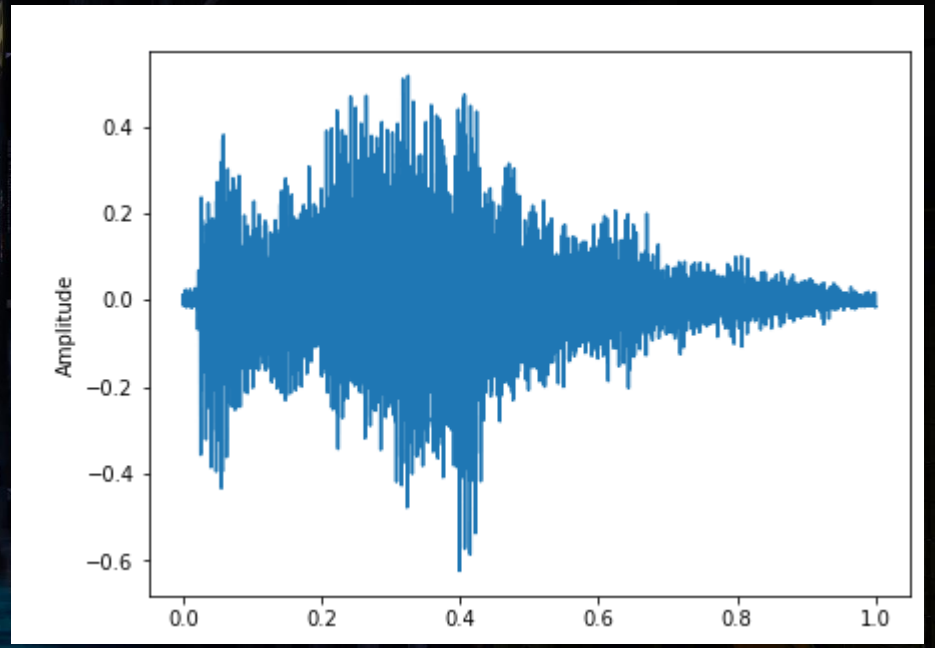
V4: Data Augmentation

- Speed Change – stretches time series by a fixed rate

Original

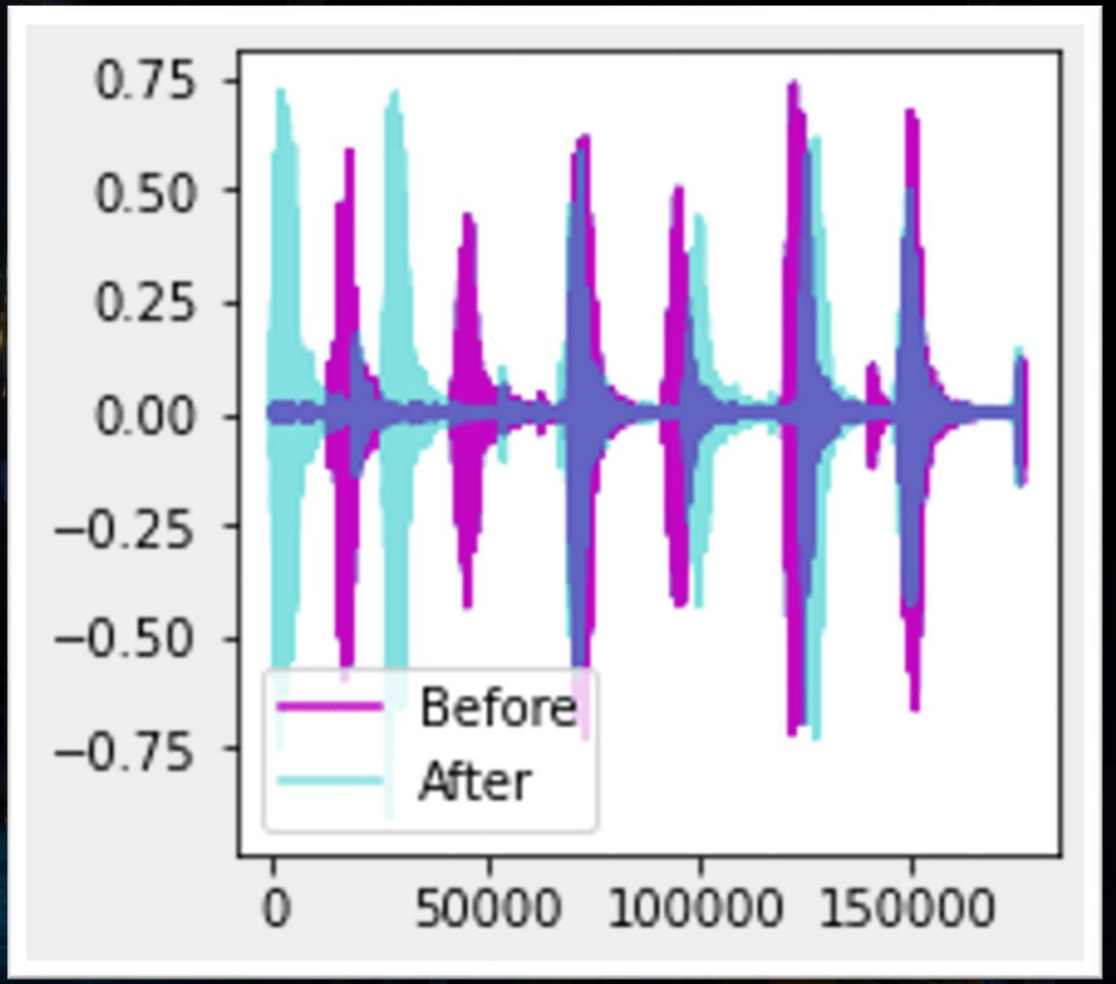


Changed



V4: Data Augmentation

- Time Shift – shift audio by a random amount
  - Create silence for the first few seconds (fast forward) by shifting audio to left
  - Create silence for last few seconds (back forward) by shifting audio right



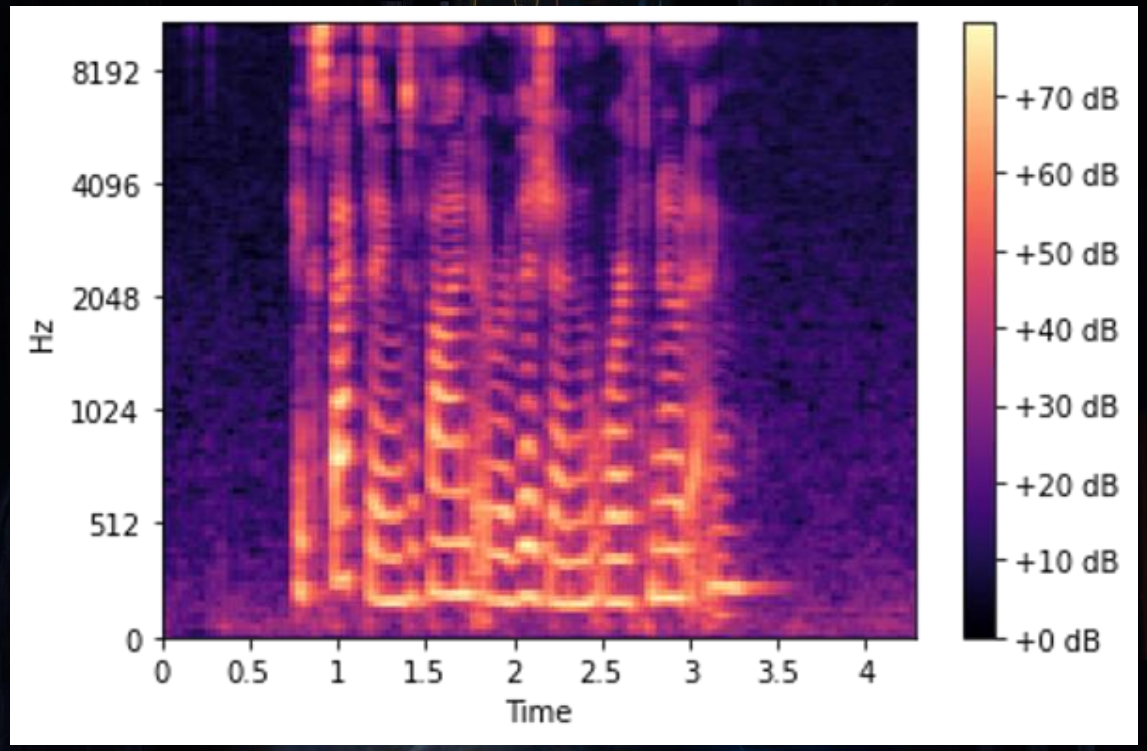
V5: Feature Extraction & Augmentation

Agenda

- Appreciate feature extraction
- Appreciate feature augmentation
- Sample codes

V5: Feature Extraction & Augmentation

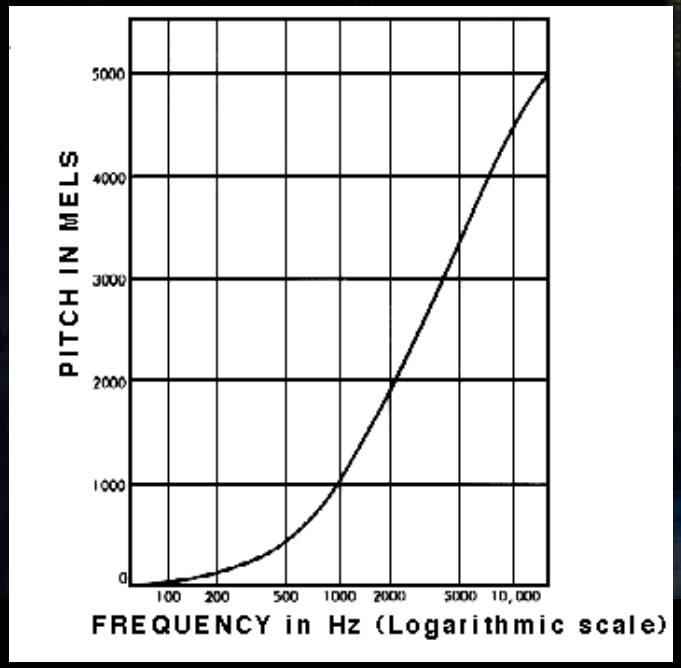
- Other than Spectrogram, there are other means to extract features from audio data.
- Mel Spectrogram is an alternative that is commonly seen to be better.



