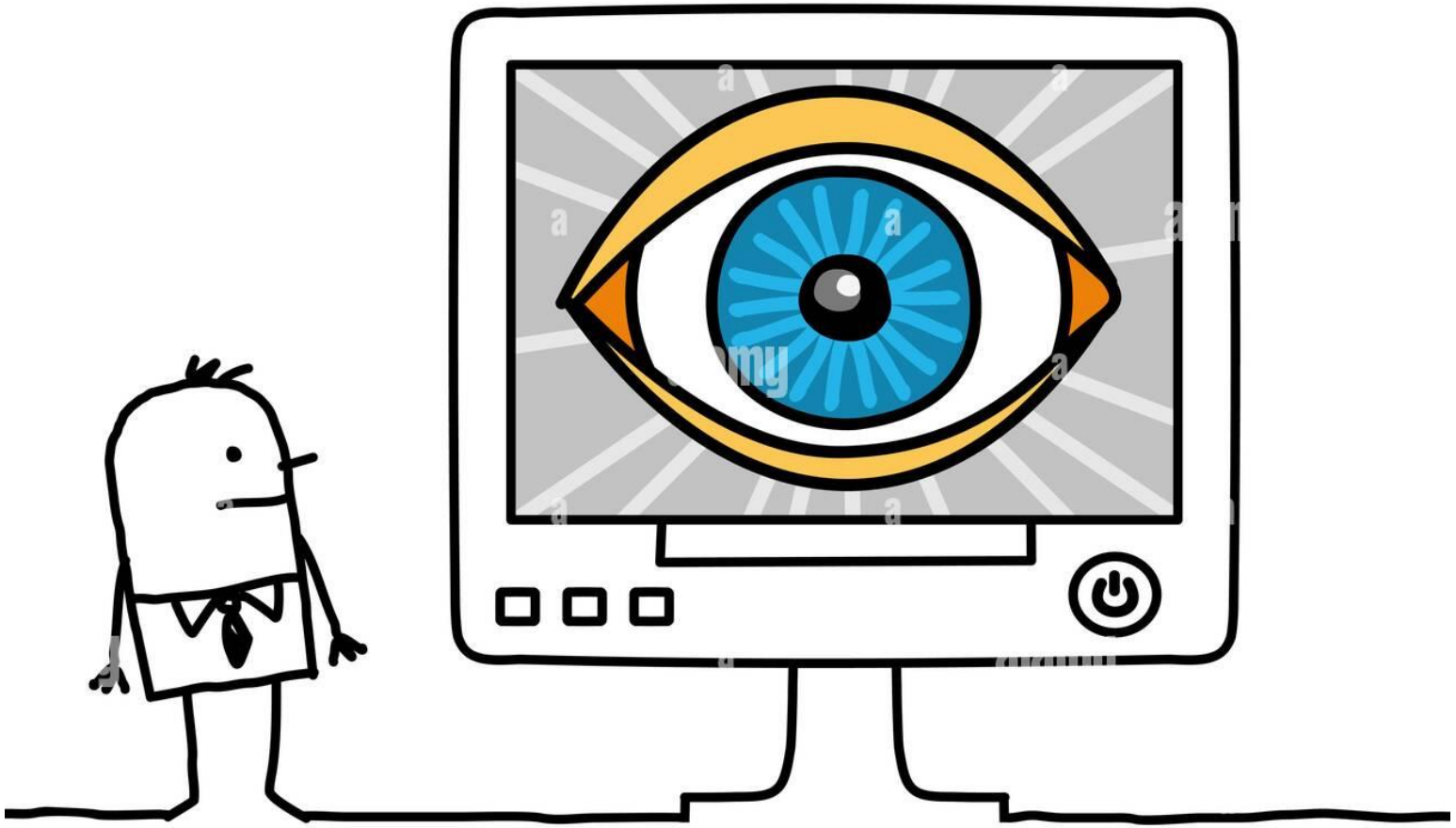
Application of Deep learning in Computer Vision

Akshay Daydar,
Research Scholar at IIT Guwahati

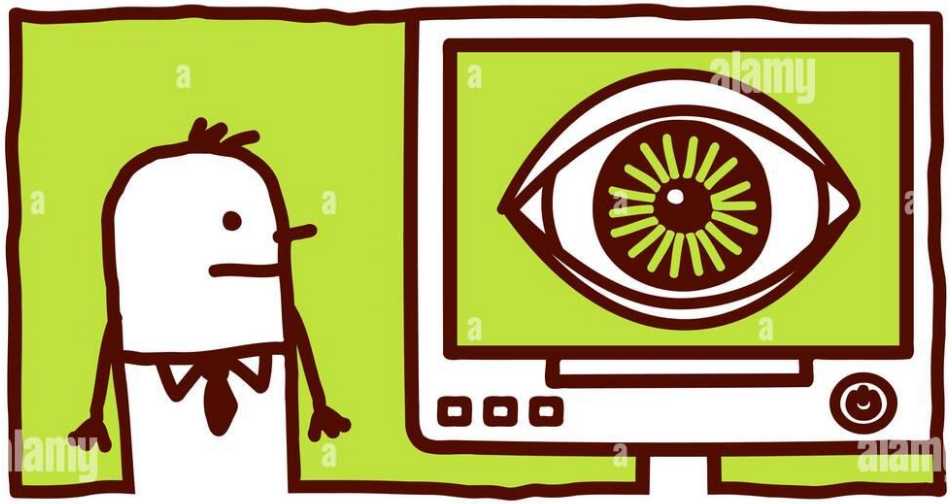


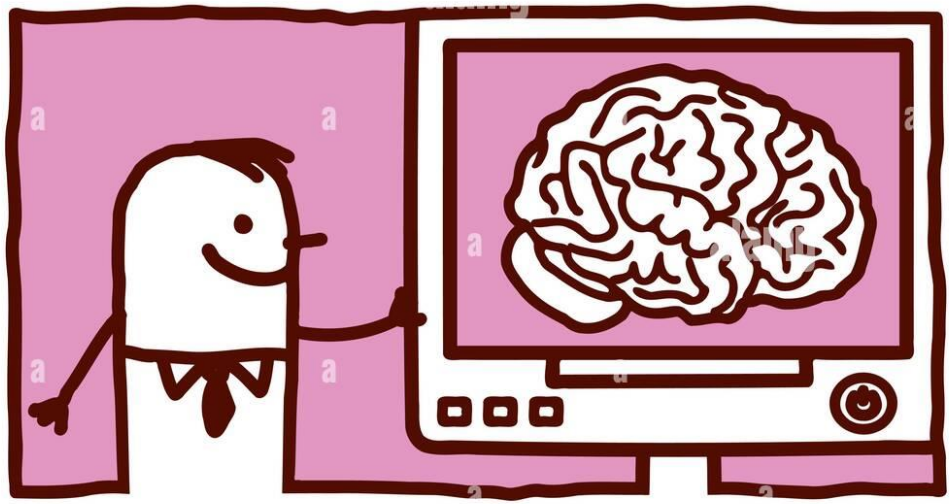
What is Computer Vision

Computer vision (CV) is the science and technology of machines that see.

Concerned with the theory for building artificial systems that obtain information from images.

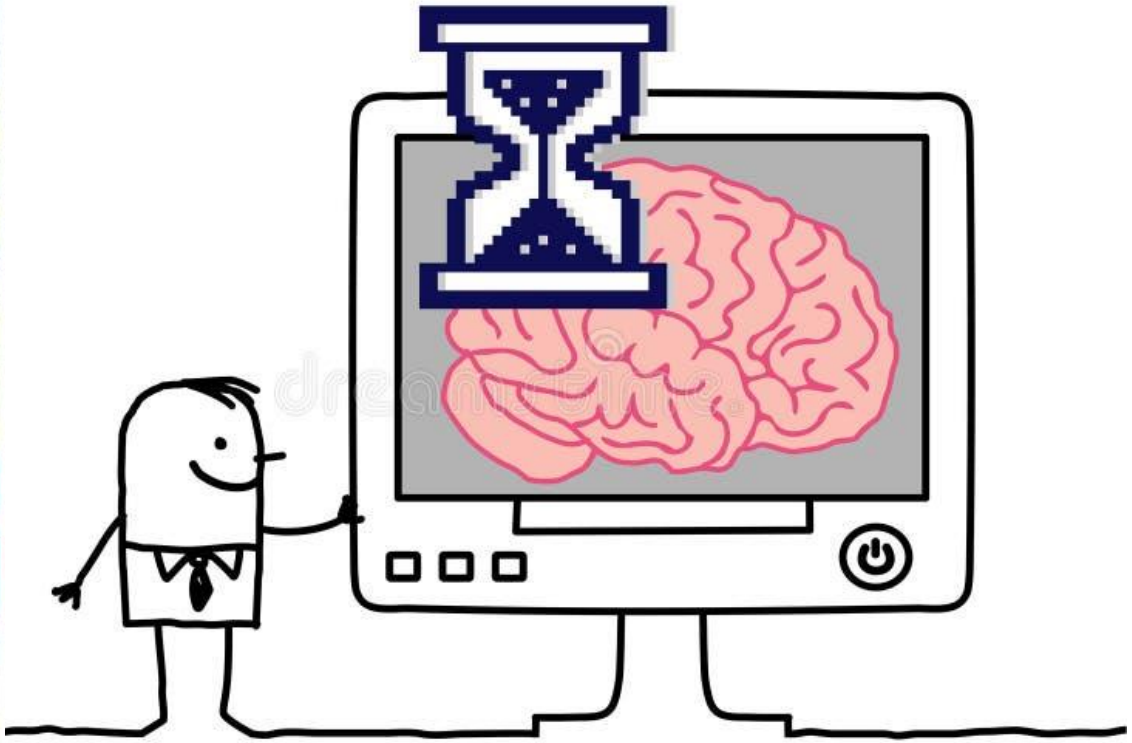
CV is concerned with automatic extraction, analysis and understanding of useful information from single image or a sequence of images– The British Machine Vision Association and Society for Pattern Recognition (BMVA)





What is Computer Vision

Computer Vision Make computers understand images and videos !!!



