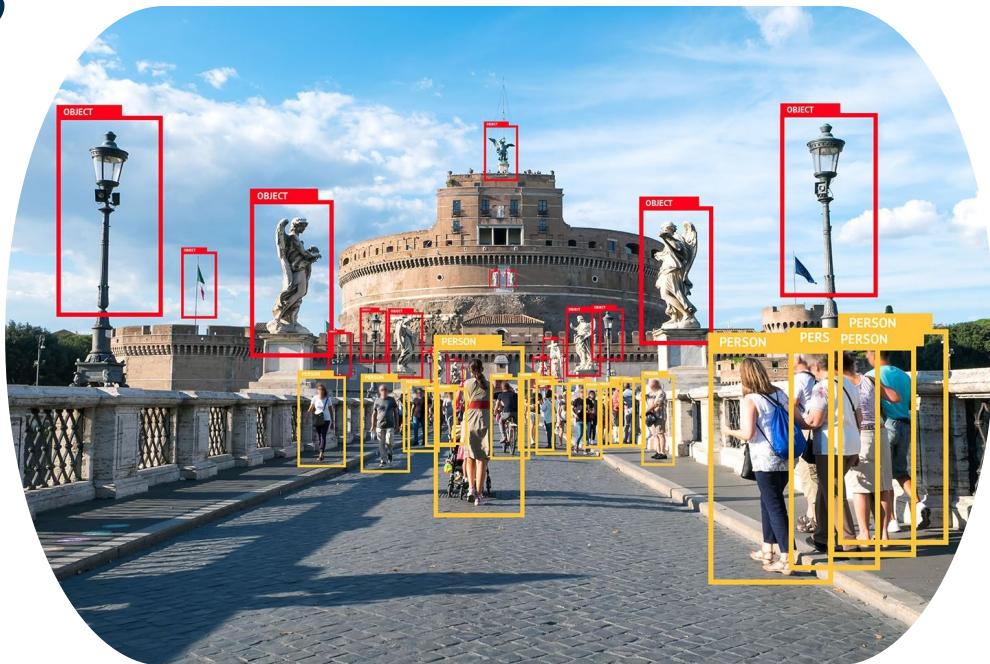


Detection: Understand, locate and classify

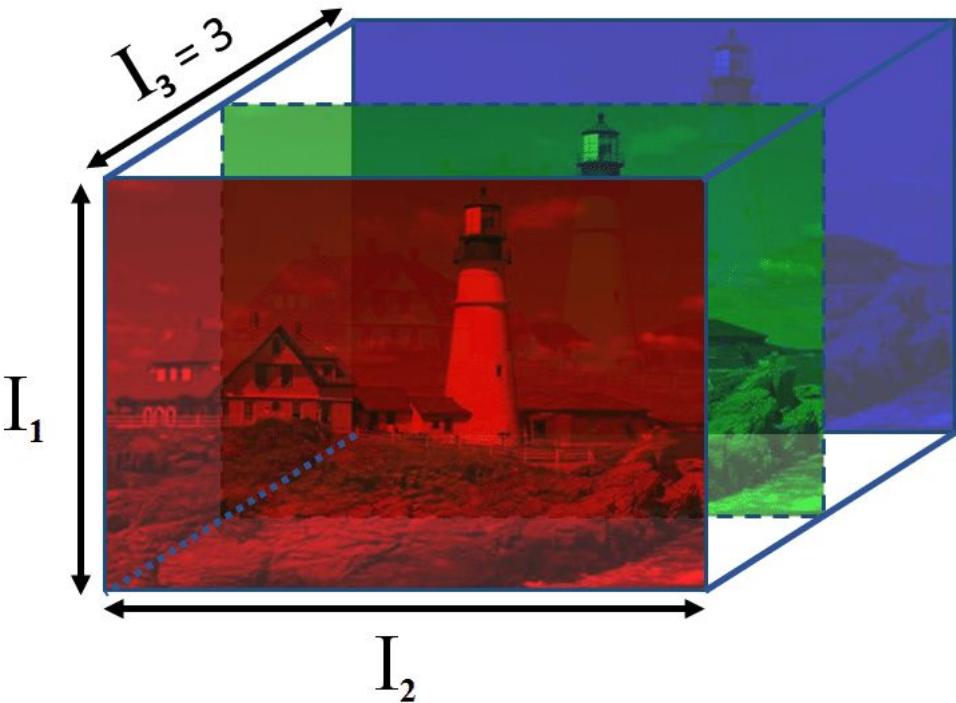
Luis Cossio

Master in Engineering sciences, mention
in Electrical Engineering



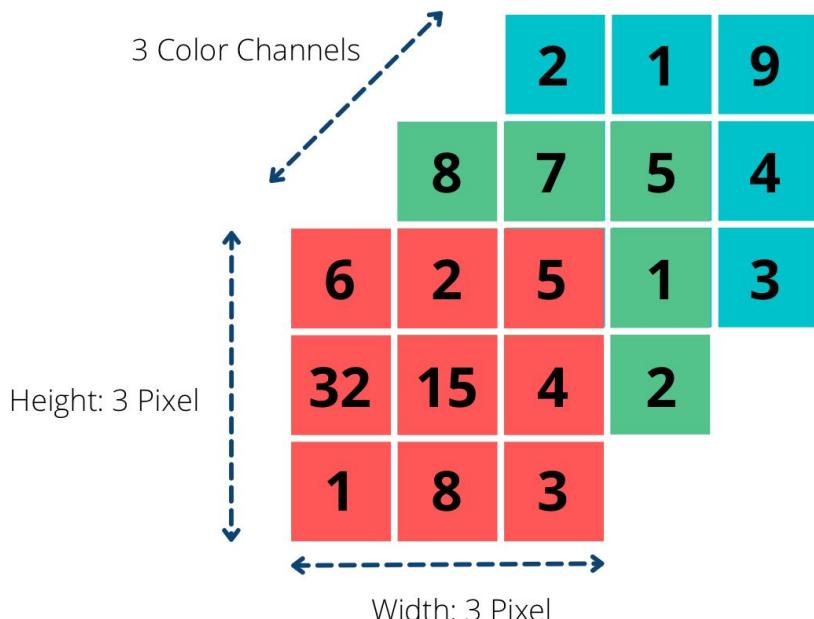
Images

- As computational objects, images are represented as 3 matrices of color.
 - Each matrix/channel represent the intensity of a color.
 - Representation Red, Green and Blue (RGB)