V5: Feature Extraction & Augmentation

- People hear frequencies in sound as “pitch”
- Higher pitch means higher frequency, vice versa
- Pitch is not perceived linearly. A 200Hz is not heard the same as twice of 100Hz.
- Humans hear frequency on a logarithmic scale rather than a linear one

Mel Scale

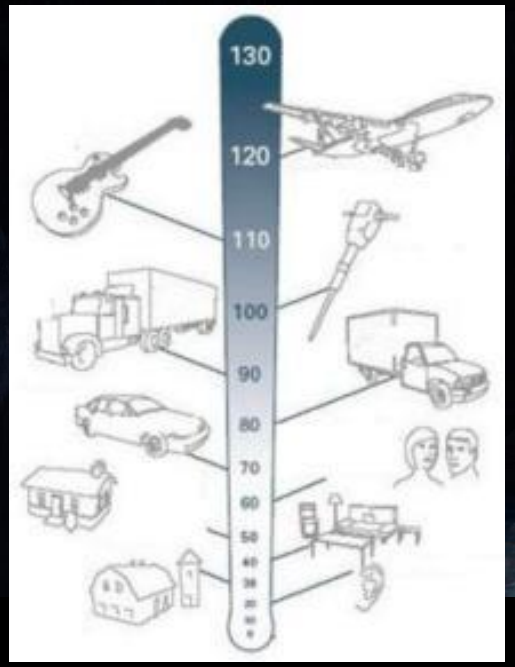


- 100Hz versus 200Hz (appear pitch higher)
- 1000Hz versus 1100Hz
- 10000Hz versus 10100Hz

V5: Feature Extraction & Augmentation

- People hear amplitudes in sound as “loudness”
- Larger amplitudes means louder, vice versa
- Loudness is not perceived linearly. A 20Hz is not heard the same as twice of 10Hz.
- Loudness deal with logarithmic scale

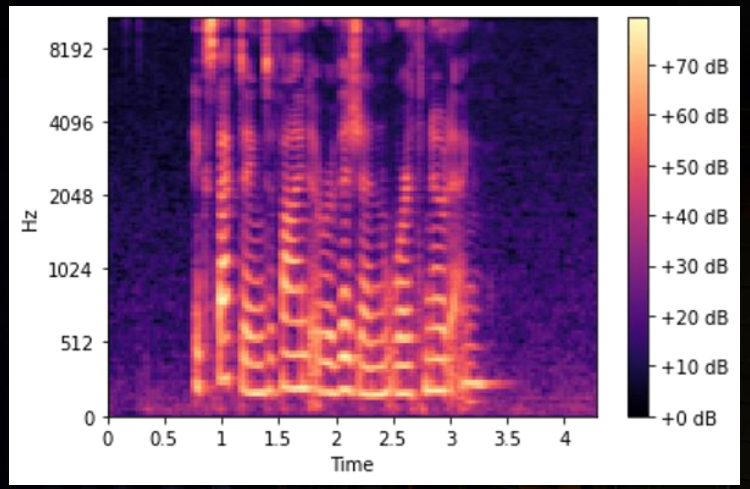
Decibel Scale



- 10dB is 10 times louder than 0dB
- 20dB is 100 times louder than 0dB
- 30dB is 1000 times louder than 0dB

V5: Feature Extraction & Augmentation

- Use a logarithmic scale via the Mel Scale and the Decibel Scale when dealing with Frequencies and Amplitudes in audio data respectively.
- Mel Spectrogram uses the Mel Scale instead of Frequency on the y-axis.
- It uses the Decibel Scale instead of Amplitude to indicate colors.




```
# use the mel-scale instead of raw frequency
sgram_mag, _ = librosa.magphase(sgram)
mel_scale_sgram = librosa.feature.melspectrogram(S=sgram_mag, sr=sample_rate)
librosa.display.specshow(mel_scale_sgram)
```

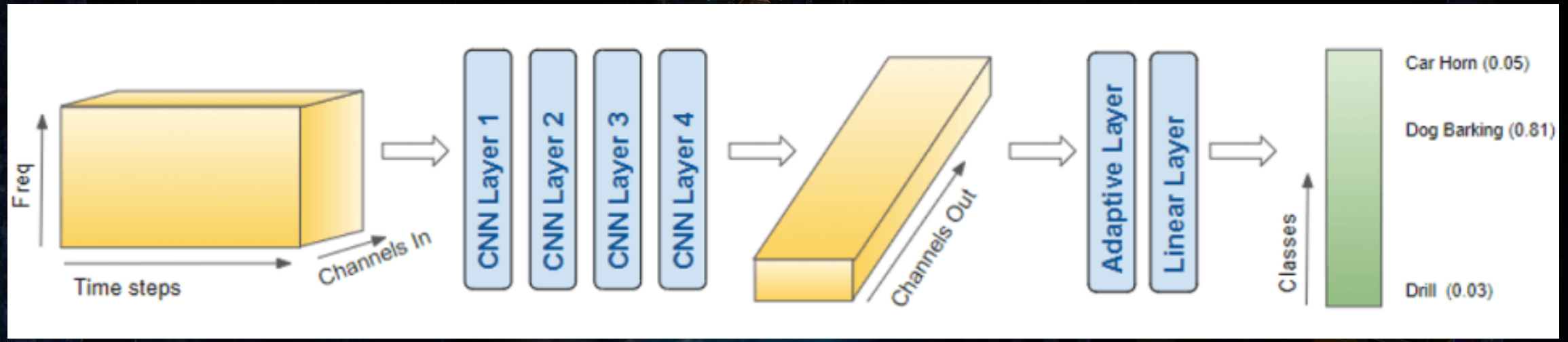
V6: CNN for Speech Classification I

Agenda

- Appreciate using CNN for speech classification
- Appreciate CNN pipeline for speech classification

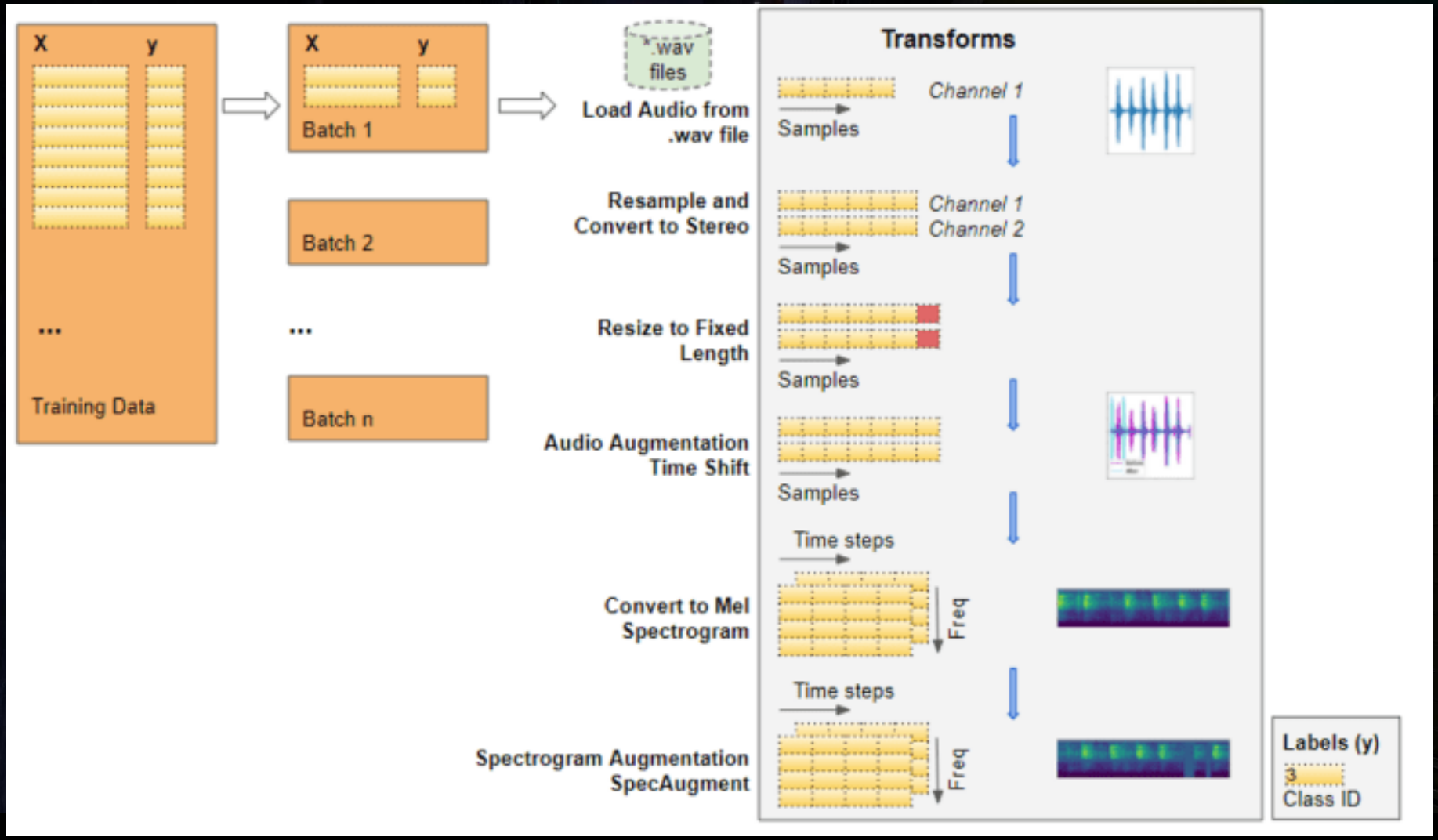
V6: CNN for Speech Classification I

- Mel spectrograms creation is the conversion of the audio data to image-like data
- Henceforth, the approach for speech classification would be similar to image classification
- Example below has 4 convolutional layers to generate feature maps
- Then data reshaped into format for linear classifier layer to predict 10 classes



Suggested Model

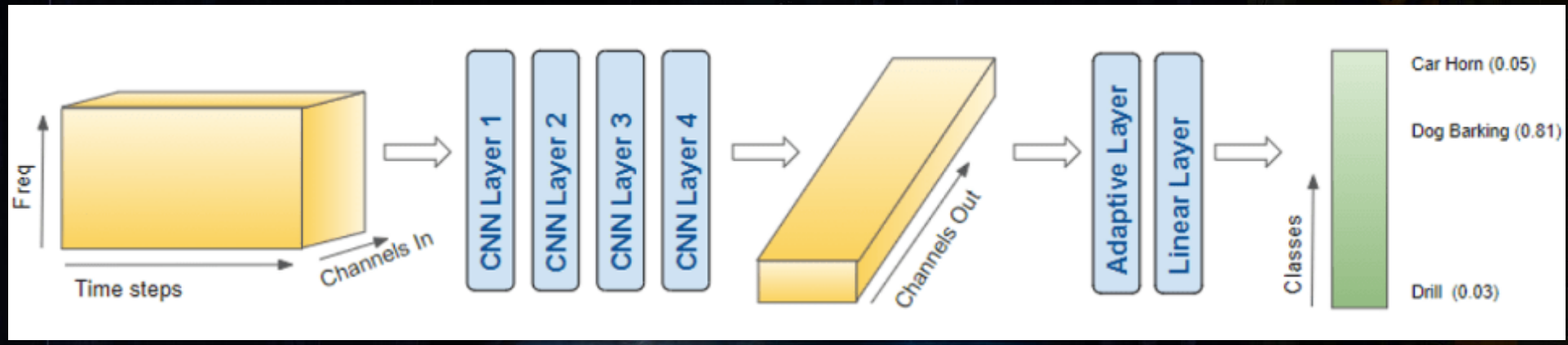
V6: CNN for Speech Classification I



Source: *TowardsDataScience*

V6: CNN for Speech Classification I

- A batch of images is input to the model with shape `(batch_sz, num_channels, Mel freq_bands, time_steps)` ie. `(16, 2, 64, 344)`.
- Each CNN layer applies its filters to step up the image depth ie. number of channels.
- The image width and height are reduced as the kernels and strides are applied.
- This gets pooled and flattened to a shape of `(16, 64)` and then input to the Linear layer.
- The Linear layer outputs one prediction score per class ie. `(16, X)` where `X` is the number of classes