Computer vision vs human vision

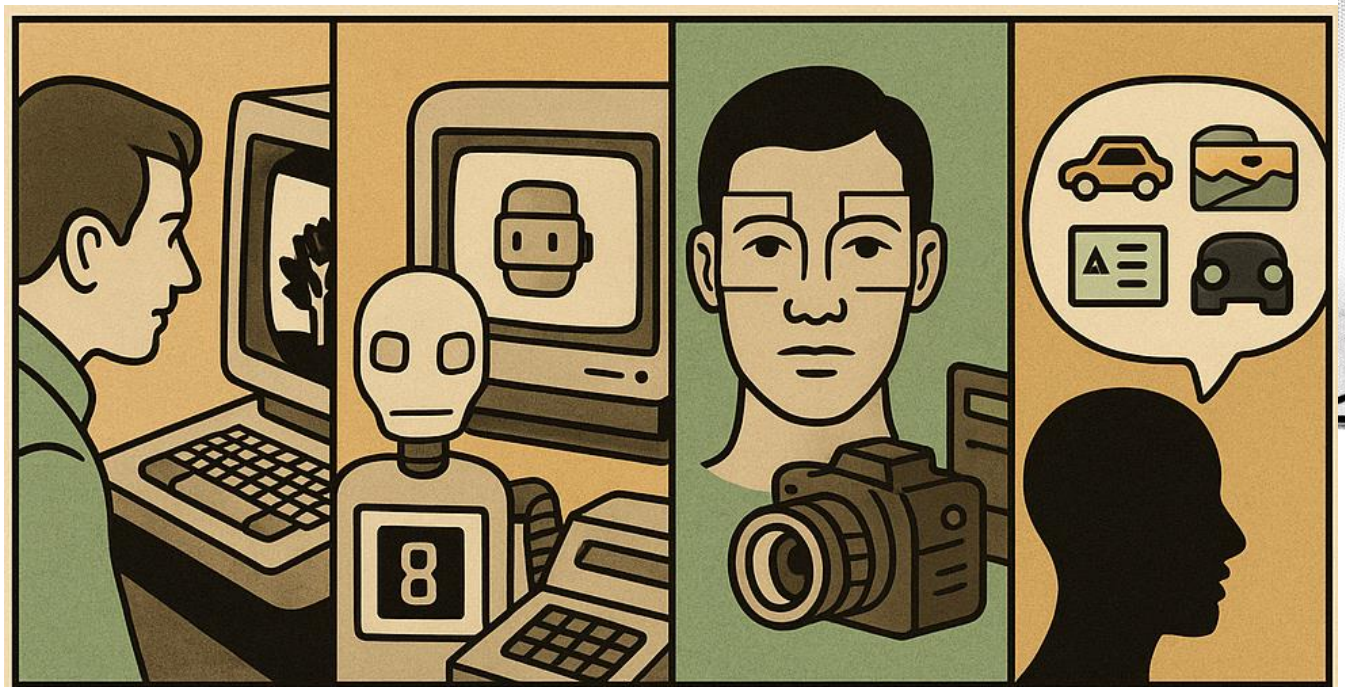
What we see

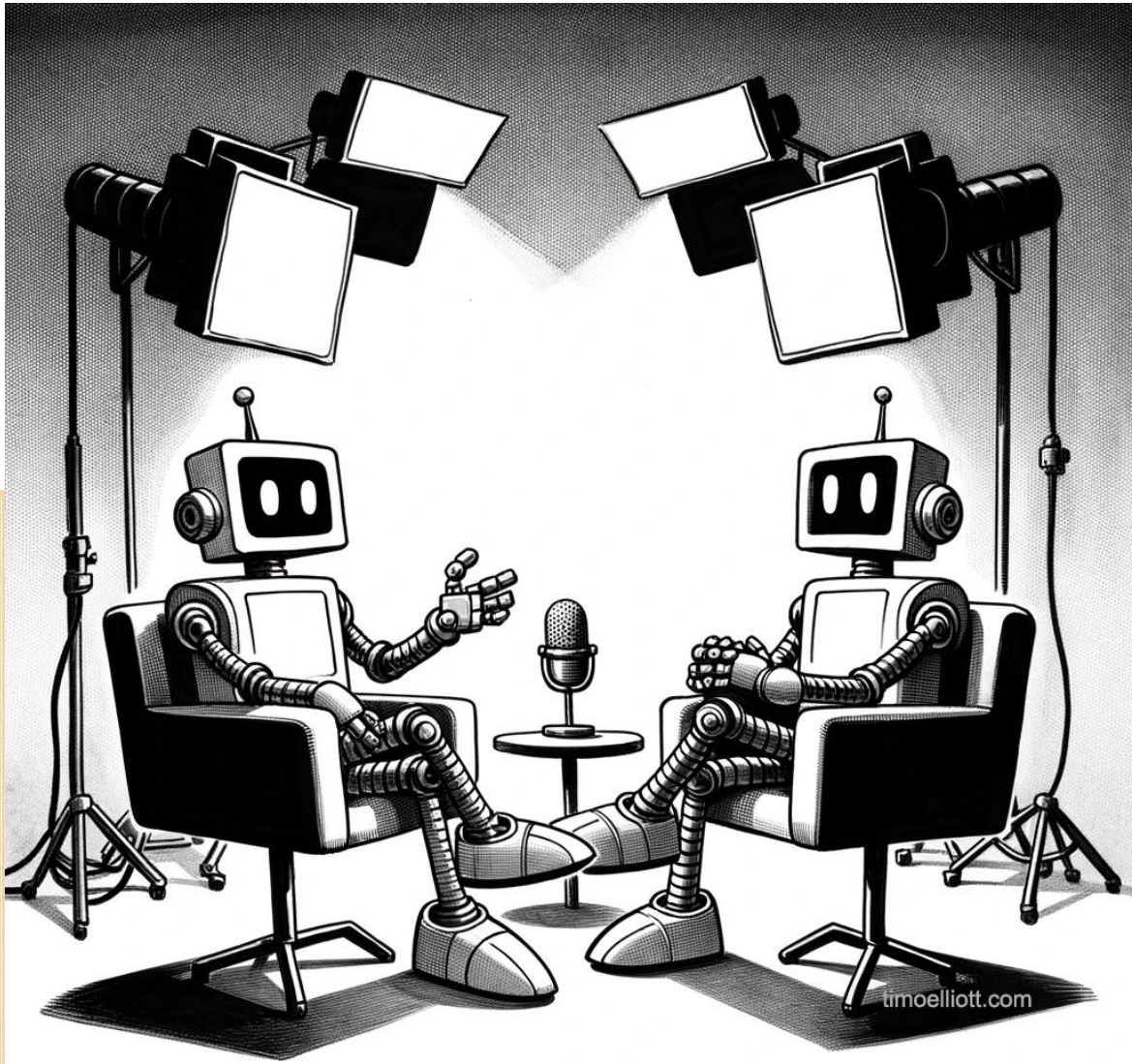
0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