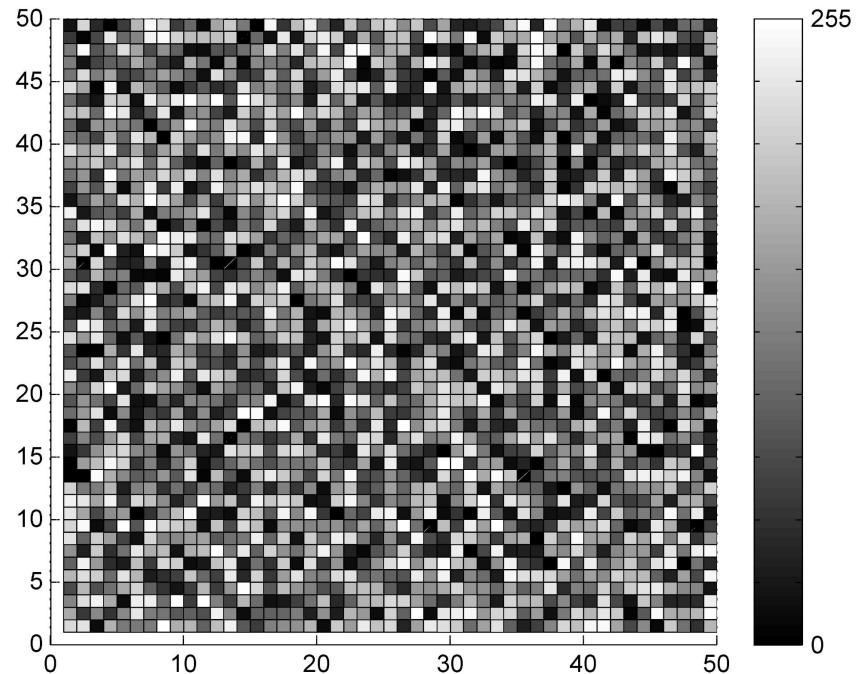
Images

- As computational objects, images are represented as 3 matrices of color.
 - Each matrix/channel represent the intensity of a color.
 - Representation Red, Green and Blue (RGB)
 - Individual pixel have values in a given range
 - Unsigned Int scale: [0,255]
 - Float scale: [0.0,1.0]
 - Each value in the image can be accessed by a triplet of indices (i,j,k)

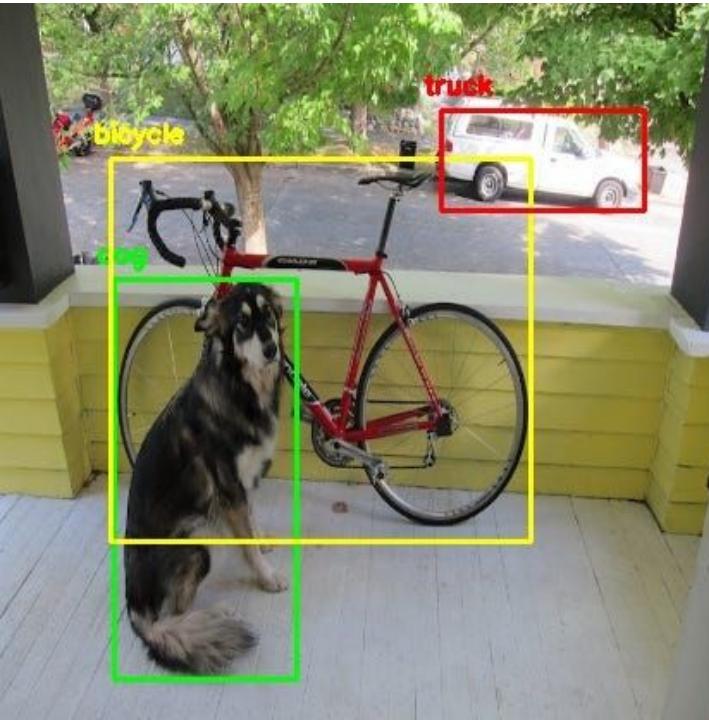


Images

- As computational objects, images are represented as 3 matrices of color.
 - Each matrix/channel represent the intensity of a color.
 - Representation Red, Green and Blue (RGB)
 - Individual pixel have values in a given range
 - Unsigned Int scale: [0,255]
 - Float scale: [0.0,1.0]
 - Each value in the image can be accessed by a triplet of indices (i,j,k)
 - Lower values represent darker colors, while higher intensity represent light color



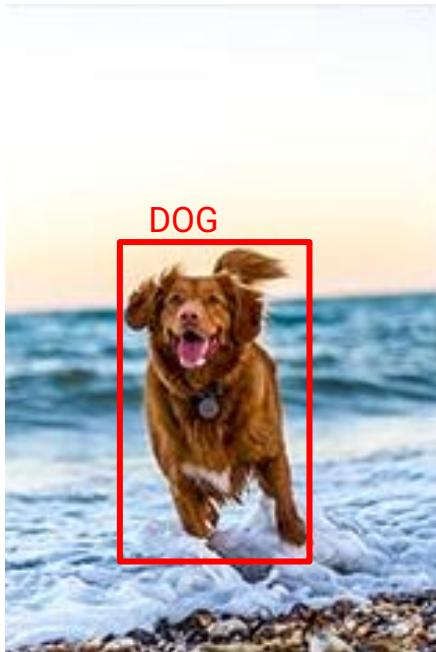
LOH Detection



- Detection consist of 3 tasks
 - Separating target objects from background
 - Locating objects
 - Often using a Bounding box
 - Classifying objects in each class



- Detection consist of 3 tasks
 - Separating target objects from background
 - Locating objects
 - Often using a Bounding box
 - Classifying objects in each class
- Very simple task
 - The objects are in plain sight!!!



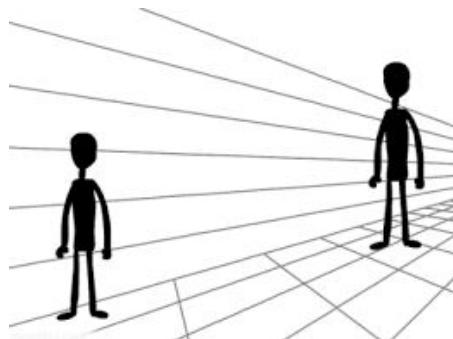
- Detection consist of 3 tasks
 - Separating target objects from background
 - Locating objects
 - Often using a Bounding box
 - Classifying objects in each class
- Very simple task
 - The objects are in plain sight!!!