Suggested Model

V7: CNN for Speech Classification II

Agenda

- Appreciate improving classification with pre-trained models
- Appreciate improving classification with augmentation
- Sample codes

V7: CNN for Speech Classification II

Model-Centric

- Torchvision built-in ResNet model
- ResNet18 model to mitigate overfitting risk and if dataset is small
- Load pretrained model and adapt input and output later to own data

```
model = models.resnet18(pretrained=True)

model.conv1=nn.Conv2d(1, model.conv1.out_channels,
kernel_size=model.conv1.kernel_size[0], stride=model.conv1.stride[0],
padding=model.conv1.padding[0])

num_ftrs = model.fc.in_features

model.fc = nn.Linear(num_ftrs, len(classes))
```

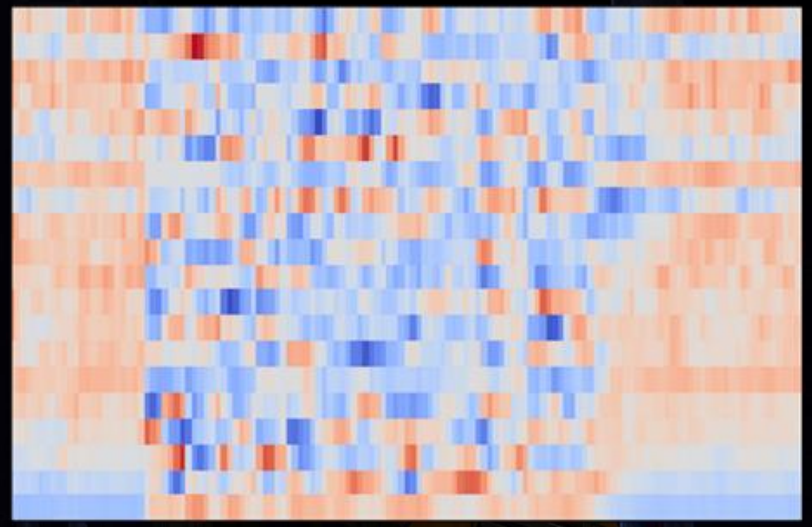
V7: CNN for Speech Classification II

- Feature Augmentation - Enhance Spectrograms features for optimal performance by hyper-parameter tuning

Data-Centric

- Mel Spectrograms work well for most audio classification
  - Alternatively, Mel Frequency Cepstral Coefficients may work better
  - MFCC extract a compressed representation of the frequency bands from the Mel Spectrograms that correspond to the most common frequencies at which humans speak

MFCC



- Data augmentation – Increase data by transforming them

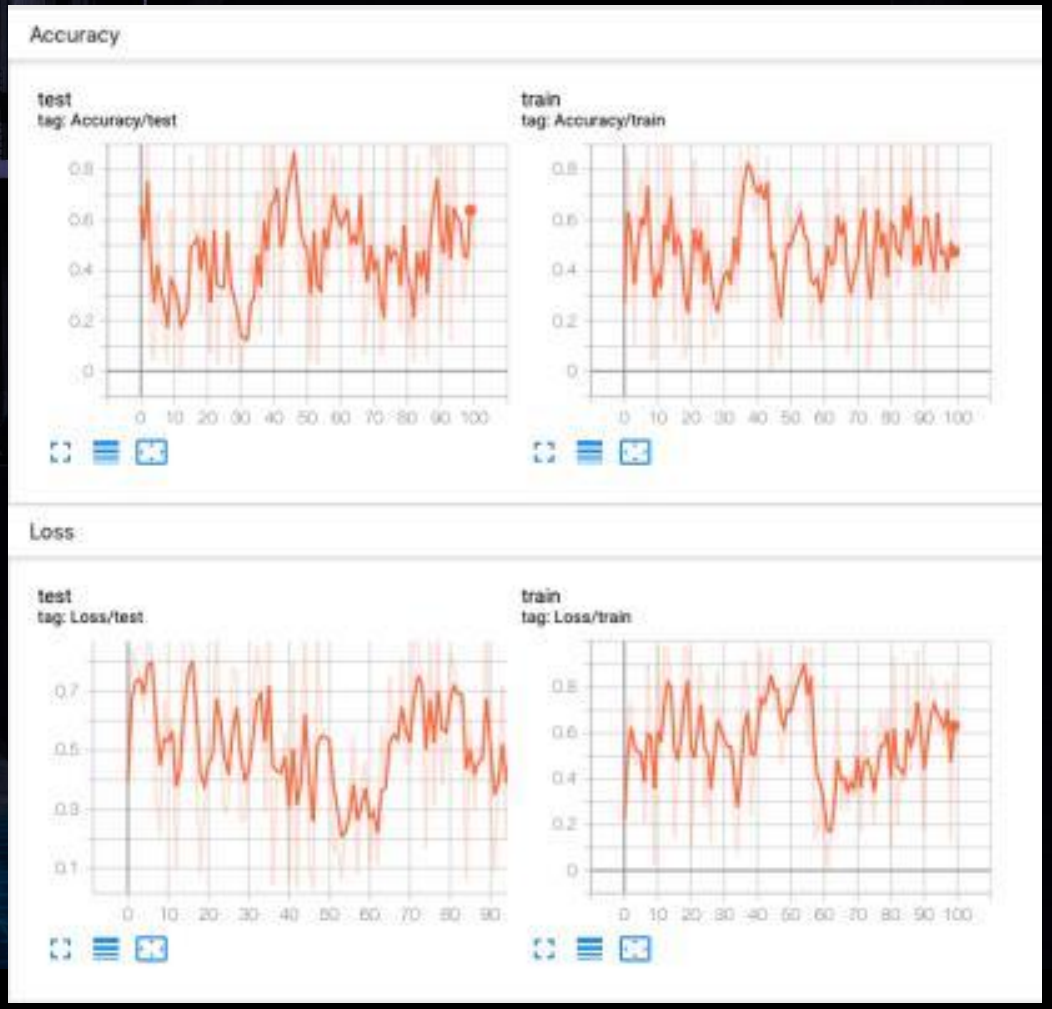
V8: Model Monitoring, Evaluation

Agenda

- Appreciate using TensorBoard for post-training analysis
- Appreciate setting up data pipeline for training
- Sample codes

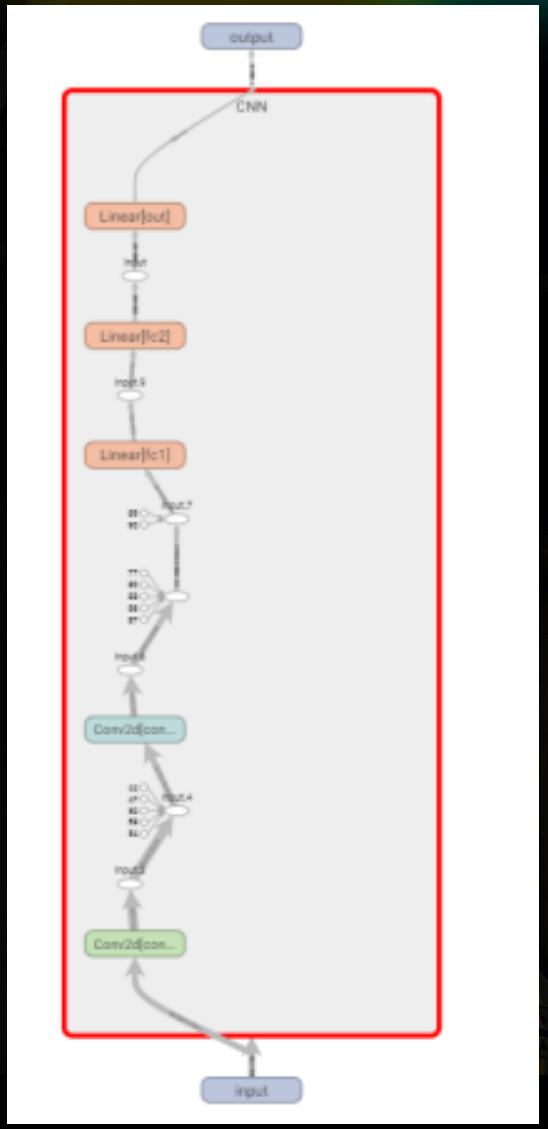
V8: Model Monitoring, Evaluation

- Lots of information can be logged for one experiment
  - Train and test data should be grouped together
  - Loss and accuracy are metrics that can be tracked



V8: Model Monitoring, Evaluation

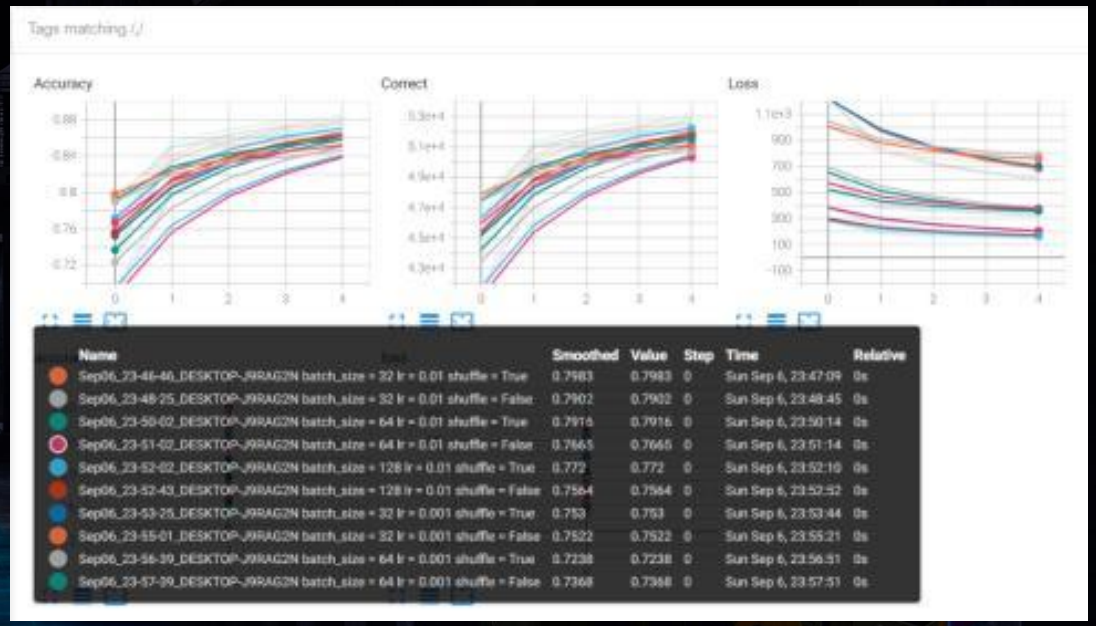
- Checking of model lineage under the “Graphs” tab
- Tells the pipeline of how dimensions of the data changes after every operations