What a computer sees

A little story about Computer Vision

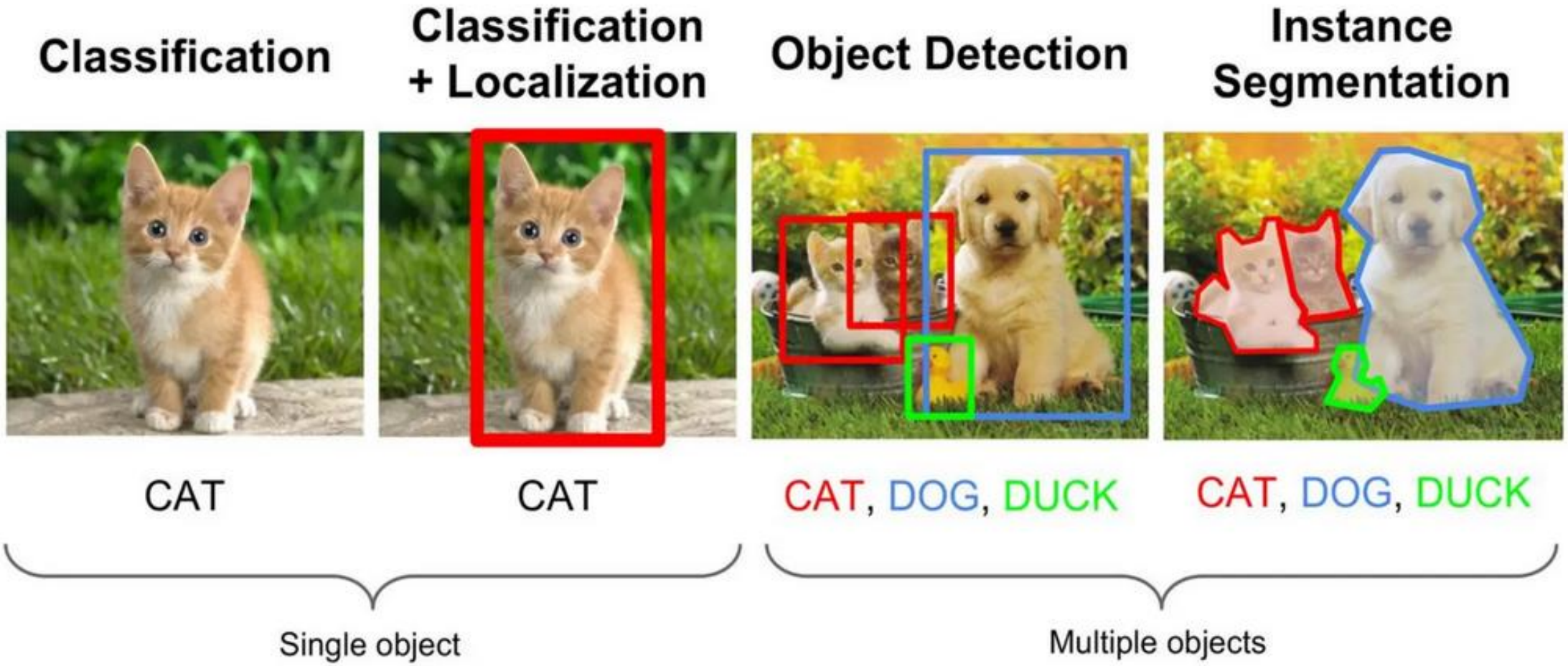
1 The First Glimpse (1960s–1970s)
2. The Growing Pains (1980s)
3. Cameras Get Smarter (1990s)
4. The Deep Learning Renaissance (2010s)
5. Vision Gets Intuitive (2020s)





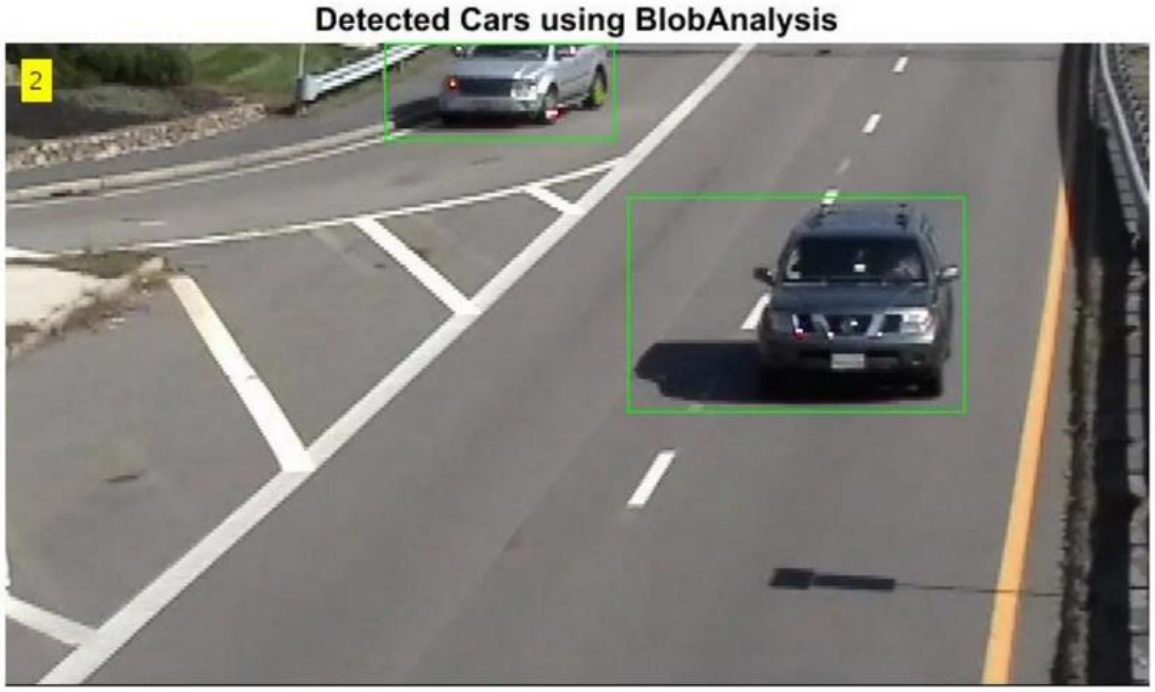
And tonight's topic is "how can we tell if humans are actually intelligent?"

Computer Vision Tasks



Computer Vision Tasks

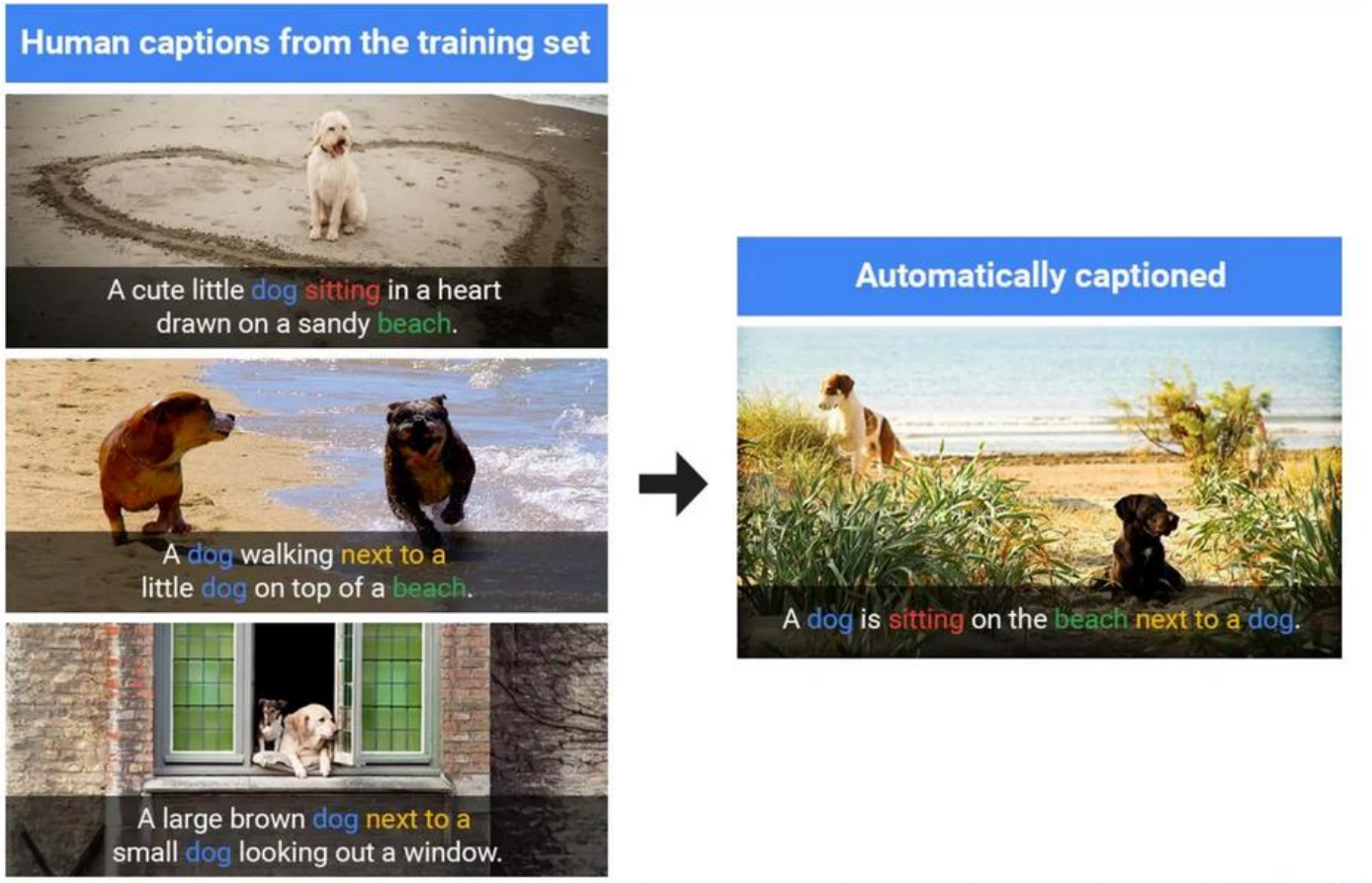
Object Tracking



Scene Parsing

Computer Vision Tasks

Image Captioning



Visual Question Answering

How many Bikes are there ?

What is the number of the Bus ?