LOH Detection



- Detection consist of 3 tasks
 - Separating target objects from background
 - Locating objects
 - Often using a Bounding box
 - Classifying objects in each class
- Very simple task
 - The objects are in plain sight!!!
- What differentiate and its not:
 - Its color

UH Detection



- Detection consist of 3 tasks
 - Separating target objects from background
 - Locating objects
 - Often using a Bounding box
 - Classifying objects in each class
- Very simple task
 - The objects are in plain sight!!!
- What differentiate objects its not:
 - Its color
 - Its size

LOH Detection



- Detection consist of 3 tasks
 - Separating target objects from background
 - Locating objects
 - Often using a Bounding box
 - Classifying objects in each class
- Very simple task
 - The objects are in plain sight!!!
- What differentiate objects its not:
 - Its color
 - Its size

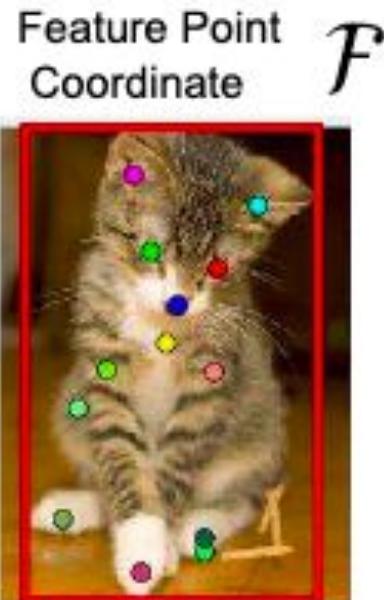
LOH Detection



- Detection consist of 3 tasks
 - Separating target objects from background
 - Locating objects
 - Often using a Bounding box
 - Classifying objects in each class
- Very simple task
 - The objects are in plain sight!!!
- What differentiate objects its not:
 - Its color
 - Its size

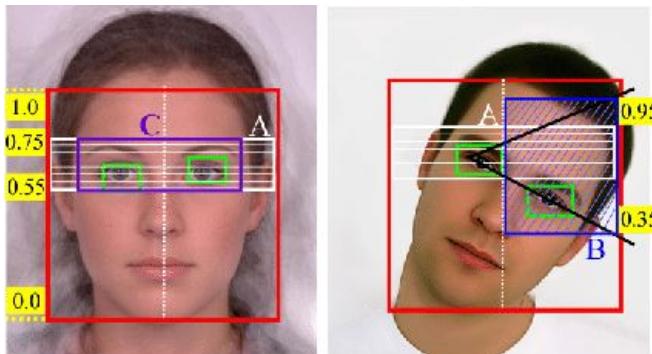


- Detection consist of 3 tasks
 - Separating target objects from background
 - Locating objects
 - Often using a Bounding box
 - Classifying objects in each class
- Very simple task
 - The objects are in plain sight!!!
- What differentiate objects its not:
 - Its color
 - Its size
 - its pose



- Detection consist of 3 tasks
 - Separating target objects from background
 - Locating objects
 - Often using a Bounding box
 - Classifying objects in each class
- Very simple task
 - The objects are in plain sight!!!
- What differentiate objects its not:
 - Its color
 - Its size
 - its pose
- What matter it's the overall pattern of all the pixels in the object
 - Certain structures are easily identifiable

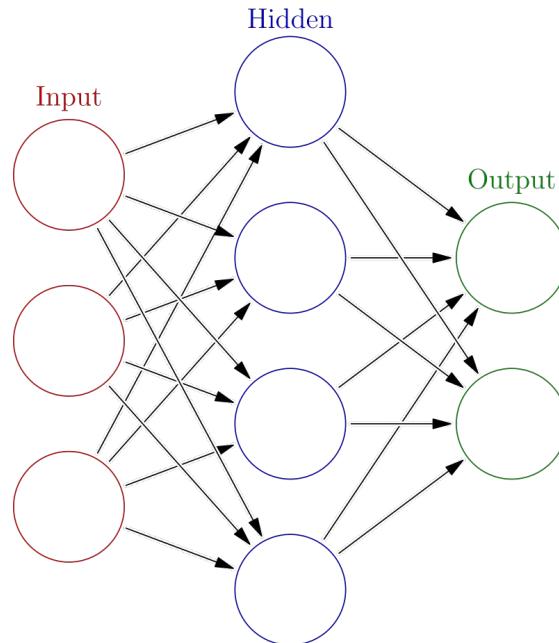
LOH Detection



- Detection consist of 3 tasks
 - Separating target objects from background
 - Locating objects
 - Often using a Bounding box
 - Classifying objects in each class
- Very simple task
 - The objects are in plain sight!!!
- What differentiate objects its not:
 - Its color
 - Its size
 - its pose
- What matter it's the overall pattern of all the pixels in the object
 - Certain structures are easily identifiable

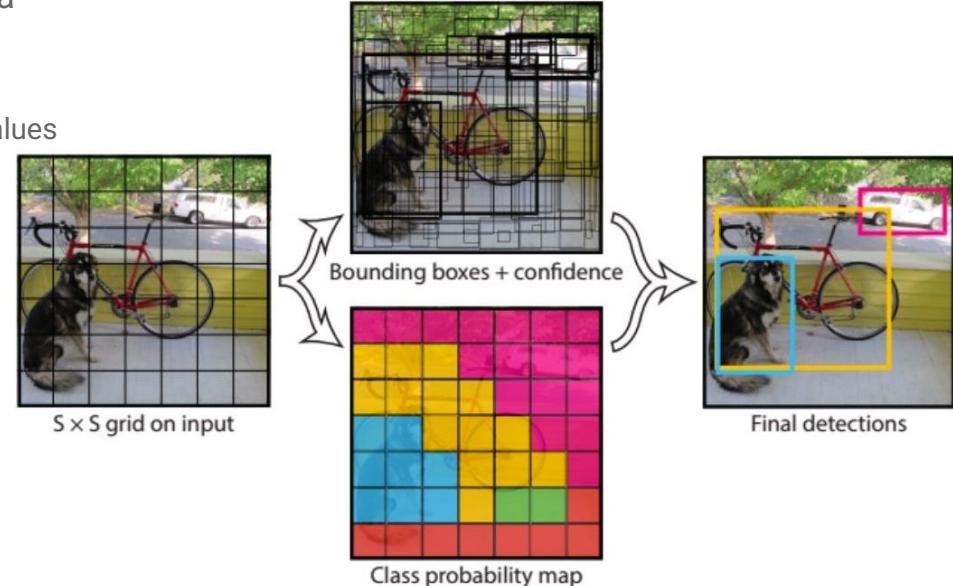
Neural Networks

- Neural networks are algorithms with trainable weights/parameters
 - Weights are jointly optimized to solve a task.
 - Each weight is a numeric value which is part of mathematical operations.
 - Sum
 - Concatenation
 - Pooling
 - Sigmoid
 - Convolution
 - Self-attention
 - etc



