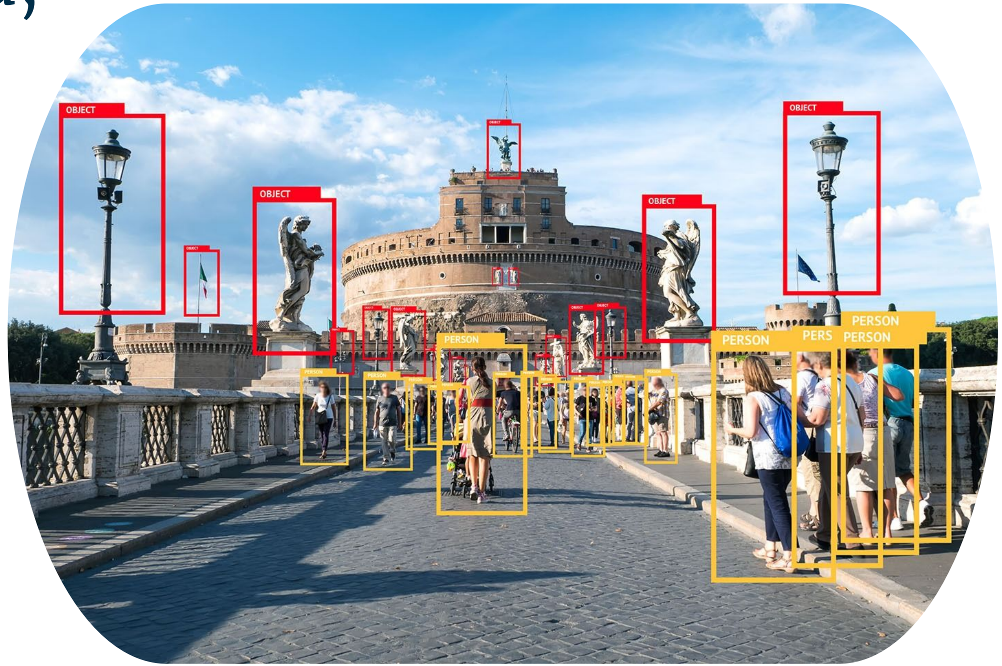
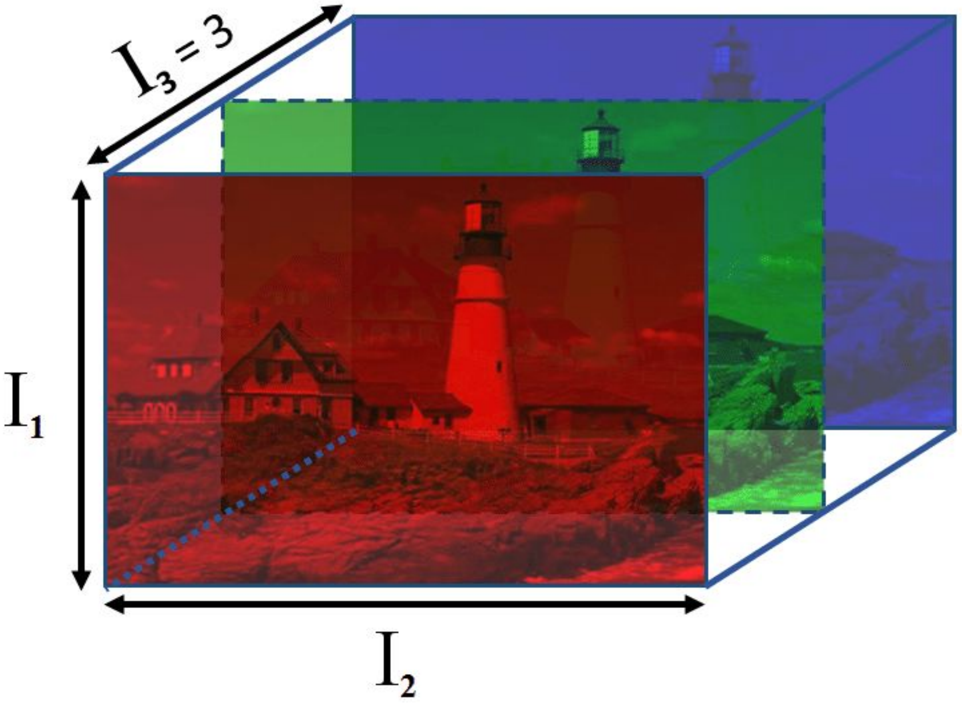
Detection: Understand, locate and classify