V8: Model Monitoring, Evaluation

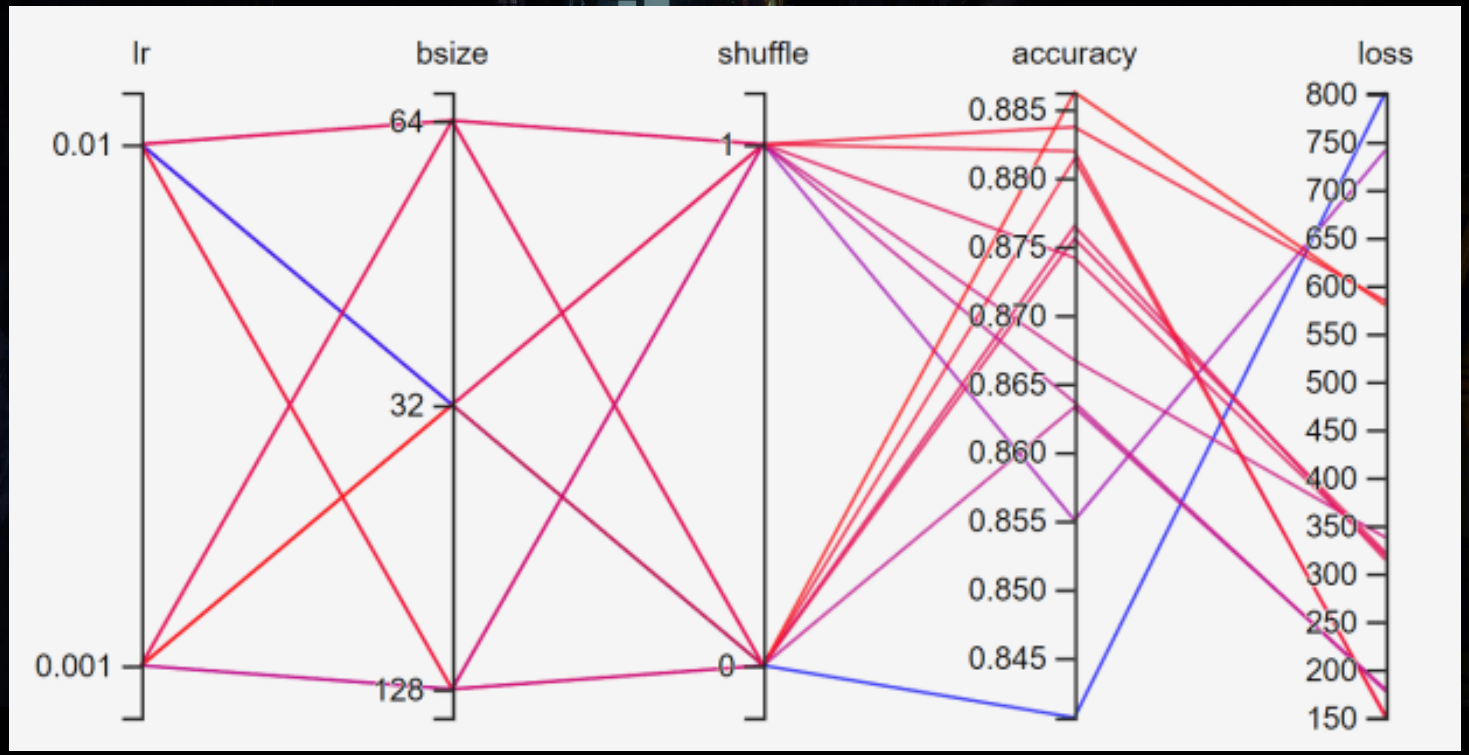
- Conduct hyperparameter tunings and add to the TensorBoard with `tb.add_hparams` to compare each training with different hyperparameters

```
tb.add_hparams(
{ "lr": lr,
  "bsize": batch_size,
  "shuffle": shuffle},
  "accuracy": total_correct/ len(train_set),
  "loss": total_loss
})
```



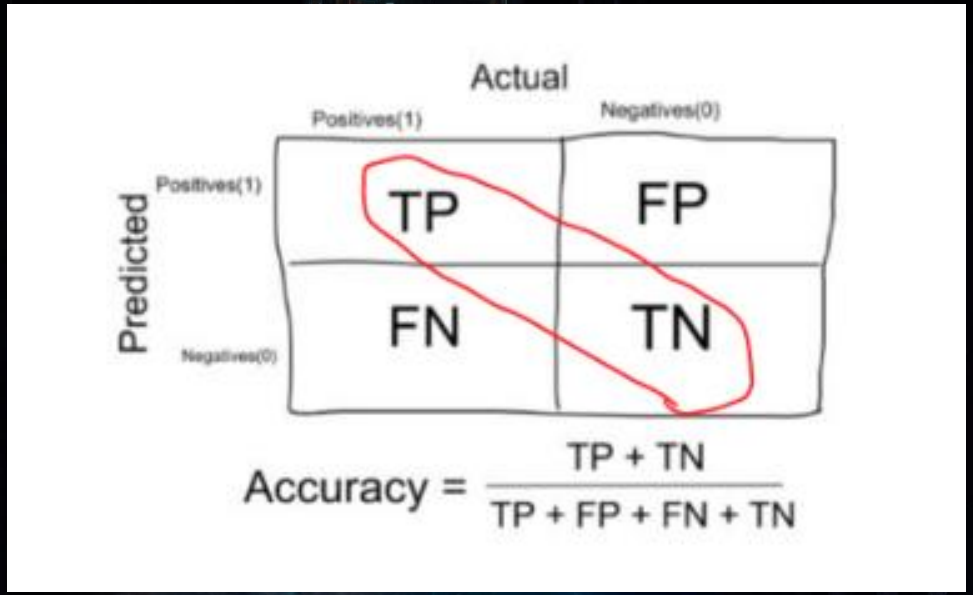
V8: Model Monitoring, Evaluation

- For example, from the Hyperparameter Graph setting shuffle to False(0) can yield very poor results.
- Hence setting shuffle to always True(1) is ideal for training as it adds randomization.



V8: Model Monitoring, Evaluation

- Could use `sklearn` package to do more analysis into the classification outputs
- `PyTorch.Ignite` also provide faster means of calculating evaluation metrics https://pytorch.org/ignite/metrics.html



DSTA
Defence Science & Technology Agency

V8: Model Monitoring, Evaluation

brainhack

- Could use `sklearn` package to do more analysis into the classification outputs
- `PyTorch.Ignite` also provide faster means of calculating evaluation metrics https://pytorch.org/ignite/metrics.html

$$Precision = \frac{truepositives}{truepositives + falsepositives}$$

$$Recall = \frac{truepositives}{truepositives + falsenegatives}$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

DSTA
Defence Science & Technology Agency

V8: Model Monitoring, Evaluation

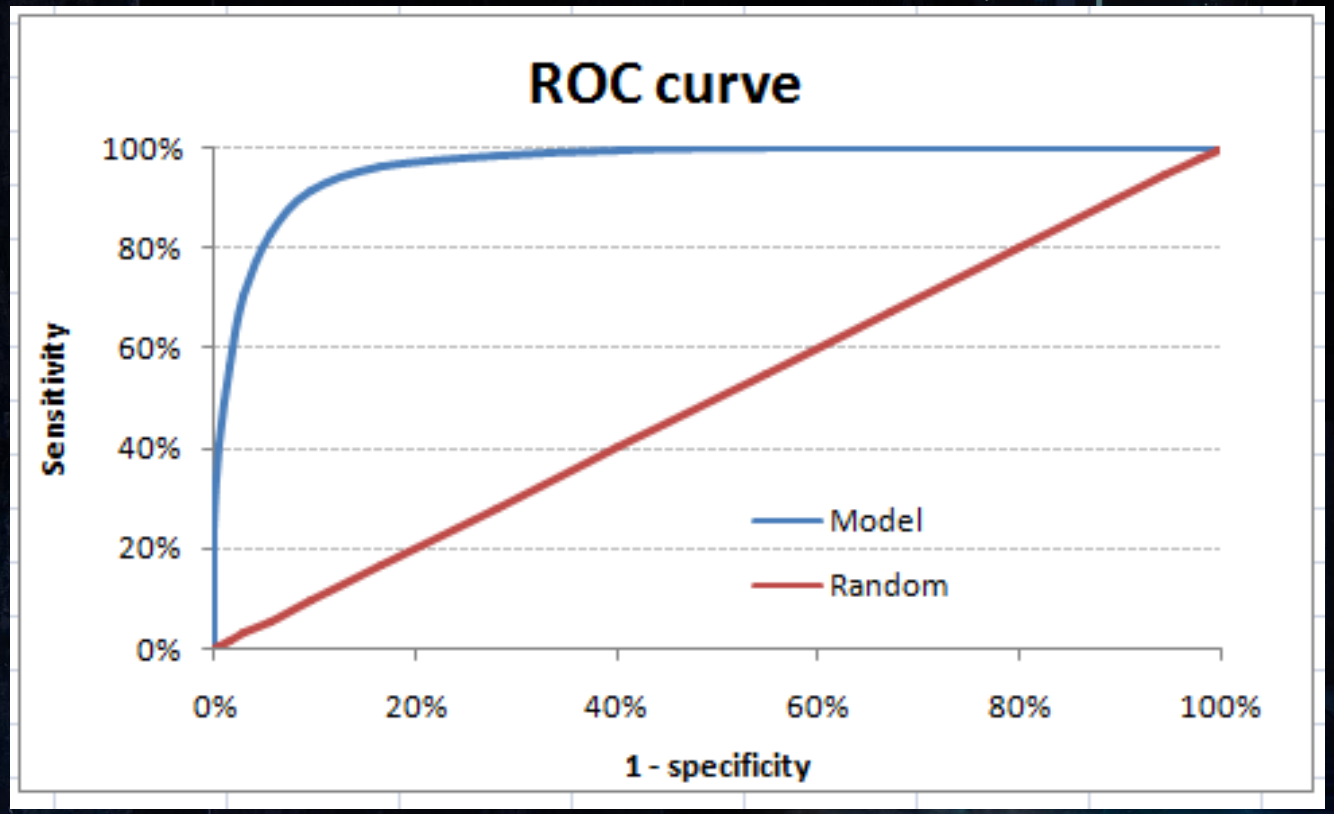
brainhack

		Predicted Category		
		Backgrnd	Birds	Insects
Labelled Category	Backgrnd	5467	1604	167
	Birds	198	1470	48
	Insects	25	9	680

		Predicted				
		Backgroun	Birds	Insects	Total	Recall
Actual	Backgroun	5467	1604	167	7238	76%
	Birds	198	1470	48	1716	86%
	Insects	25	9	680	714	95%
	Total	5690	3083	895		
	Precision	96%	48%	76%		

- Very precise with background but over-predict many audios as non-background
- Very good recall of the species which helps in detection

V8: Model Monitoring, Evaluation



- ROC curve informs the optimal trade-offs between precision and recall for the current model

V9: RNN for sequence Audio data

Agenda

- Appreciate RNN for sequencing data
- Appreciate use-cases for RNN
- Appreciate the difference between CNN and RNN

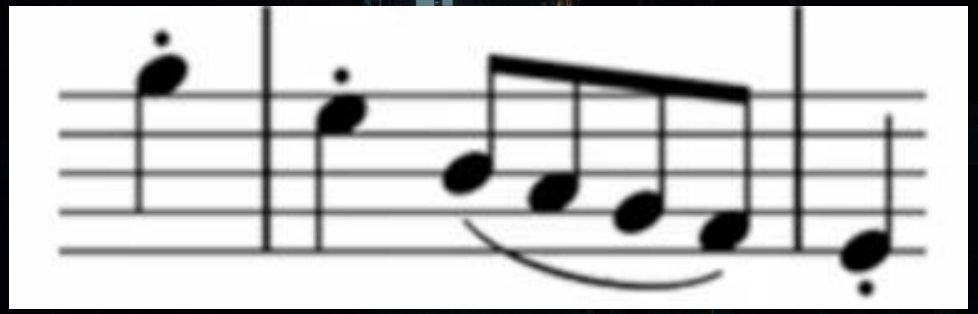
V9: RNN for sequence Audio data

- Recurrent Neural Networks are NN specially designed to deal with input data that are **sequences**, meaning repeated observations in time.
- **Standard ML/DL**: One input → One output.
  - Examples:
    - One vector of observations => one classification y
    - One text => one prediction y
    - One image => one classification y
- **Sequence data**: Multiple inputs => One output (potentially multiple).
- RNN very popular for text and speech data recognition problems.