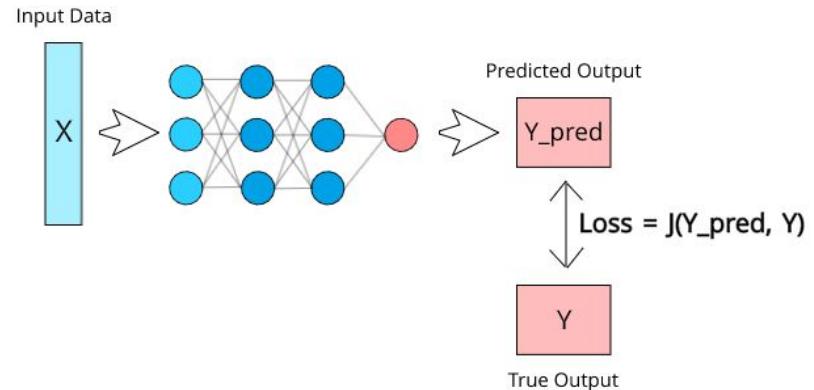
Object detection using Neural Networks

- Neural networks for object detection will work in a simple manner
 - a. Receive an image
 - b. Will produce several predictions, composed of 3 values
 - Confidence on the prediction [0,1]
 - Location of the object [x, y, width, height]
 - Class of the object



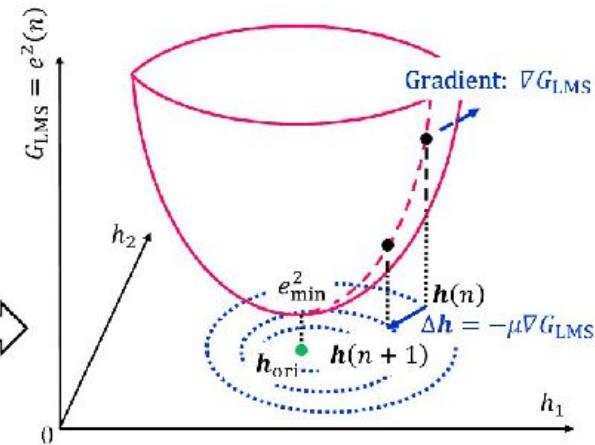
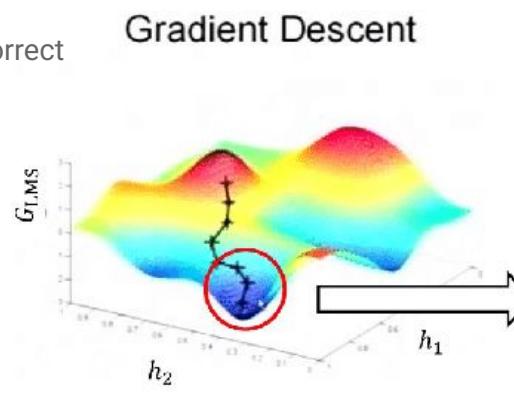
Training a neural Network

- Training a neural networks for object detection will work in a simple manner
 - a. Receive an image
 - b. Will produce several predictions, composed of 3 values
 - Confidence on the prediction [0,1]
 - Location of the object [x, y, width, height]
 - Class of the object
 - c. Evaluate the prediction with labels/target/correct values



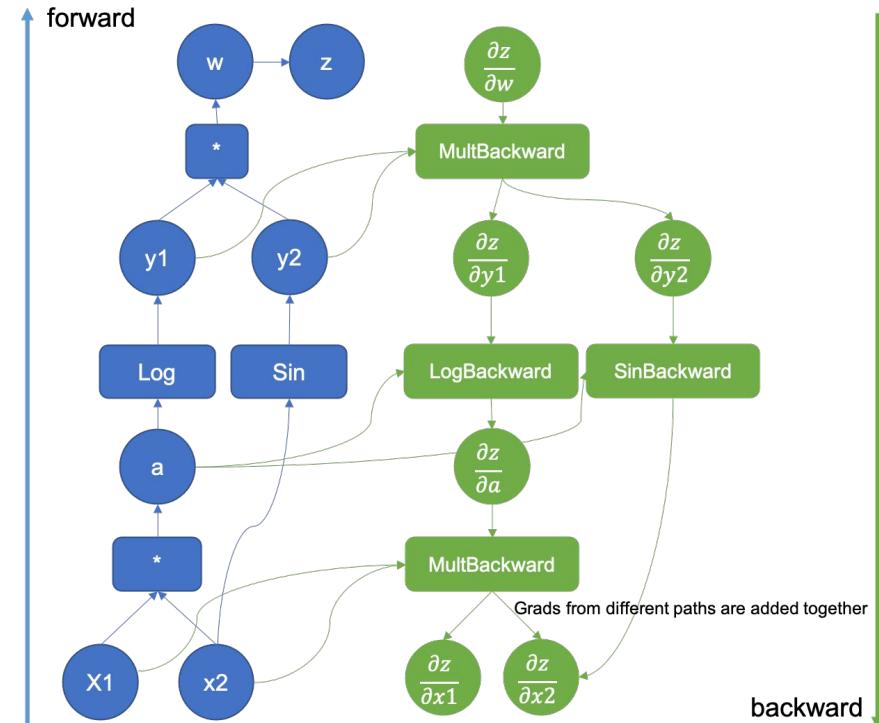
Training a neural Network

- Training a neural networks for object detection will work in a simple manner
 - a. Receive an image
 - b. Will produce several predictions, composed of 3 values
 - Confidence on the prediction [0,1]
 - Location of the object [x, y, width, height]
 - Class of the object
 - c. Evaluate the prediction with labels/target/correct values
 - d. Backpropagation of the error
 - Gradient descent



Training a neural Network

- Training a neural networks for object detection will work in a simple manner
 - a. Receive an image
 - b. Will produce several predictions, composed of 3 values
 - Confidence on the prediction [0,1]
 - Location of the object [x, y, width, height]
 - Class of the object
 - c. Evaluate the prediction with labels/target/correct values
 - d. Backpropagation of the error
 - Gradient descent