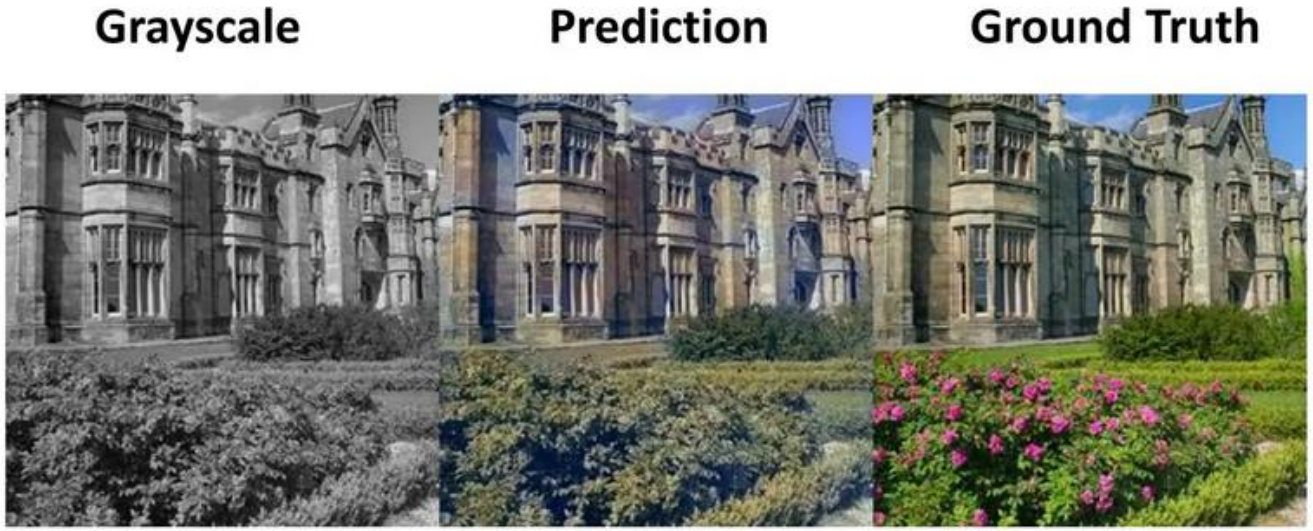
Computer Vision Tasks

Image Generation

orange → apple

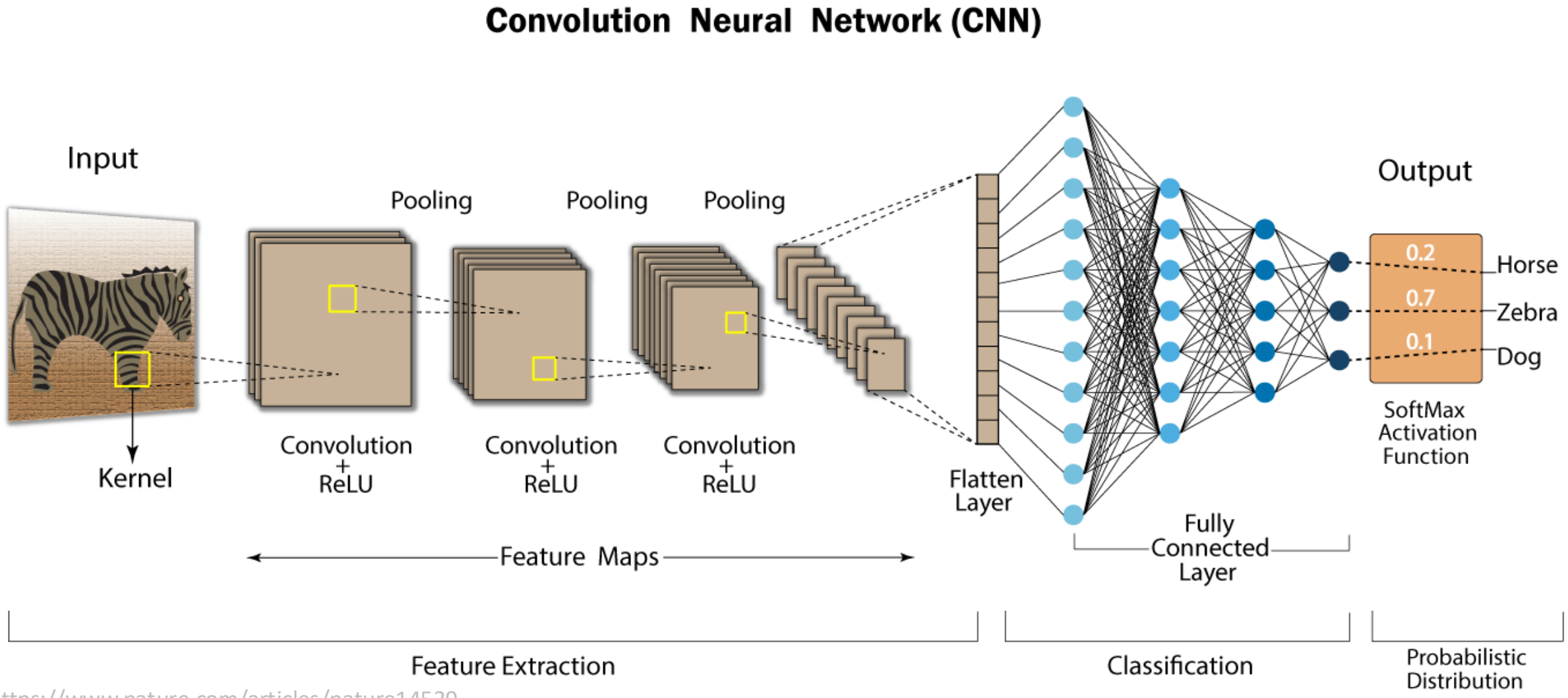
horse → zebra

Image Colorization



How CNN Works ?

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.

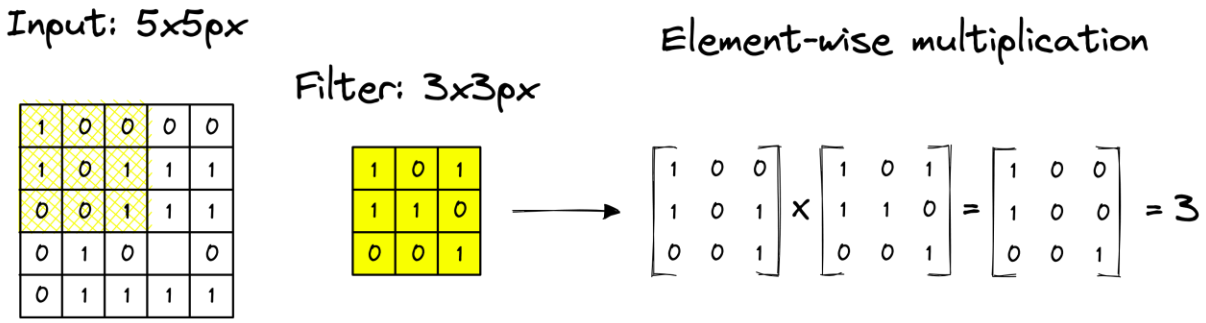


[1] https://www.nature.com/articles/nature14539

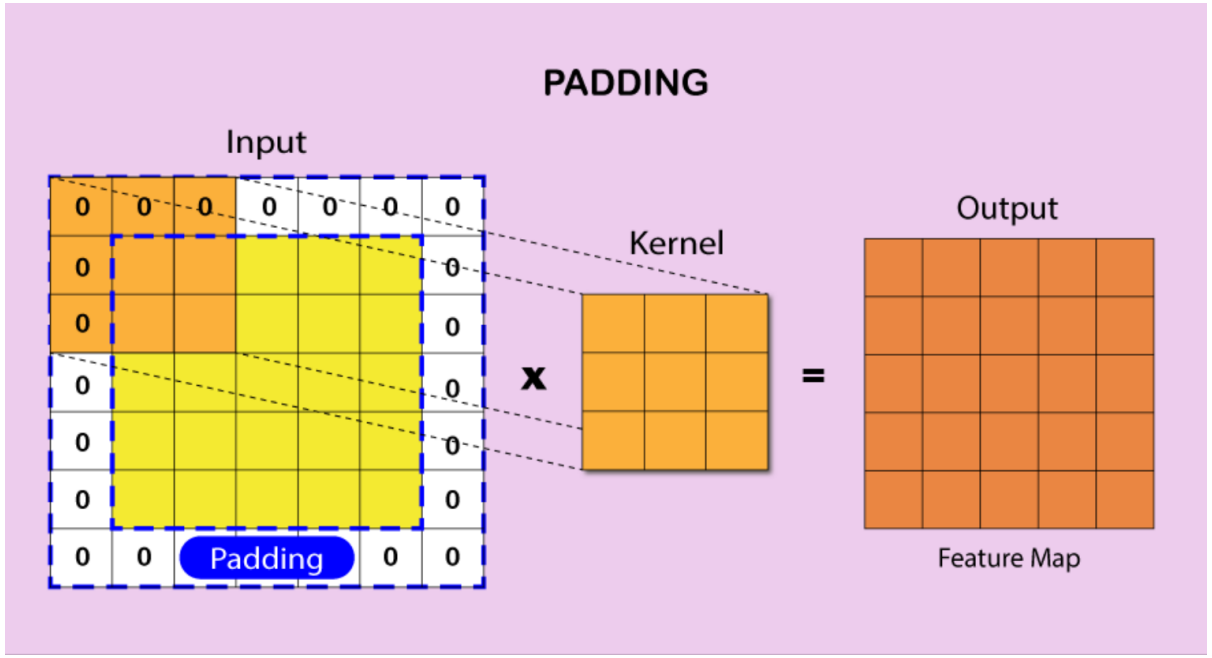
How Does it Works ?