V9: RNN for sequence Audio data

- Speech Recognition

- Music Generation



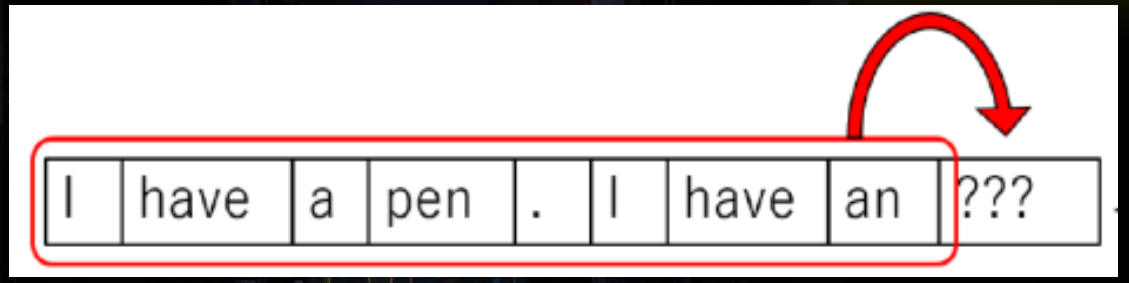
- Translation

Voulez-vous chanter avec moi?

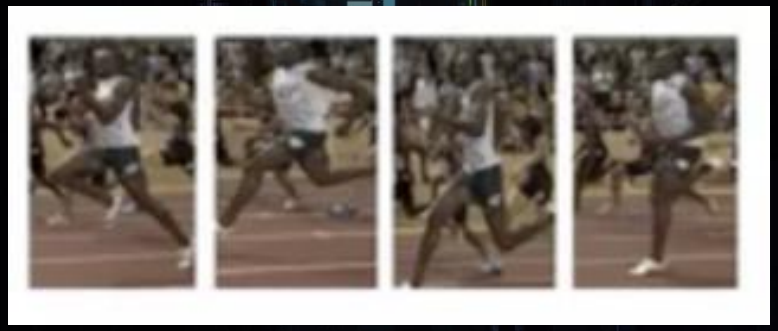
Do you want to sing with me?

V9: RNN for sequence Audio data

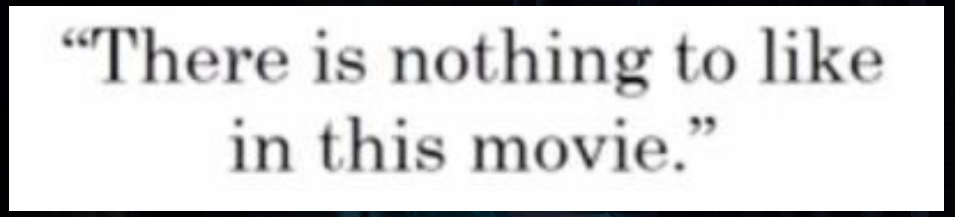
- Text Generation

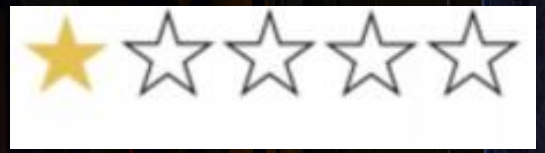


- Video Recognition



- Sentiment Classification

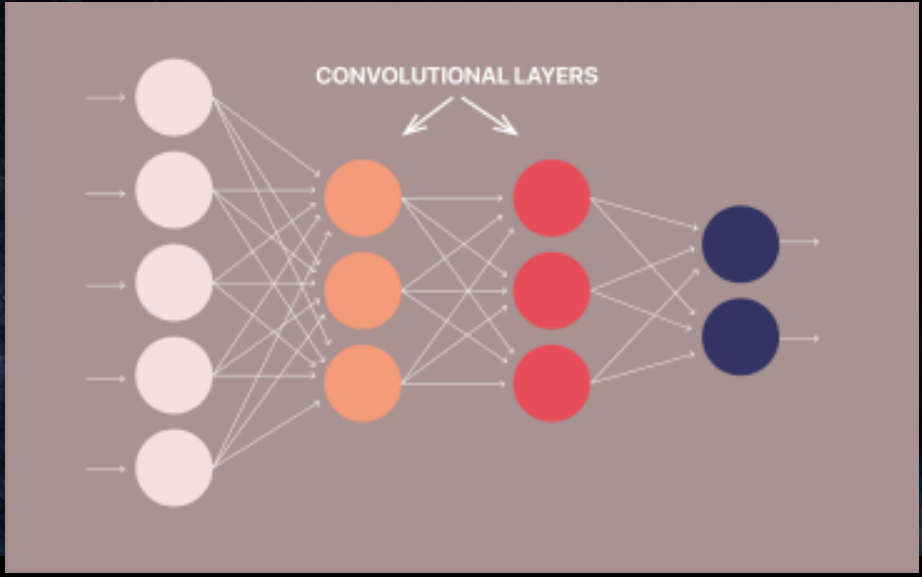


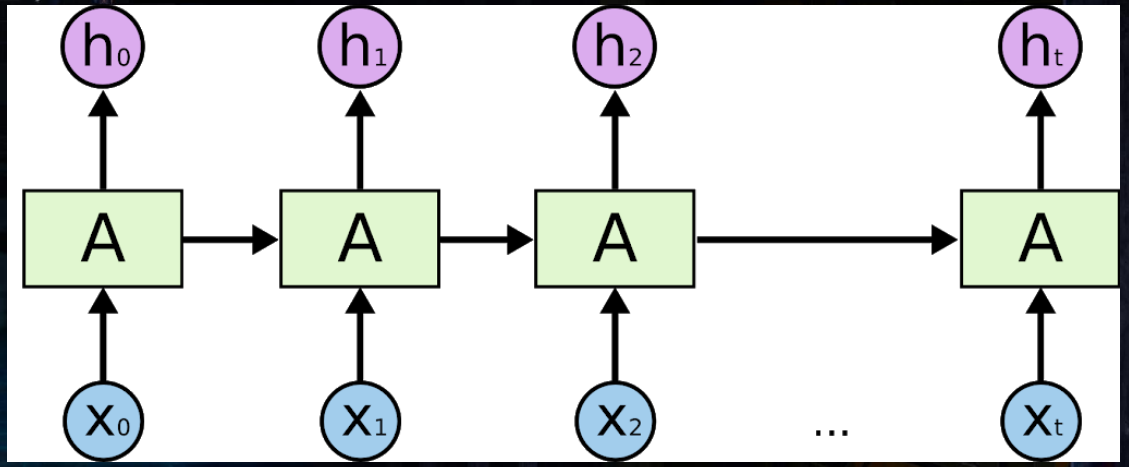


- DNA sequencing

V9: RNN for sequence Audio data

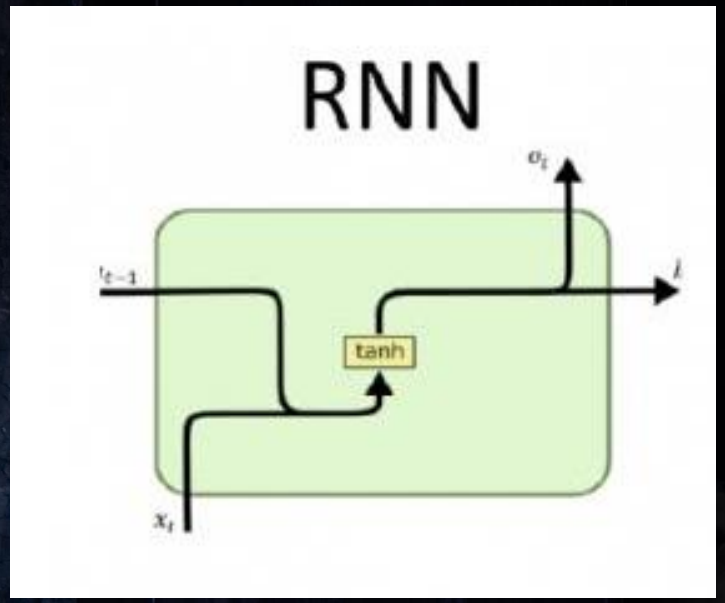
CNN	RNN
• Visual data	• Data that has time element to it (Audio, Video)
• Static data	• Sequence data
• No memory	• Memory inherent
• Convolutional filters	• Gated units (forget gates etc.)

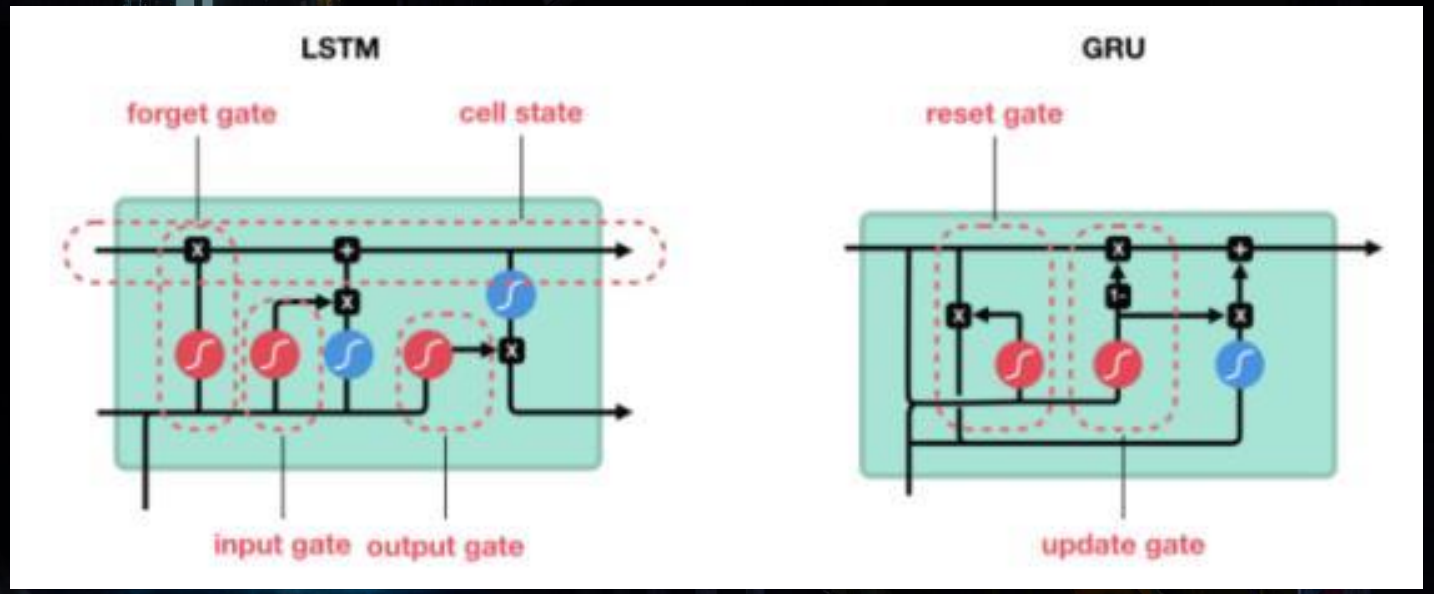




V9: RNN for sequence Audio data

- RNN unlike CNN has a family of different architectures
- If CNN is glucose, RNN is protein/fats