Training Cycle



1. Pack multiple samples into a package/batche

```
for i in range(epochs):
    for (img,label) in dataloader:
        prediction = model(img)

        boxes_pred = prediction['boxes']
        classes_pred = prediction['classes']
        confidence_pred = prediction['confidence']

        loss_box, matches = compute_loss_box(label['boxes'],boxes_pred)
        loss_classification = compute_loss_class(matches, label['classes'],classes_pred)
        loss_confidence = compute_loss_conf(matches, confidence_pred)

        total_loss = loss_box + loss_classification + loss_confidence
        total_loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Training Cycle



1. Pack multiple samples into a package/batche
2. Pass them to the model

```
for i in range(epochs):
    for (img,label) in dataloader:
        prediction = model(img)

        boxes_pred = prediction['boxes']
        classes_pred = prediction['classes']
        confidence_pred = prediction['confidence']

        loss_box, matches = compute_loss_box(label['boxes'],boxes_pred)
        loss_classification = compute_loss_class(matches, label['classes'],classes_pred)
        loss_confidence = compute_loss_conf(matches, confidence_pred)

        total_loss = loss_box + loss_classification + loss_confidence
        total_loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Training Cycle



1. Pack multiple samples into a package/batche
2. Pass them to the model
3. Calculate the loss

```
for i in range(epochs):
    for (img,label) in dataloader:
        prediction = model(img)

        boxes_pred = prediction['boxes']
        classes_pred = prediction['classes']
        confidence_pred = prediction['confidence']

        loss_box, matches = compute_loss_box(label['boxes'],boxes_pred)
        loss_classification = compute_loss_class(matches, label['classes'],classes_pred)
        loss_confidence = compute_loss_conf(matches, confidence_pred)

        total_loss = loss_box + loss_classification + loss_confidence
        total_loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Training Cycle

1. Pack multiple samples into a package/batche
2. Pass them to the model
3. Calculate the loss
4. Calculate gradient

```
for i in range(epochs):
    for (img,label) in dataloader:
        prediction = model(img)

        boxes_pred = prediction['boxes']
        classes_pred = prediction['classes']
        confidence_pred = prediction['confidence']

        loss_box, matches = compute_loss_box(label['boxes'],boxes_pred)
        loss_classification = compute_loss_class(matches, label['classes'],classes_pred)
        loss_confidence = compute_loss_conf(matches, confidence_pred)

        total_loss = loss_box + loss_classification + loss_confidence
        total_loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Training Cycle



1. Pack multiple samples into a package/batche
2. Pass them to the model
3. Calculate the loss
4. Calculate gradient
5. Update weights

```
for i in range(epochs):
    for (img,label) in dataloader:
        prediction = model(img)

        boxes_pred = prediction['boxes']
        classes_pred = prediction['classes']
        confidence_pred = prediction['confidence']

        loss_box, matches = compute_loss_box(label['boxes'],boxes_pred)
        loss_classification = compute_loss_class(matches, label['classes'],classes_pred)
        loss_confidence = compute_loss_conf(matches, confidence_pred)

        total_loss = loss_box + loss_classification + loss_confidence
        total_loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Training Cycle



1. Pack multiple samples into a package/batche
2. Pass them to the model
3. Calculate the loss
4. Calculate gradient
5. Update weights
6. Set gradients to 0 for the next iteration

```
for i in range(epochs):
    for (img,label) in dataloader:
        prediction = model(img)

        boxes_pred = prediction['boxes']
        classes_pred = prediction['classes']
        confidence_pred = prediction['confidence']

        loss_box, matches = compute_loss_box(label['boxes'],boxes_pred)
        loss_classification = compute_loss_class(matches, label['classes'],classes_pred)
        loss_confidence = compute_loss_conf(matches, confidence_pred)

        total_loss = loss_box + loss_classification + loss_confidence
        total_loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Training Cycle



1. Pack multiple samples into a package/batche
2. Pass them to the model
3. Calculate the loss
4. Calculate gradient
5. Update weights
6. Set gradients to 0 for the next iteration
7. Repeat for every element in the dataset

```
for i in range(epochs):
    for (img,label) in dataloader:
        prediction = model(img)

        boxes_pred = prediction['boxes']
        classes_pred = prediction['classes']
        confidence_pred = prediction['confidence']

        loss_box, matches = compute_loss_box(label['boxes'],boxes_pred)
        loss_classification = compute_loss_class(matches, label['classes'],classes_pred)
        loss_confidence = compute_loss_conf(matches, confidence_pred)

        total_loss = loss_box + loss_classification + loss_confidence
        total_loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Training Cycle



1. Pack multiple samples into a package/batche
2. Pass them to the model
3. Calculate the loss
4. Calculate gradient
5. Update weights
6. Set gradients to 0 for the next iteration
7. Repeat for every element in the dataset
8. Repeat N times to further improve performance.
Each iteration is known as an epoch

```
for i in range(epochs):
    for (img,label) in dataloader:
        prediction = model(img)

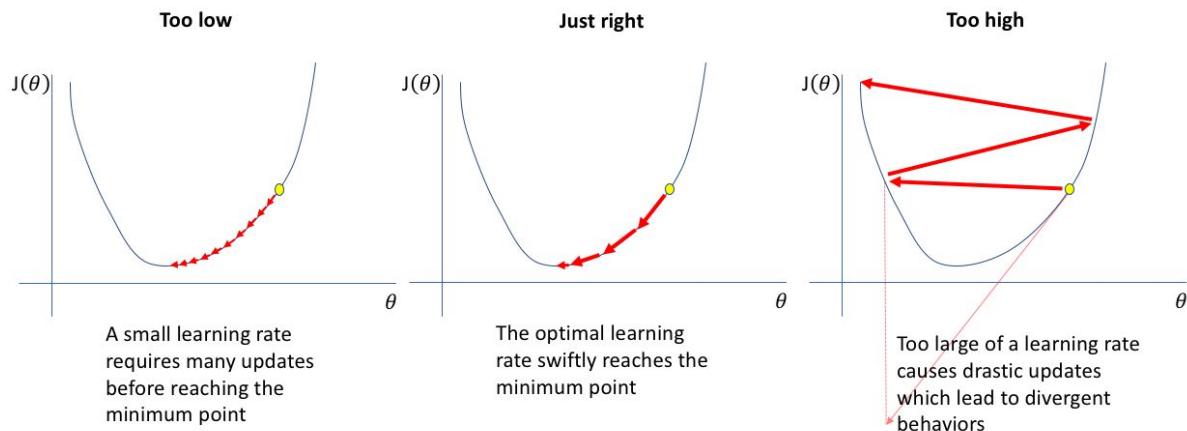
        boxes_pred = prediction['boxes']
        classes_pred = prediction['classes']
        confidence_pred = prediction['confidence']

        loss_box, matches = compute_loss_box(label['boxes'],boxes_pred)
        loss_classification = compute_loss_class(matches, label['classes'],classes_pred)
        loss_confidence = compute_loss_conf(matches, confidence_pred)

        total_loss = loss_box + loss_classification + loss_confidence
        total_loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Parameters training