Luis Cossio
Master in Engineering sciences, mention in Electrical Engineering

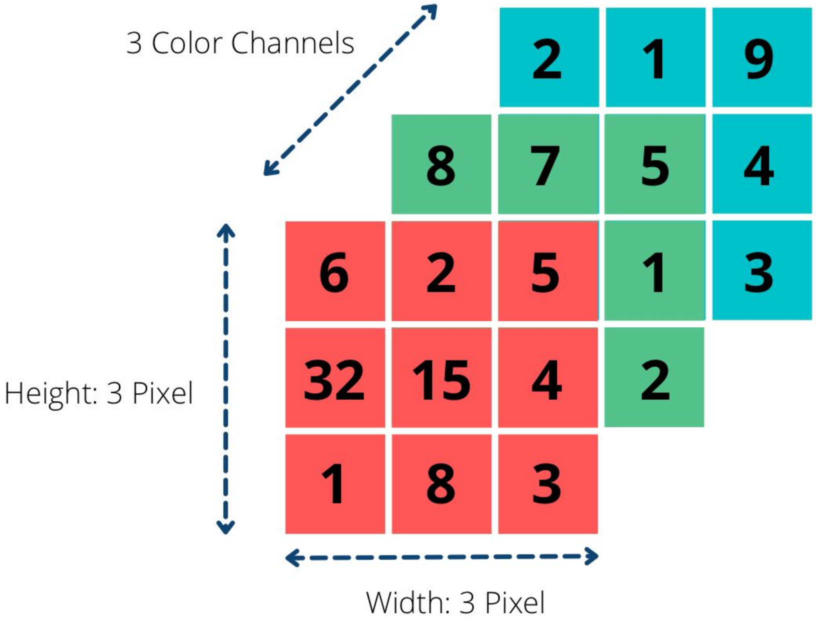
Images

- 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)



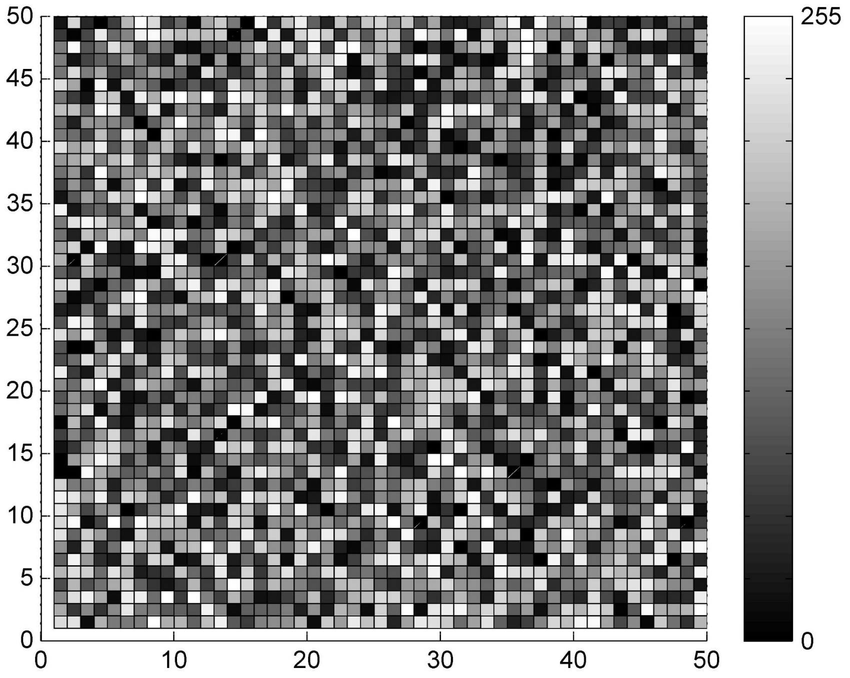
Images

- 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)