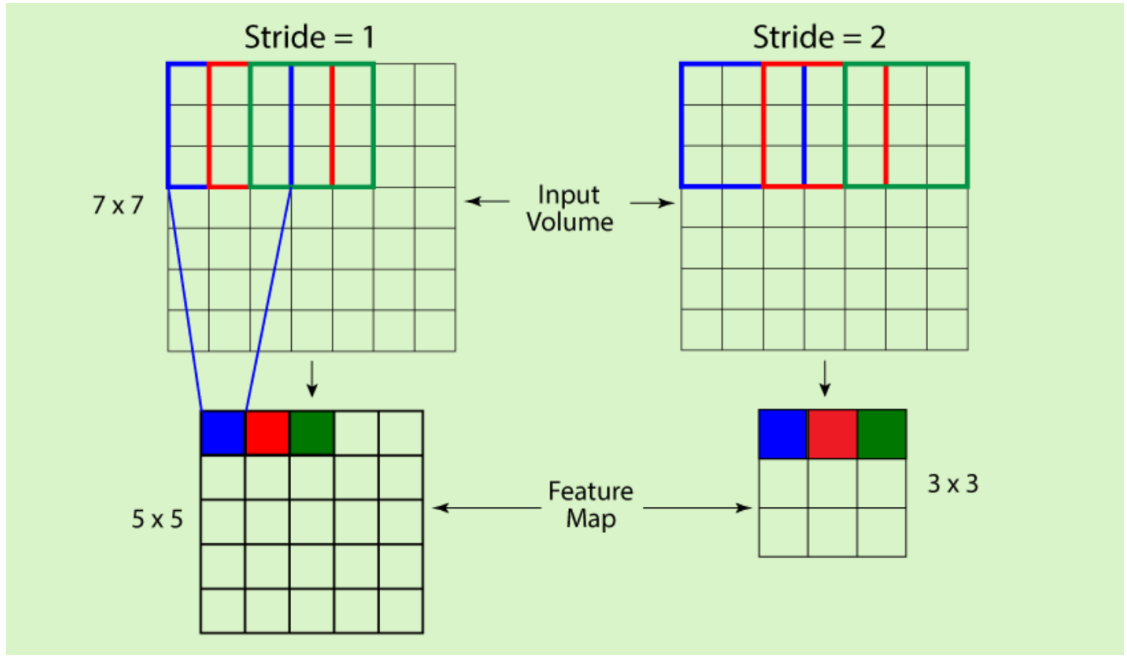
Convolution [2]

Convolution is the process involving combination of two functions that produces the other function as a result. In CNN's, the input image is subjected to convolution with use of filters that produces a **Feature map**.



Local connectivity **Parameter Sharing**





[2] https://developersbreach.com/convolution-neural-network-deep-learning/

How Does it Works ?

Activation Functions

ReLU is computed after convolution. It is most commonly deployed activation function that allows the neural network to account for non-linear relationships.

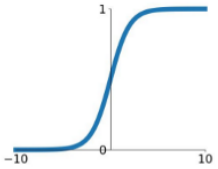
Pooling

Pooling layer operates on each feature map independently. This reduces resolution of the feature map by reducing height and width of features maps, but retains features of the map required for classification. This is called ***Down-sampling***.

Activation Functions

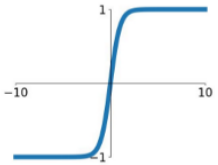
Sigmoid

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



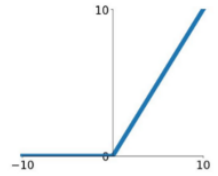
tanh

$\tanh(x)$



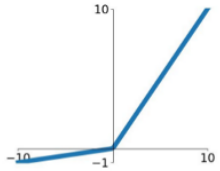
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

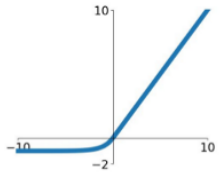


Maxout

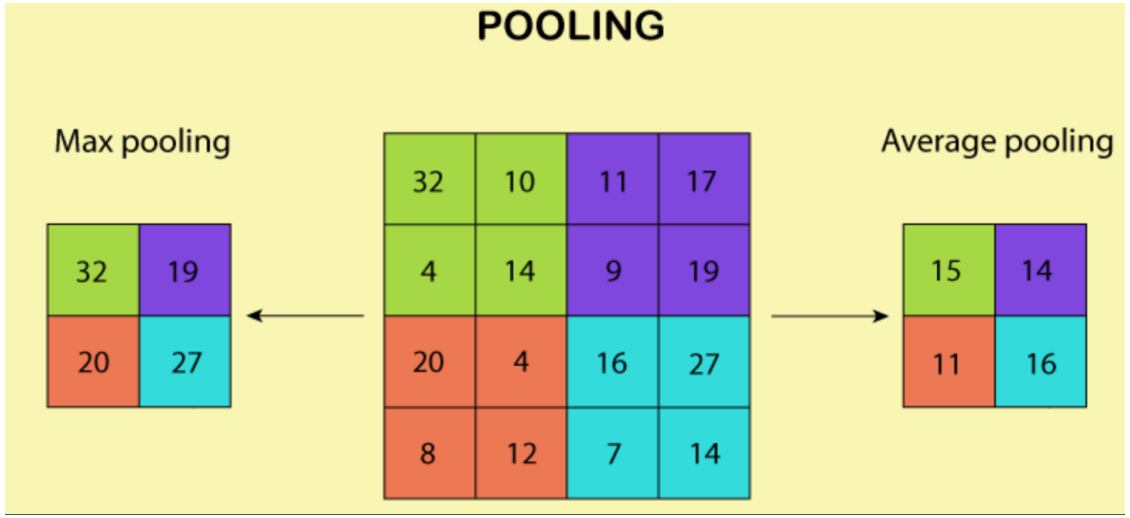
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

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



Max Pooling **Avg. Pooling**



How Does it Works ?

Fully Connected Layer

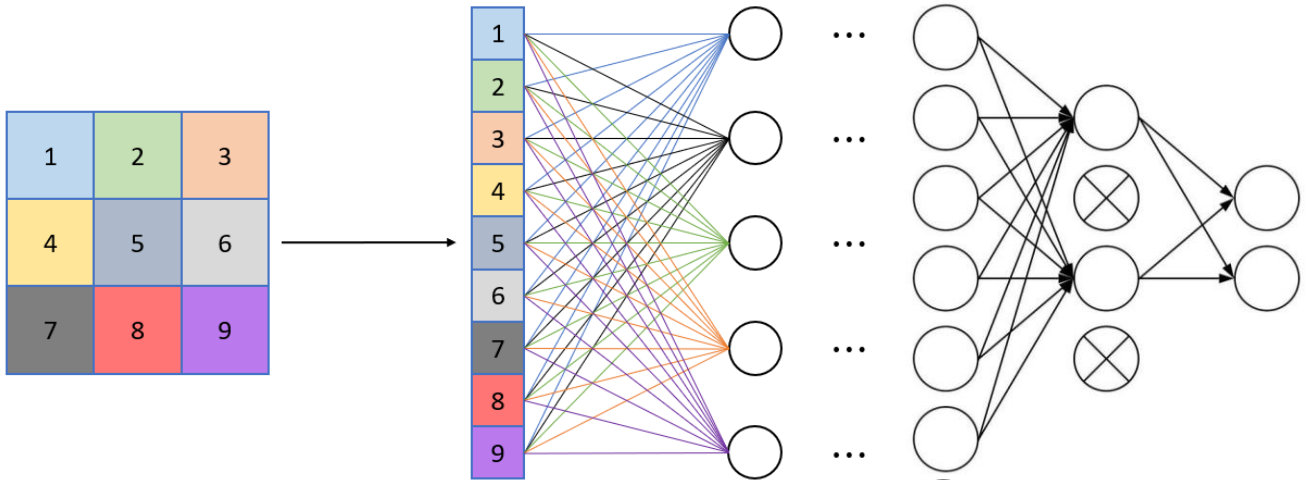
Fully connected layer looks like a regular neural network connecting all neurons and forms the last few layers in the network. The output from flatten layer is fed to this fully-connected layer.

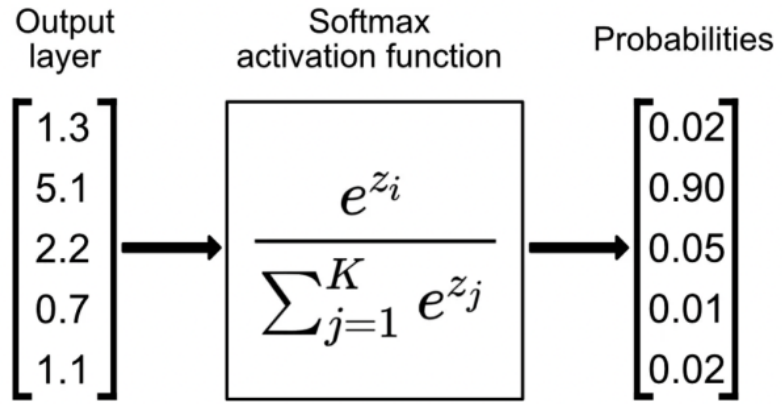
Dropout

Dropout is an approach used for *regularization* in neural networks. It is a technique where randomly chosen nodes are ignored in network during training phase at each stage.