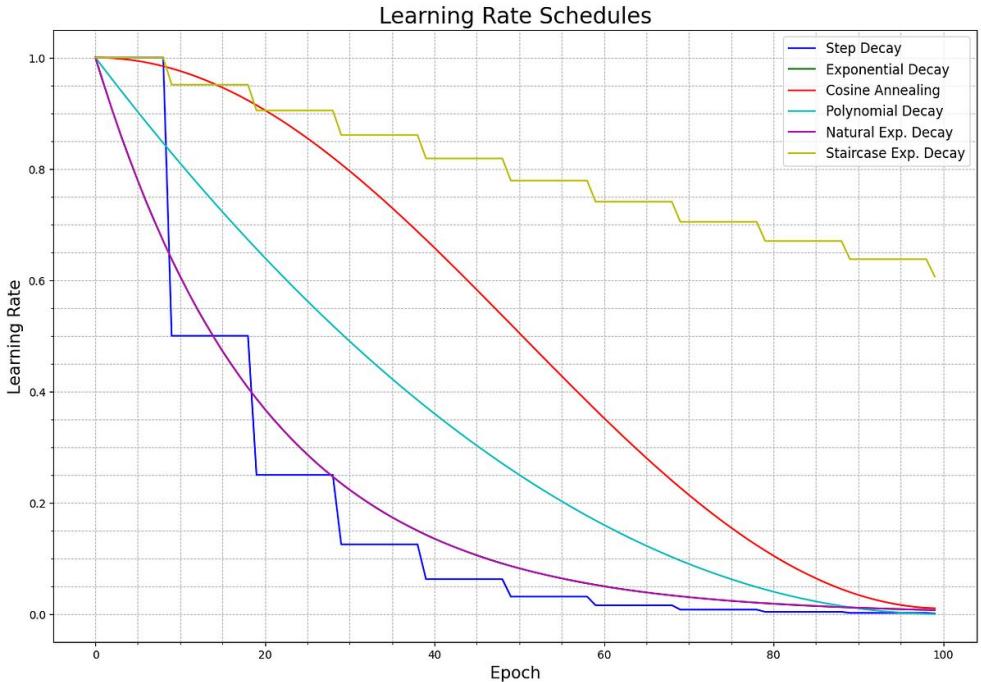
- Learning-rate
 - Learning rate scheduler