DSTA
Defence Science & Technology Agency

V9: RNN for sequence Audio data

brainhack

RNN	LSTM	GRU
Uses previous data point to predict the next point	Store previous data points in long sequences	
Short-term memory problem	Long-term and Short-term memory advantage	
	Gates to control flow of information	
	Reset gate Update gate Forget gate Output gate	Reset gate Update gate
	Output pass through activation function	Output does not pass through activation function

DSTA
Defence Science & Technology Agency

V10: RNN Architectures

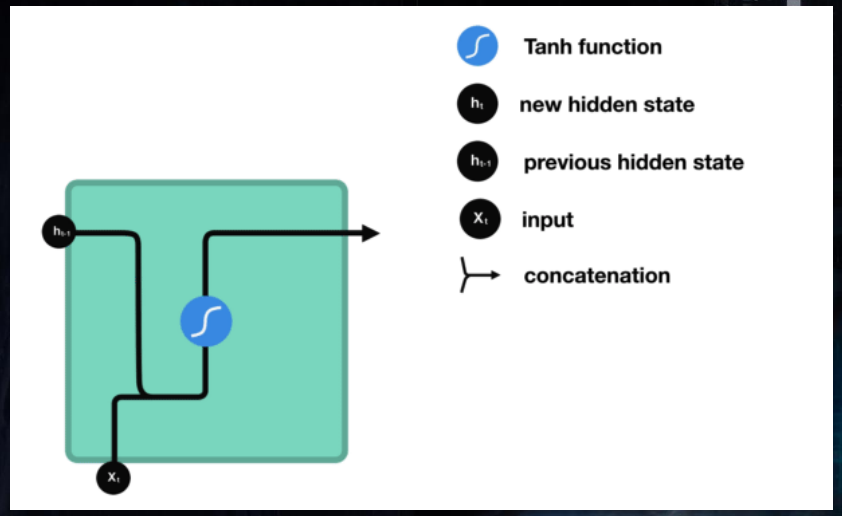
brainhack

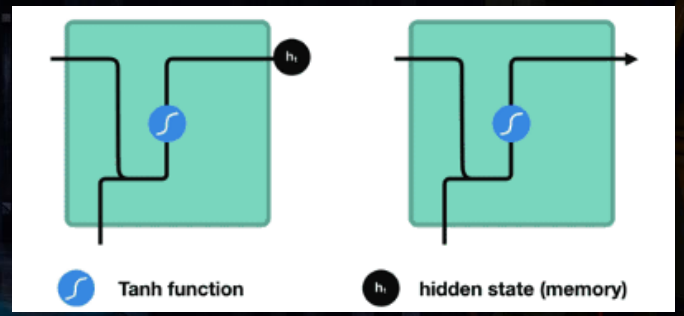
Agenda

- Appreciate RNN architecture
- Appreciate the building blocks to RNN model

V10: RNN Architectures

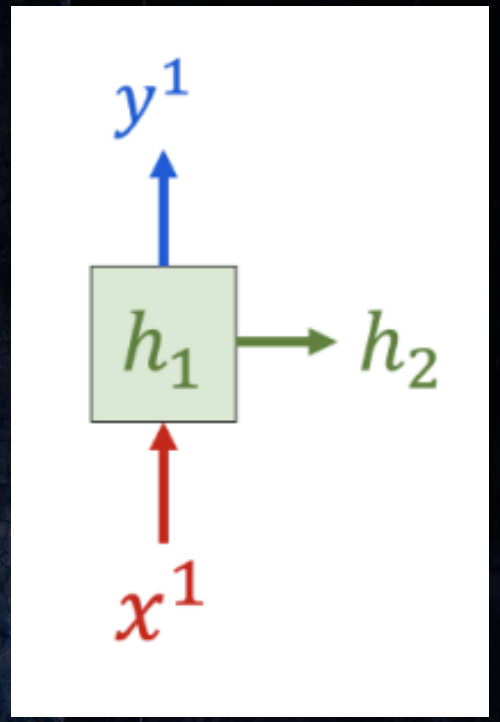
- Most basic building block of RNN – a single cell
- Input vector combined (multiplied/added) with hidden state vector to form a vector
- Vector gets transformed by the activation function and output a new hidden state vector
- New hidden state vector passed to another RNN cell



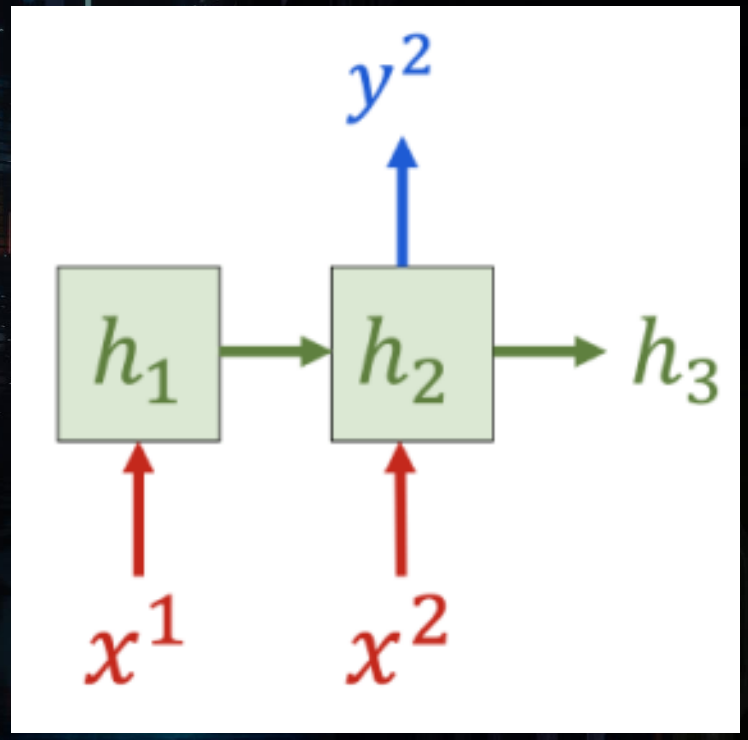


V10: RNN Architectures

- RNN has memory power



- Predict y1
- Calculate h2 (weight)

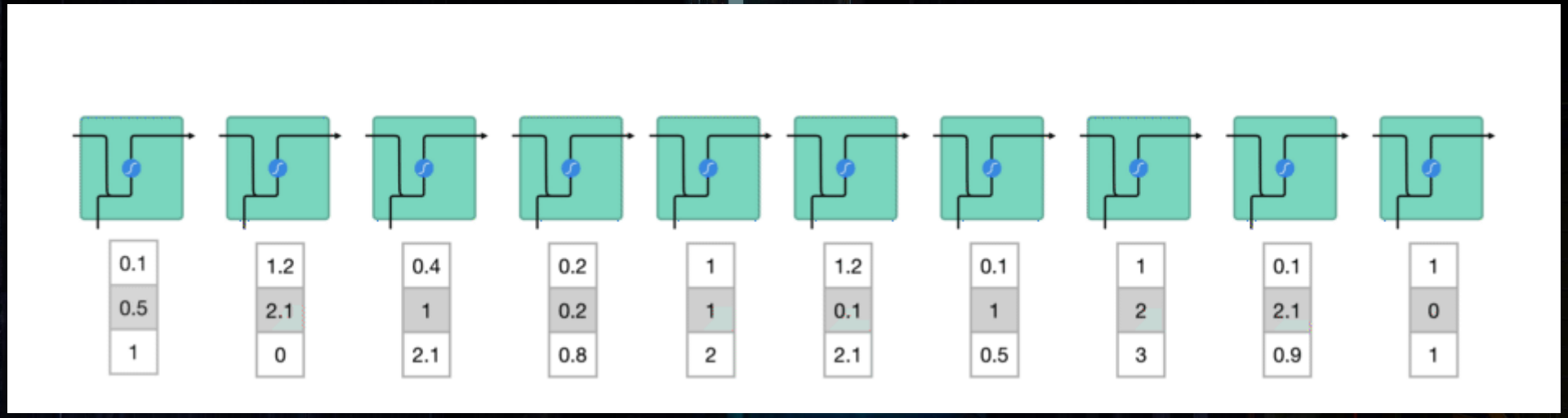


- Solve h2
- Predict y2
- Calculate h3

RNN has memory that updates the internal state h(n) weights every time a new observation is seen

V10: RNN Architectures

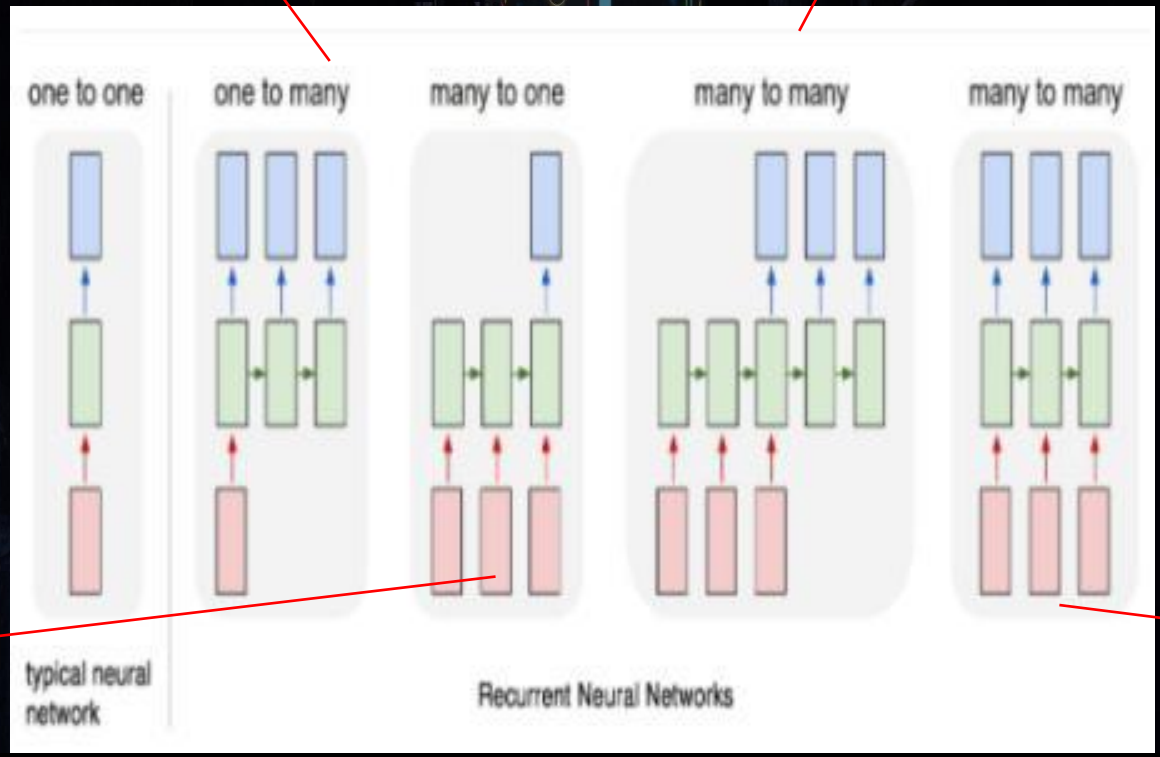
- Audio sequences get transformed into vectors per timeframe (seconds)
- RNN processes sequences of vectors one by one
- Hidden state vectors store the memory of all previous input data