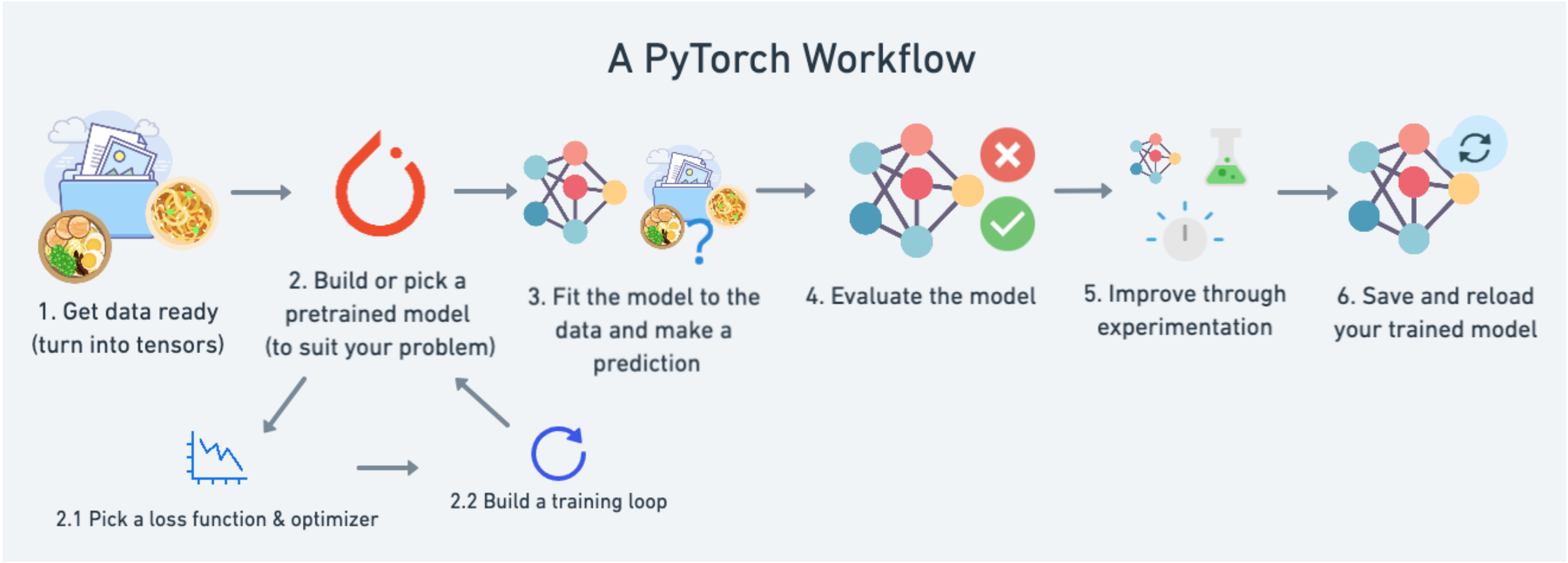
Softmax

Soft-max is an activation layer normally applied to the last layer of network that acts as a classifier.





Model Building

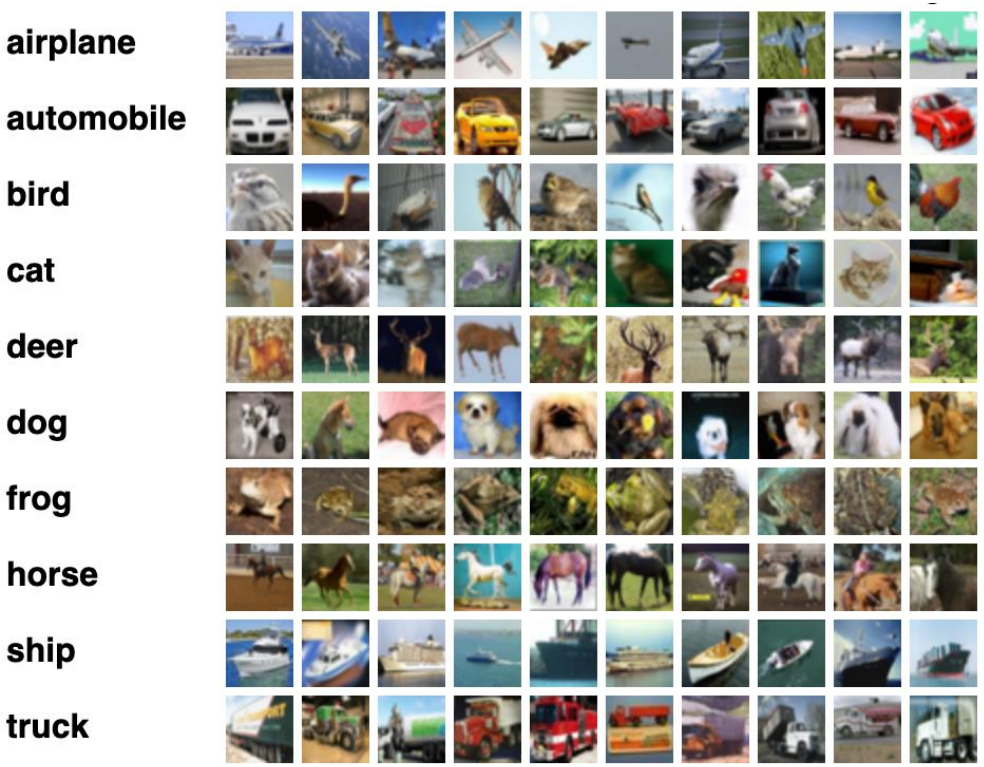


Source: https://www.learnpytorch.io/01_pytorch_workflow/

Dataset: CIFAR-10

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.



Source: https://www.cs.toronto.edu/~kriz/cifar.html

1. Get the data ready

```
TRANSFORMS = ptransforms.Compose([
    transforms.ToTensor(),
    transforms.RandomRotation(30),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter()
])
```

image transformations

```
batch_size = 32
```

```
train = torchvision.datasets.CIFAR10(root="./data",
                                     train=True,
                                     transform=TRANSFORMS,
                                     download=True
                                     )
```

Load the CIFAR-10 training dataset

```
train_dl = DataLoader(train,
                      batch_size=batch_size,
                      shuffle=True,
                      num_workers=0)
```

Get the CIFAR-10 for training

```
classes = train.classes
```

1. Get the data ready

Load the test dataset

```
test = torchvision.datasets.CIFAR10(root="./data",
                                    train=False,
                                    transform=TRANSFORMS,
                                    download=True)
```

```
test_dl = DataLoader(test,
                     batch_size=1,
                     shuffle=False,
                     num_workers=0)
```

Get test dataset ready for testing

```
dataloaders ={'train': train_dl, 'valid': test_dl, 'test': test_dl}
```

Create a dictionary

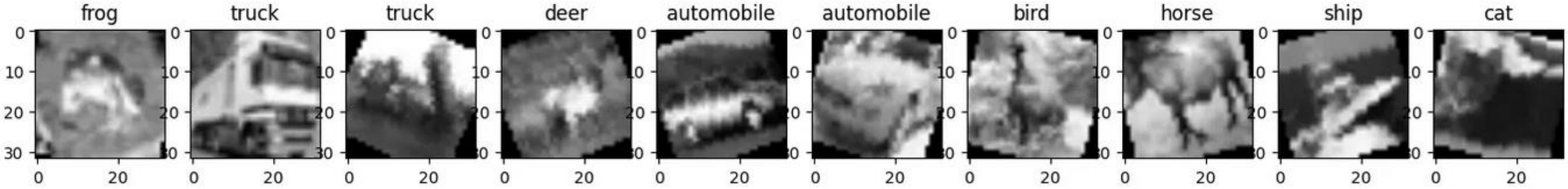
2. Data Visualization

```
N_IMAGES = 10

fig, ax = plt.subplots(1, N_IMAGES, figsize=(17,7))
```

Initialize the subplots

```
for i in range(N_IMAGES):
    im, lbl = train[i]

    ax[i].imshow(im[0,:,:], 'gray', interpolation='bilinear')
    ax[i].set_title(f'{classes[lbl]}')
```

Plot for 10 images



3. Model building

```
class CustomCNN(nn.Module):
 def __init__(self):
  super(CustomCNN, self).__init__()

  self.layer1 = self.ConvModule(in_features=3, out_features=64)        #16,16
  self.layer2 = self.ConvModule(in_features=64, out_features=128)      #8,8
  self.layer3 = self.ConvModule(in_features=128, out_features=256)     #4,4
  self.layer4 = self.ConvModule(in_features=256, out_features=512)     #2,2

  self.classifier = nn.Sequential(nn.Flatten(),
   nn.Linear(2*2*512, 1024),
   nn.ReLU(),
   nn.Linear(1024, 512),
   nn.ReLU(),
   nn.Linear(512,10),
   nn.Softmax())
```

3. Model building

```
def forward(self, x):
 x = self.layer1(x)
 x = self.layer2(x)
 x = self.layer3(x)
 x = self.layer4(x)
 x = self.classifier(x)
 return x

def ConvModule(self, in_features, out_features):
 return nn.Sequential(nn.Conv2d(in_channels=in_features,
.out_channels=out_features, kernel_size=3, padding=1),
 nn.BatchNorm2d(out_features),
 nn.ReLU(),
 nn.MaxPool2d(2,2)
)

model = CustomCNN().to(device)
```