Parameters training

UH

- Learning-rate
 - Learning rate scheduler



Parameters training

- Learning-rate
 - Learning rate scheduler
- Resolution



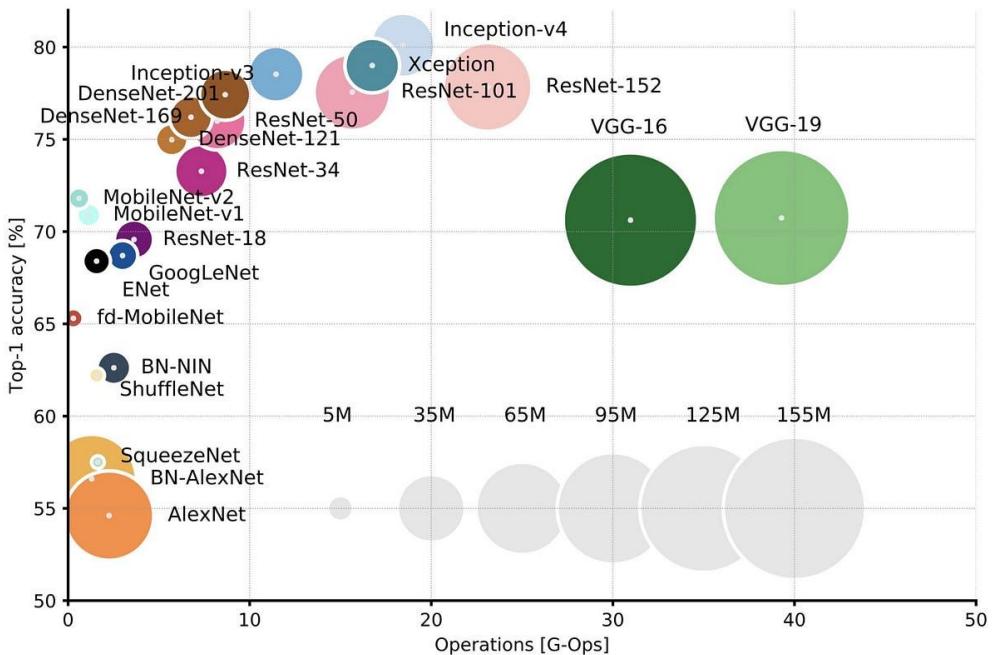
Parameters training

- Learning-rate
 - Learning rate scheduler
- Resolution



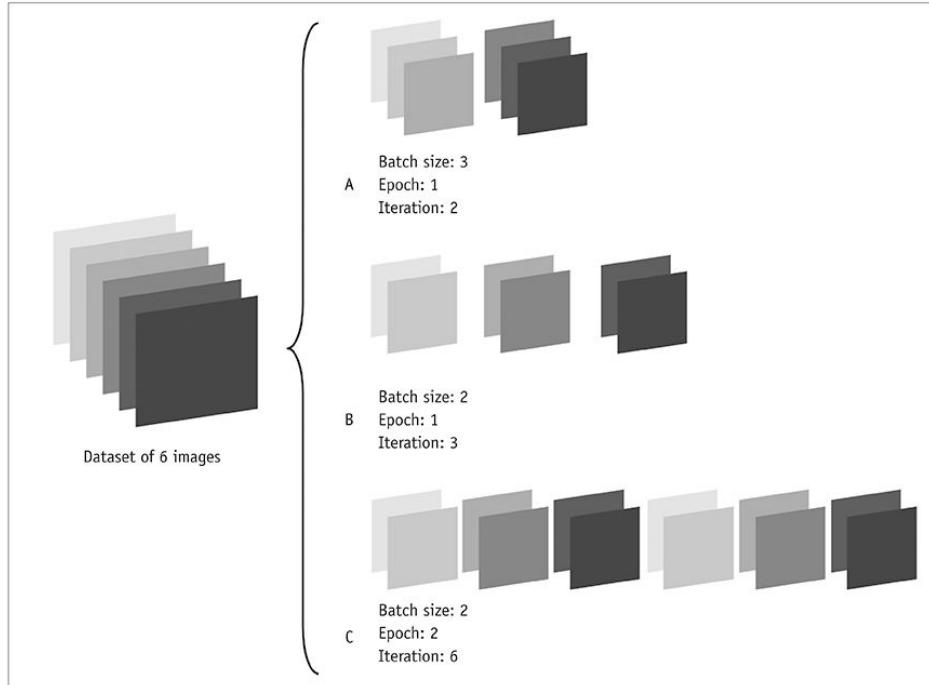
Parameters training

- Learning-rate
 - Learning rate scheduler
- Resolution
- Computational capacity



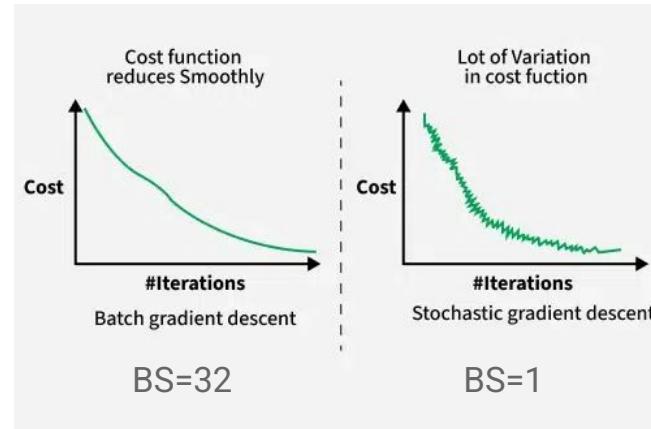
Parameters training

- Learning-rate
 - Learning rate scheduler
- Resolution
- Computational capacity
- Batch-size



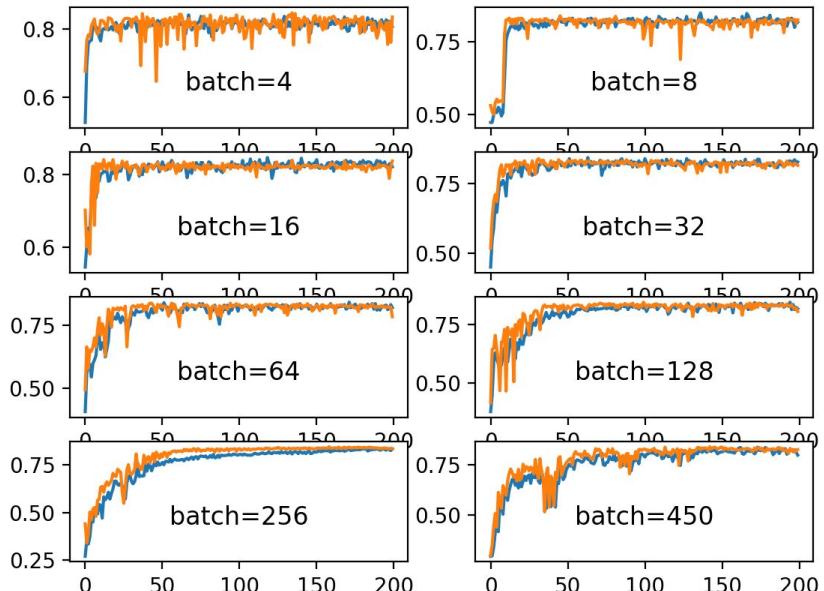
Parameters training

- Learning-rate
 - Learning rate scheduler
- Resolution
- Computational capacity
- Batch-size



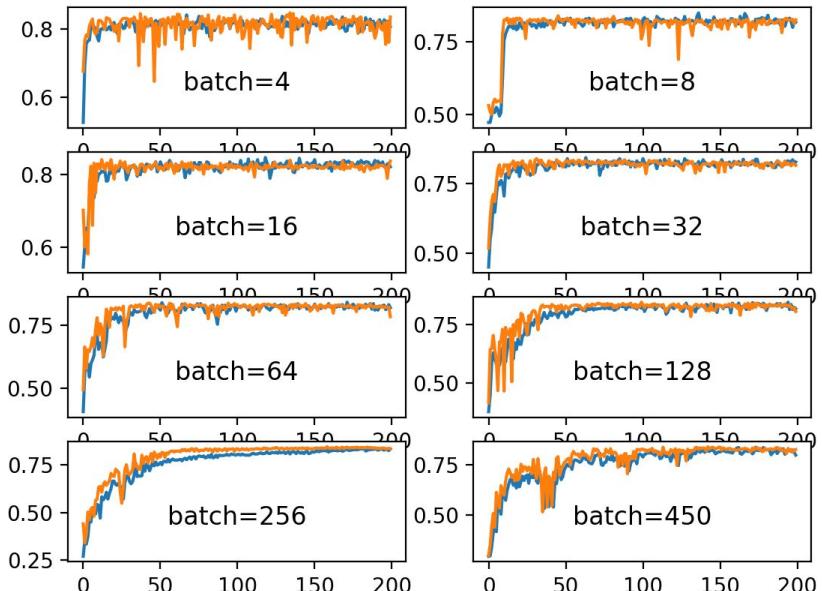
Parameters training

- Learning-rate
 - Learning rate scheduler
- Resolution
- Computational capacity
- Batch-size

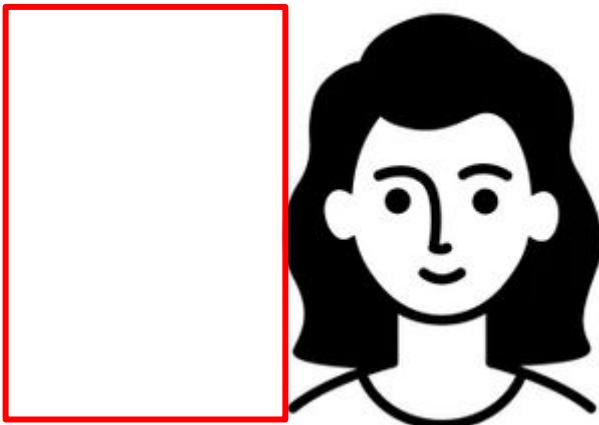


Parameters training

- Learning-rate
 - Learning rate scheduler
- Resolution
- Computational capacity
- Batch-size
- Optimizer
 - SGD
 - Adam
 - Adamr
 - etc
- Gradient scaler
- Weights training
- Auxiliar loss functions
- etc