V10: RNN Architectures

- A huge diverse range of RNN exists for various problem
  - Music Generation
  - Image captioning

Translation



- Classification
- Anomaly detection

- Autonomous Driving
- Video Recognition

DSTA
Defence Science & Technology Agency

V11: LSTM Architectures

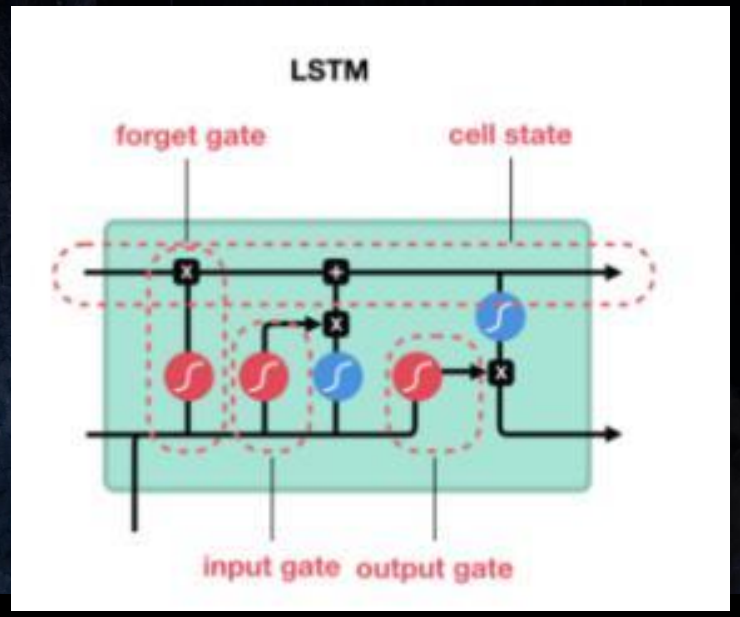
brainhack

Agenda

- Appreciate LSTM architecture
- Appreciate the building blocks to LSTM model
- Sample codes

V11: LSTM Architectures

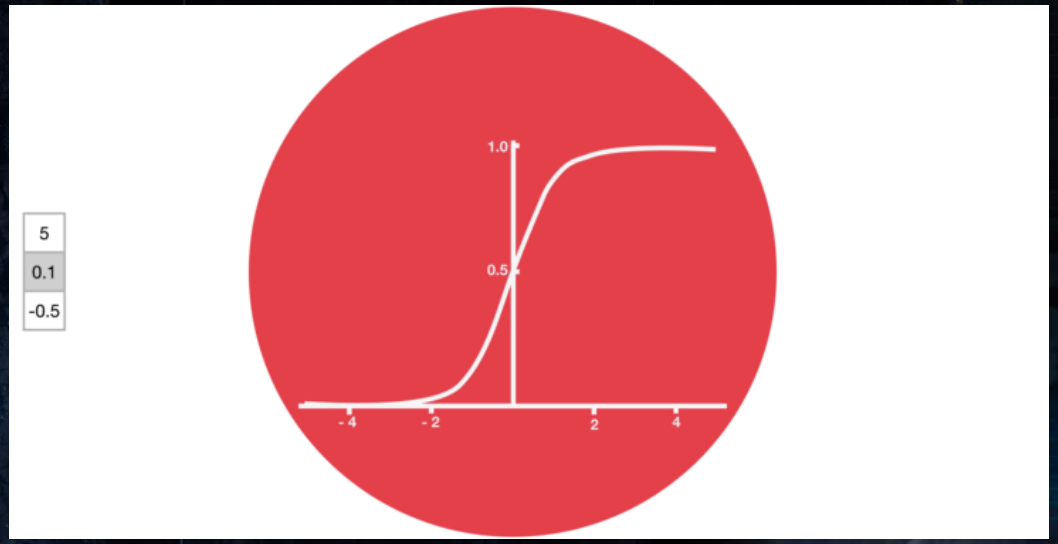
- Memory has long-term and short-term – RNN suffers from short-term memory
- Challenging for hidden states to carry huge information to future time periods
- LSTM (Long Short Term memory) saves the day
  - Gates that regulate the flow of information



V11: LSTM Architectures

- Gates learn which data is important and which is droppable
- Gates are neural networks that decide which data should be kept in the hidden state
- Hold data from the far past and reduce effects of short-term memory

Forget Gate

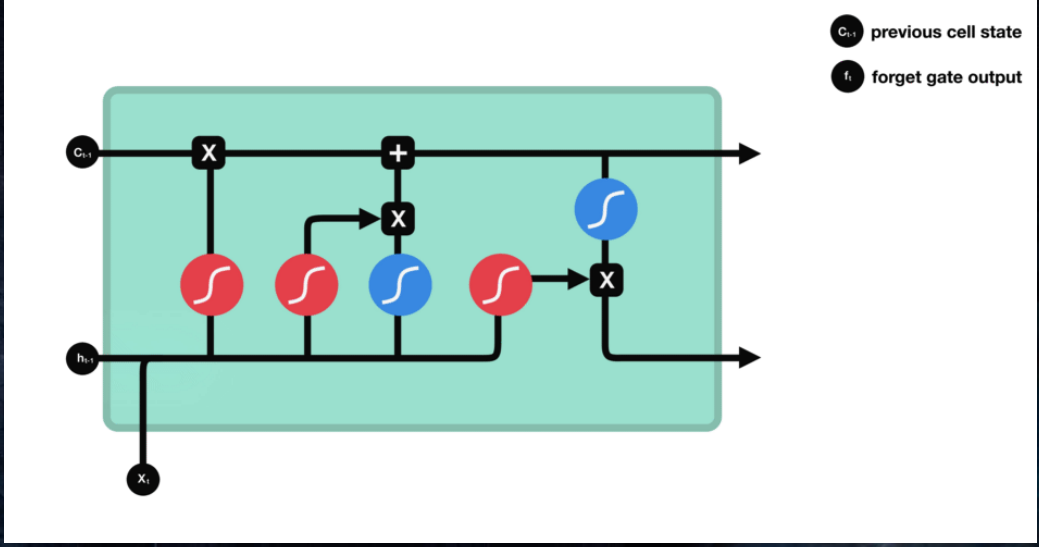


- Sigmoid function is bounded differentiable function
- Between 0 and 1
- Data passes through the sigmoid function

V11: LSTM Architectures

- Gates learn which data is important and which is droppable
- Gates are neural networks that decide which data should be kept in the hidden state
- Hold data from the far past and reduce effects of short-term memory

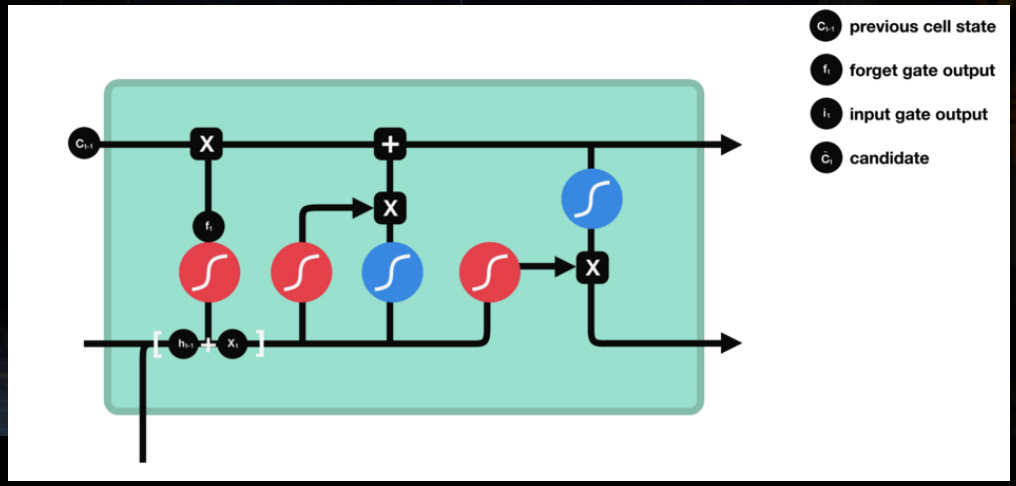
Forget Gate



- Any values multiplied by 0 is 0
- Any values multiplied by 1 is itself
- Forget data as it becomes 0
- Keep data as data remains the same when multiplied by 1
- Can also be used to update data

V11: LSTM Architectures

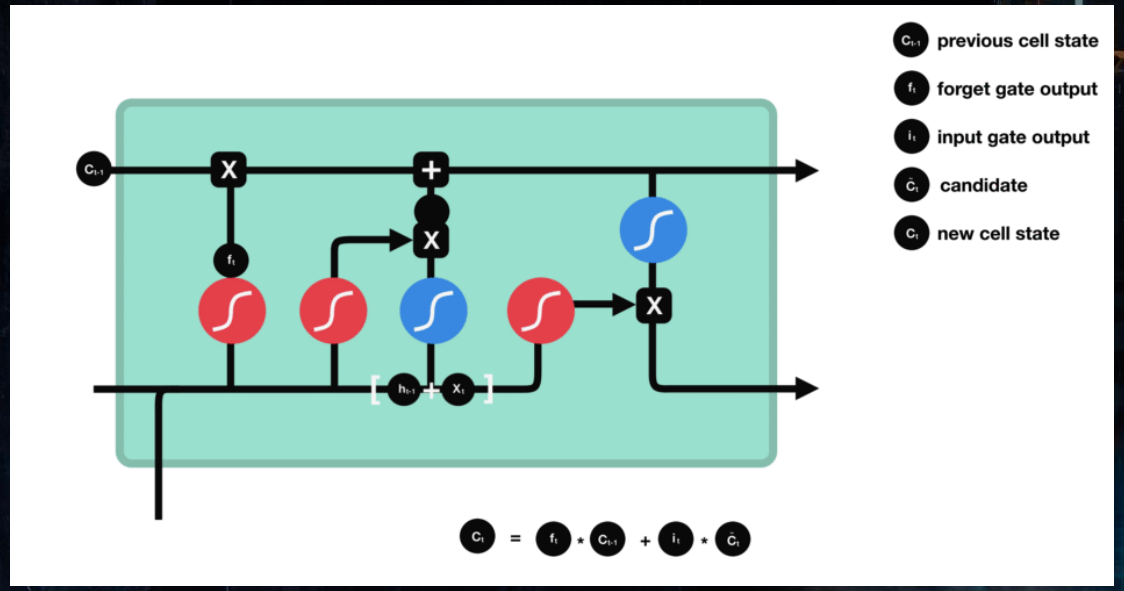
- Current data enters the input gate
- Current data input and previous hidden state are passed into the sigmoid function [0 to 1] and tanh function [-1 to 1]
- Sigmoid - 0 means not important, and 1 means important
- Tanh - regulate the network with negative to positive weights
- Multiply the tanh output with the sigmoid output to decide which to keep, increase and decrease
  - Prevent exploding gradients