3. Model building

```
torchsummary.summary(model, (3,32,32))
```

Number of parameters for convolution

N = (3*3*3+1)*64

N = (Number of channels in i^{th} layer
* filter size + 1) * Number of channels in $i+1^{th}$ layer

Number of parameters for batchnorm

N = 128 *2

N = 256

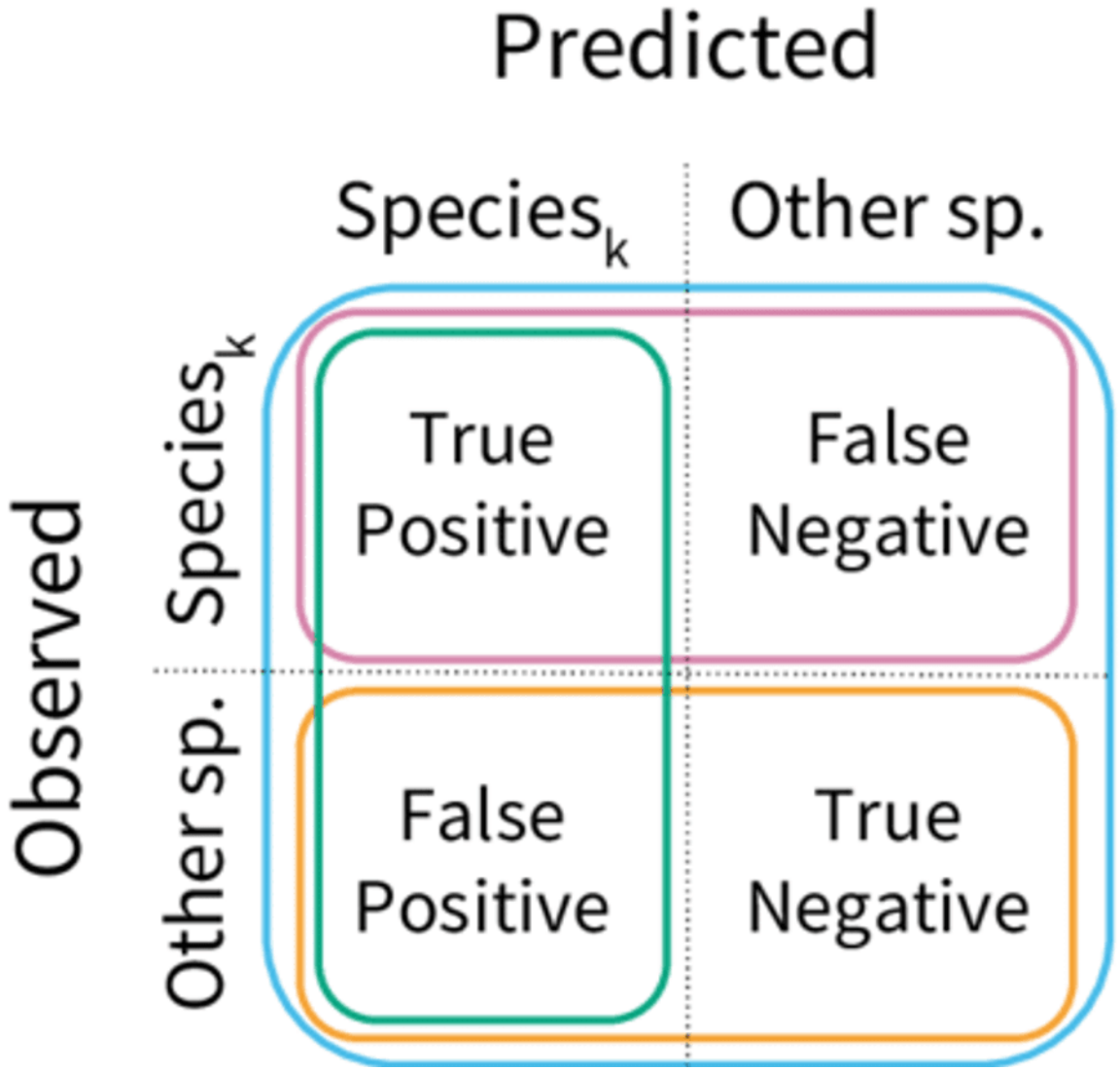
```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
       BatchNorm2d-2           [-1, 64, 32, 32]             128
              ReLU-3           [-1, 64, 32, 32]               0
         MaxPool2d-4           [-1, 64, 16, 16]               0
            Conv2d-5          [-1, 128, 16, 16]          73,856
       BatchNorm2d-6          [-1, 128, 16, 16]             256
              ReLU-7          [-1, 128, 16, 16]               0
         MaxPool2d-8            [-1, 128, 8, 8]               0
            Conv2d-9            [-1, 256, 8, 8]         295,168
      BatchNorm2d-10            [-1, 256, 8, 8]             512
             ReLU-11            [-1, 256, 8, 8]               0
        MaxPool2d-12            [-1, 256, 4, 4]               0
           Conv2d-13            [-1, 512, 4, 4]       1,180,160
      BatchNorm2d-14            [-1, 512, 4, 4]           1,024
             ReLU-15            [-1, 512, 4, 4]               0
        MaxPool2d-16            [-1, 512, 2, 2]               0
          Flatten-17                 [-1, 2048]               0
           Linear-18                 [-1, 1024]       2,098,176
             ReLU-19                 [-1, 1024]               0
           Linear-20                  [-1, 512]         524,800
             ReLU-21                  [-1, 512]               0
           Linear-22                   [-1, 10]           5,130
          Softmax-23                   [-1, 10]               0
================================================================
Total params: 4,181,002
Trainable params: 4,181,002
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 3.09
Params size (MB): 15.95
Estimated Total Size (MB): 19.05
```