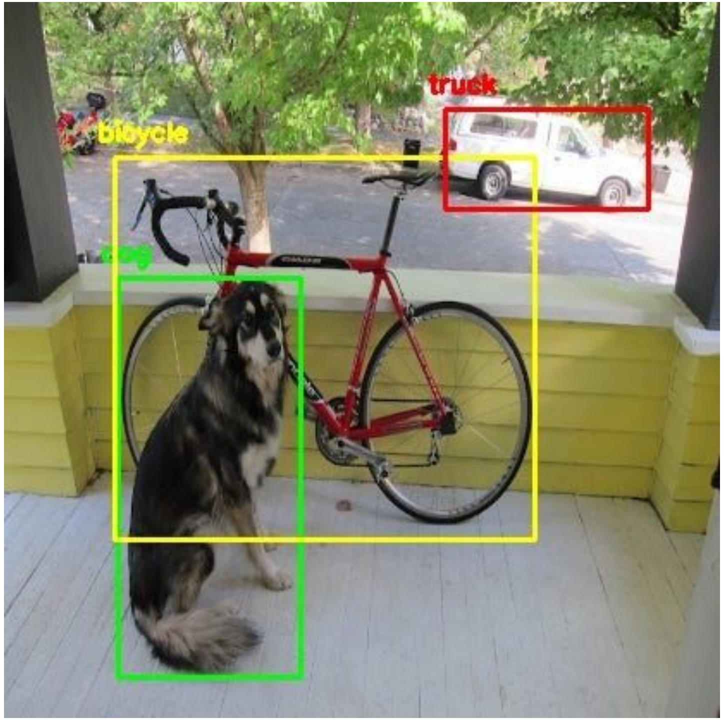


Images

- 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



Detection

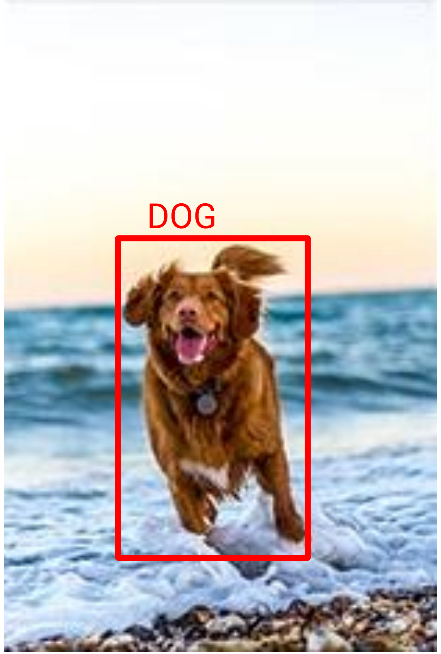


- Detection consist of 3 tasks
  - Separating target objects from background
  - Locating objects
    - Often using a Bounding box
  - Classifying objects in each class

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!!!

Detection

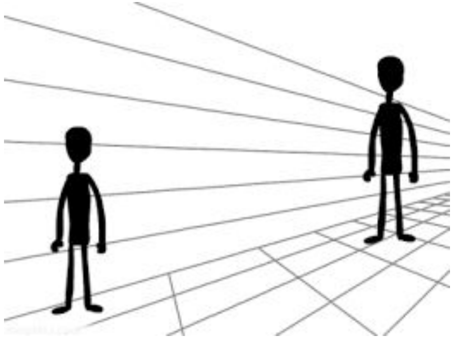


- Detection consist of 3 tasks
  - Separating target objects from background
  - Locating objects
    - Often using a Bounding box
  - Classifying objects in each class
- Very simple task
  - The objects are in plain sight!!!