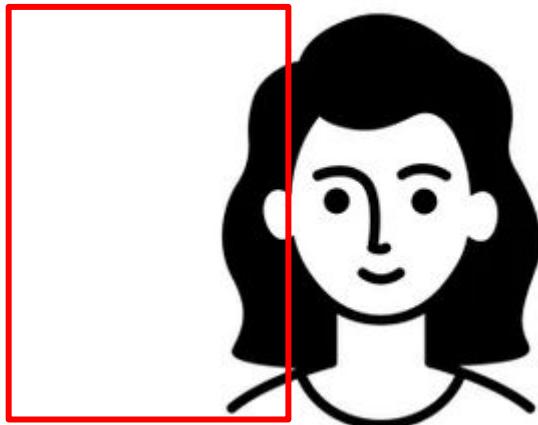


- Box loss
 - IoU loss



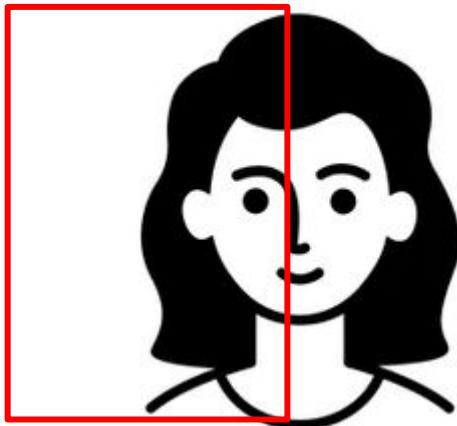
Metrics and losses

- Box loss
 - IoU loss



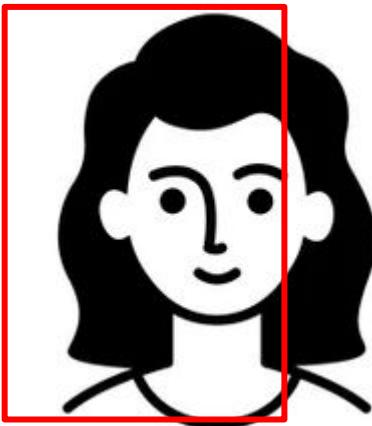
Metrics and losses

- Box loss
 - IoU loss

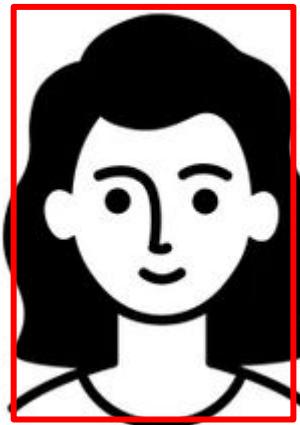


Metrics and losses

- Box loss
 - IoU loss



- Box loss
 - IoU loss



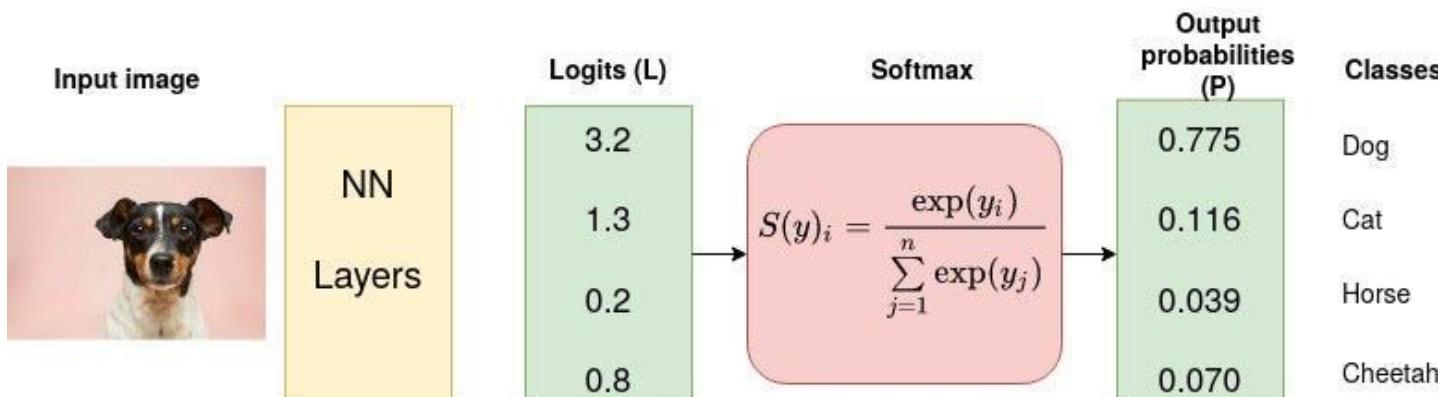
Metrics and losses

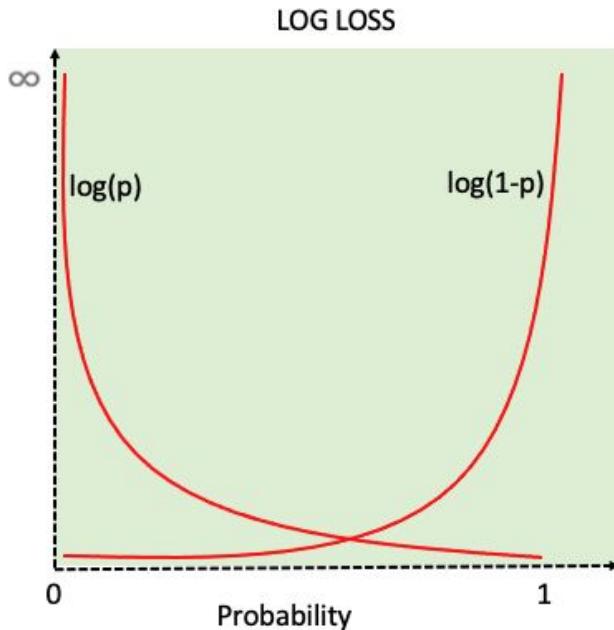
- Box loss
 - IoU loss

Intersection over Unit (IoU) =
$$\frac{\text{Area of overlap}}{\text{Area of union}}$$

$$= \frac{|A \cap B|}{|A \cup B|}$$

- Box loss
 - IoU loss
- Class loss
 - Cross entropy loss





- Box loss
 - IoU loss
- Class loss
 - Cross entropy loss

$$H(x) = -y\log_2(p_1) - (1-y)\log_2(1-p_1)$$