3. Model building

Function	What does it do?	Where does it live in PyTorch?	Common values
Loss function	Measures how wrong your model's predictions (e.g. `y_preds`) are compared to the truth labels (e.g. `y_test`). Lower the better.	PyTorch has plenty of built-in loss functions in `torch.nn`.	Mean absolute error (MAE) for regression problems (`torch.nn.L1Loss()`). Binary cross entropy for binary classification problems (`torch.nn.BCELoss()`).
Optimizer	Tells your model how to update its internal parameters to best lower the loss.	You can find various optimization function implementations in `torch.optim`.	Stochastic gradient descent (`torch.optim.SGD()`). Adam optimizer (`torch.optim.Adam()`).

```
learning_rate = 0.001
num_epochs = 35
criterion = nn.CrossEntropyLoss()
optimizer= torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

Classification Metrics



$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

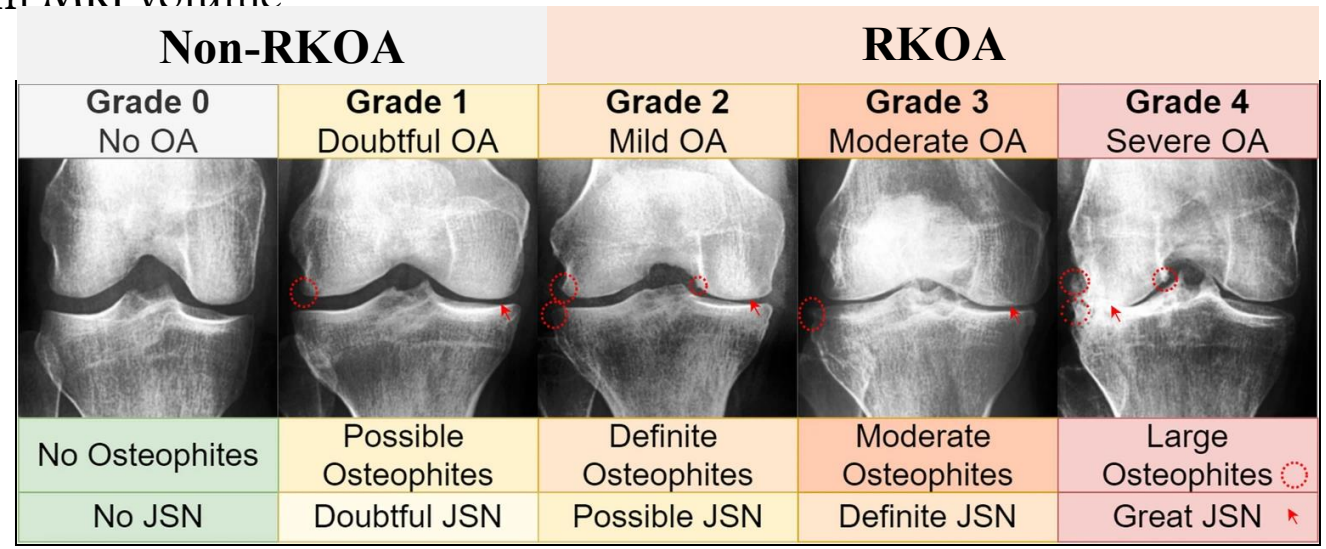
$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

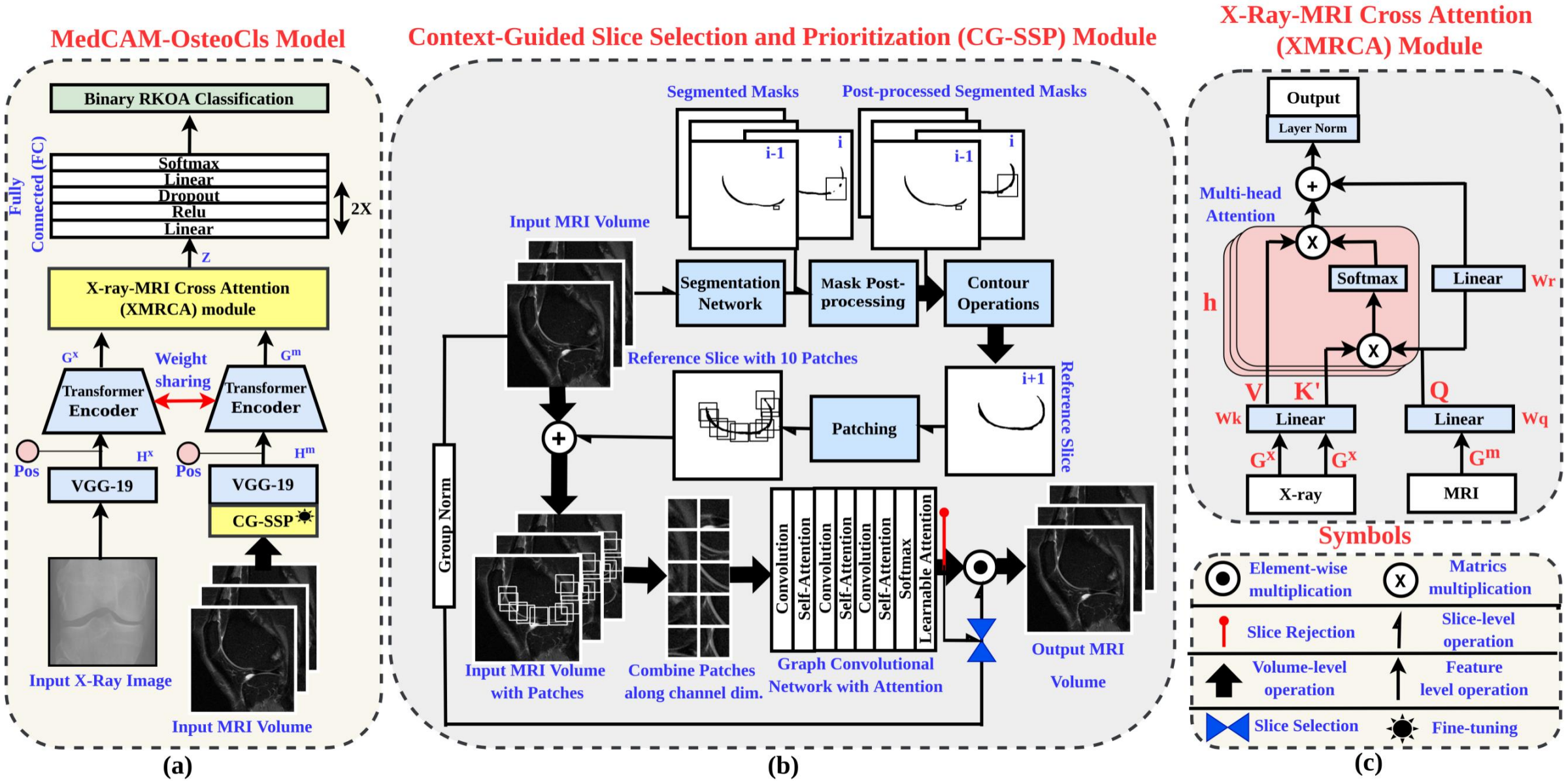
MedCAM-OsteoCls: Medical Context Aware Multimodal Classification of Knee Osteoarthritis

1. Heterogeneity in KL grading: Less compactness within feature clusters
2. Distinct nature of diagnostic modalities: Hard to determine nature and location at which information exchange can take place across modalities
3. Irregularities in number of scans per MRI volume: Difficulties in training due to sparse anatomical representation in MRI volume



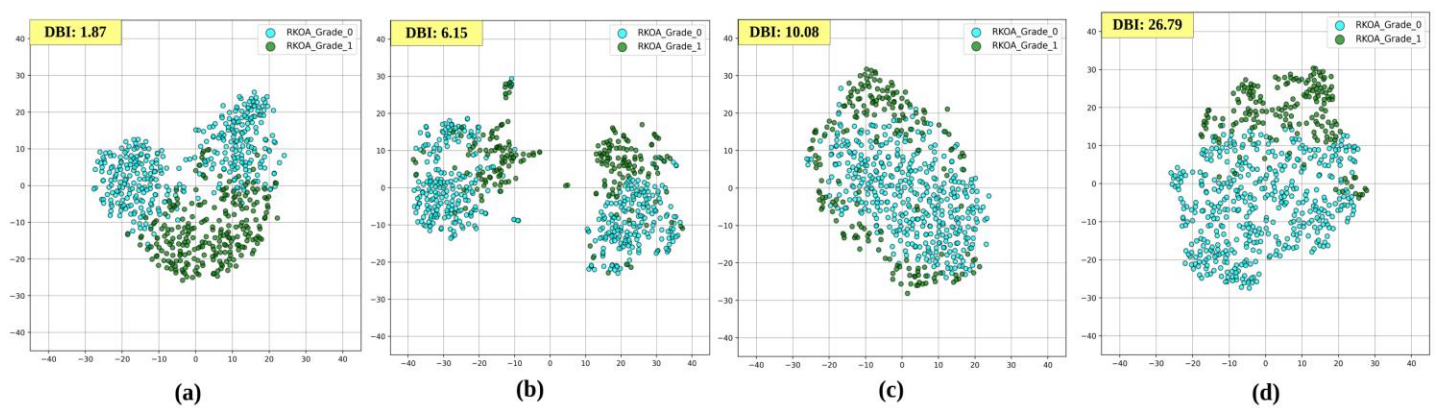
Daydar, Akshay, et al. "MedCAM-OsteoCls: Medical Context Aware Multimodal Classification of Knee Osteoarthritis." *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2025.

MedCAM-OsteoCls: Medical Context Aware Multimodal Classification of Knee Osteoarthritis

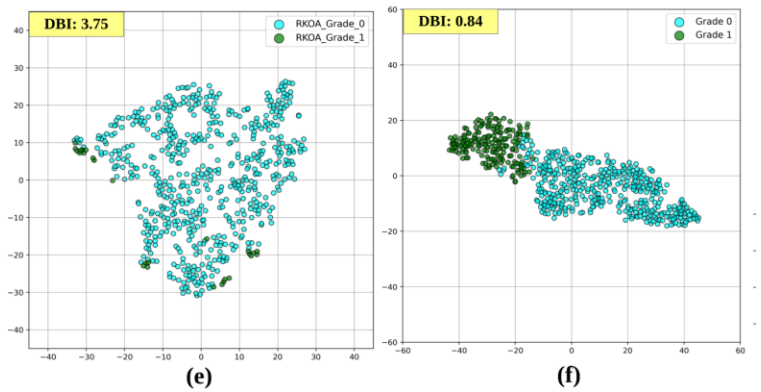


MedCAM-OsteoCls: Medical Context Aware Multimodal Classification of Knee Osteoarthritis

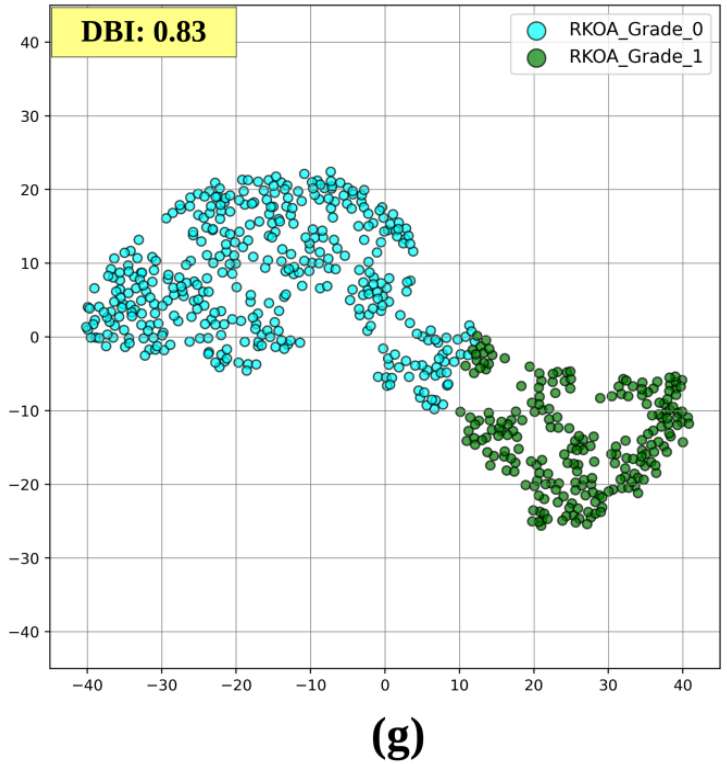
Unimodal



Multimodal



MedCAM-OsteoCls



Akshay Daydar

Research Scholar, IITG

Research Areas: Medical Imaging, Deep Learning, Biomechanics

https://adaydar.github.io/