Input Gate

V11: LSTM Architectures

- New cell state is calculated
- Cell state multiplies by forget gate vector
- Input gate vector adds to the cell state vector



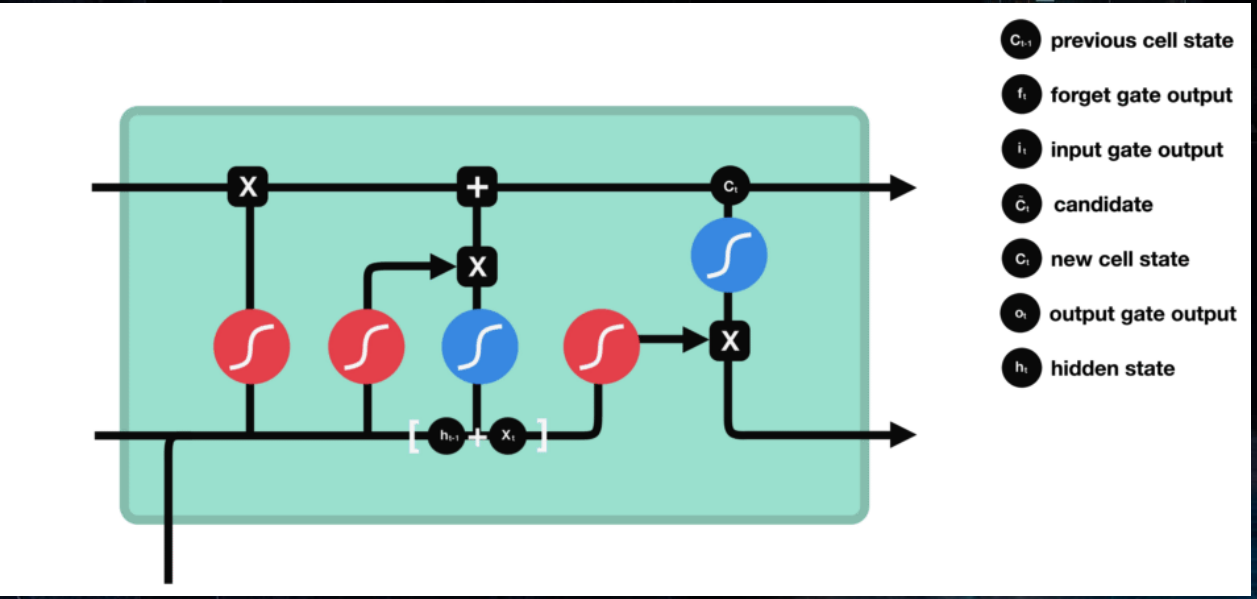
New Cell State

DSTA
Defence Science & Technology Agency

V11: LSTM Architectures

brainhack

- New cell state passed into tanh function
- Output data and previous hidden state are passed into sigmoid function
- Multiply both outputs to decide new hidden state vector output



Output Gate

DSTA
Defence Science & Technology Agency

V11: LSTM Architectures

brainhack

- Forget gate decides what data to drop (hidden sates, input data from previous time)
- Input gate decides what data to add from current time
- Output gate decides next hidden state for next time period

```
def LSTM(prev_ct, prev_ht, input):
  combine = prev_ht + input
  ft = forget_layer(combine
  candidate = candidate_layer(combine)
  it = input_layer(combine)
  Ct = prev_ct * ft + candidate * it
  ot = output_layer(combine)
  ht = ot * tanh(Ct)
  return ht, Ct
```

DSTA
Defence Science & Technology Agency

V12: RNN with PyTorch

brainhack

Agenda

- Appreciate RNN(LSTM) model development on PyTorch
- Sample codes

DSTA
Defence Science & Technology Agency

V12: RNN with PyTorch

brainhack

- Use `nn.LSTM` for forget, input and output gate network layer
- Use `nn.Linear` with the number of class for a fully connected flat layer to predict the classes
- Create a hidden state vector initialized to zero
- Create a cell state vector initialized to zero
- Forward push propagate the LSTM

```
# Recurrent neural network (many-to-one)
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, num_classes, device):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)
        self.device = device

    def forward(self, x):
        # Set initial hidden and cell states
        batch_size = x.size(0)
        h0 = torch.zeros(self.num_layers, batch_size, self.hidden_size).to(self.device)
        c0 = torch.zeros(self.num_layers, batch_size, self.hidden_size).to(self.device)

        # Forward propagate LSTM
        out, _ = self.lstm(x, (h0, c0))  # shape = (batch_size, seq_length, hidden_size)

        # Decode the hidden state of the last time step
        out = self.fc(out[:, -1, :])
        return out
```

V13: Model Training

Agenda

- Appreciate setting up training runtime
- Appreciate setting up data pipeline for training
- Sample codes

V13: Model Training

- Set global parameters such as epochs, learning rate and train test split etc.

- Set augmentation steps as a sequential list for model to load and process

V14: Additional Resources and Links

Agenda

- Sources for additional resources and links

V14: Additional Resources and Links

Resource Links

Data Augmentation - https://medium.com/@makcedward/data-augmentation-for-audio-76912b01fdf6

PyTorch - https://PyTorch.org/tutorials/

Recurrent Neural Networks- https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

Long Short Term Memory - https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359#:~:text=LSTM%20is%20a%20recurrent%20neural,time%20lags%20of%20unknown%20duration.&text=RNN%20cell%20takes%20in%20two,and%20observation%20at%20time%20%3D%20t.

Gated Recurrent Unit - https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

V14: Additional Resources and Links

Resource Links

Markov for Speech Classification- https://jonathan-hui.medium.com/speech-recognition-gmm-hmm-8bb5eff8b196

Machine Learning (Gaussian) for Speech Classification - https://medium.com/analytics-vidhya/speaker-identification-using-machine-learning-3080ee202920

CNN & RNN combined for Speech Classification - https://towardsdatascience.com/using-cnns-and-rnns-for-music-genre-recognition-2435fb2ed6af

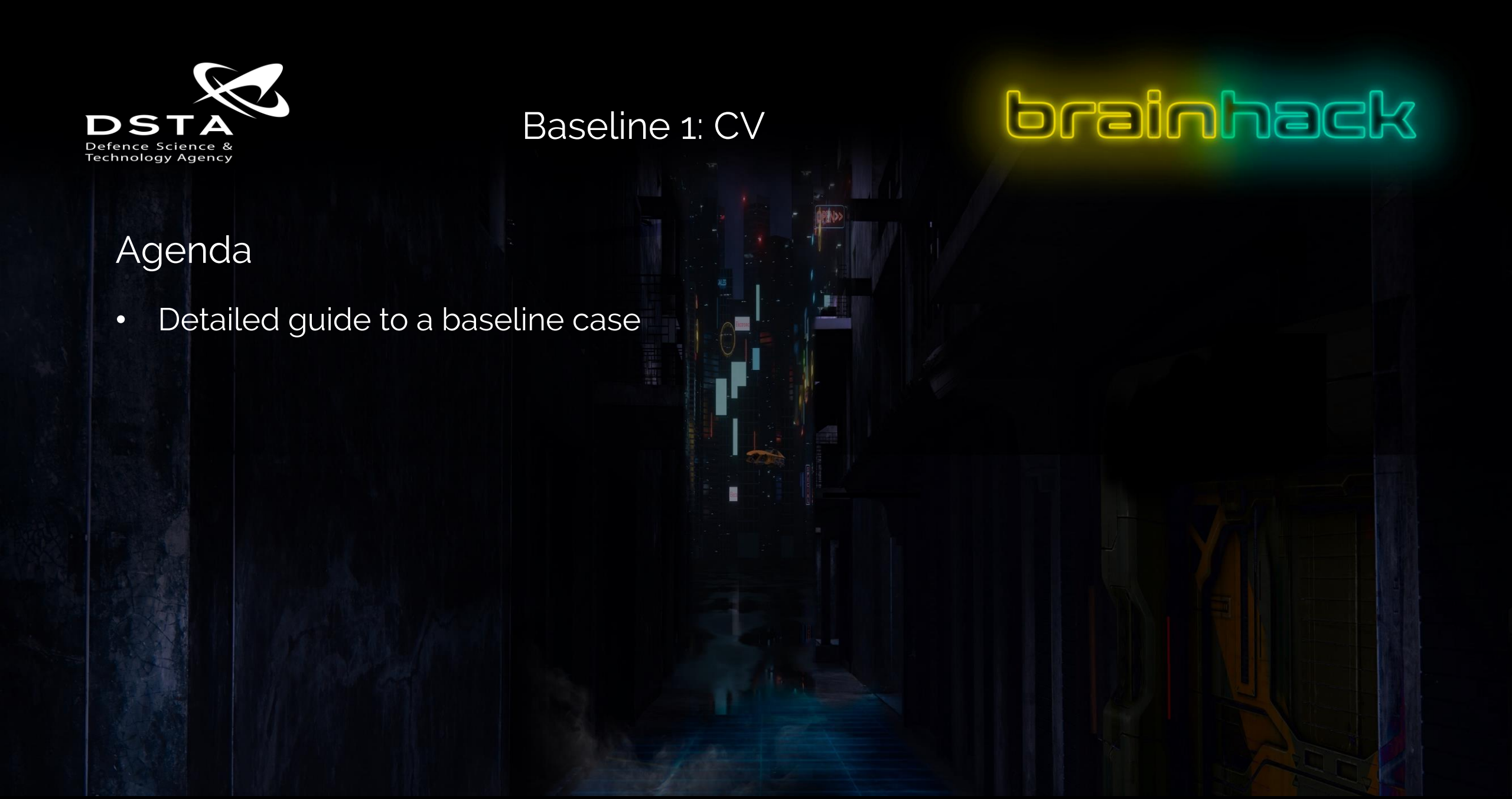
Hyperparameter Tuning - https://towardsdatascience.com/simple-guide-to-hyperparameter-tuning-in-neural-networks-3fe03dad8594

Public Datasets - https://PyTorch.org/vision/stable/datasets.html

Baseline 1: CV

Agenda

- Detailed guide to a baseline case



Baseline 2: SC

Agenda

- Detailed guide to a baseline case


DSTA
Defence Science &
Technology Agency
brainhack
DEFENDING
CYBER
DEFENDERS
DISCOVERY
CAMP
OUR CYBERSPACE
TIL
2019
Today I Learned
CODE_EXP
SEETRUE
SPACECUBE
XR PERIENCE
TECH
SHOWCASE


DSTA
Defence Science &
Technology Agency
brainhack
Thank You