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

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

- 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

- 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

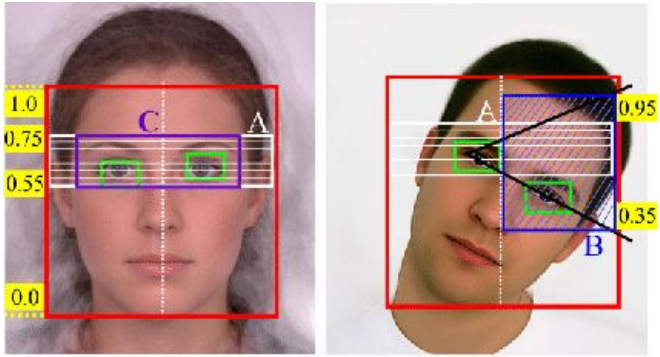
Detection

- 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

- 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

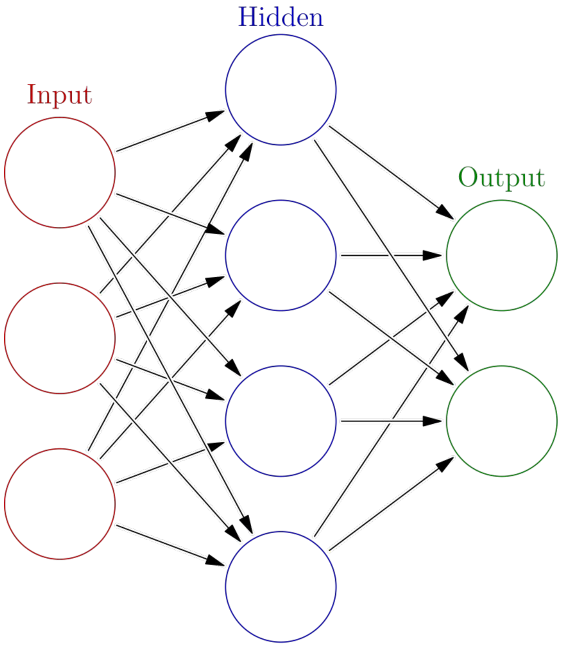
Detection



- 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

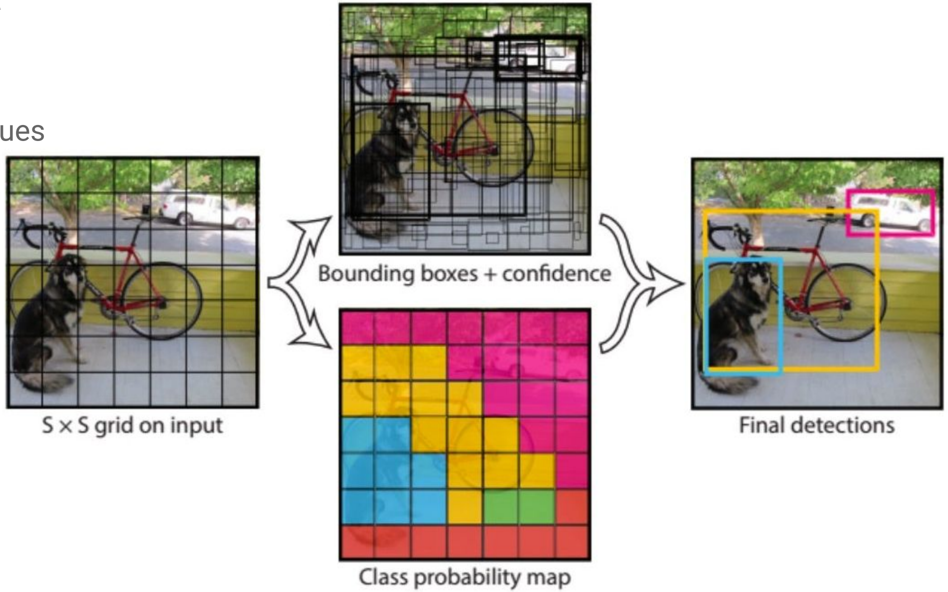
Neural Networks

- 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



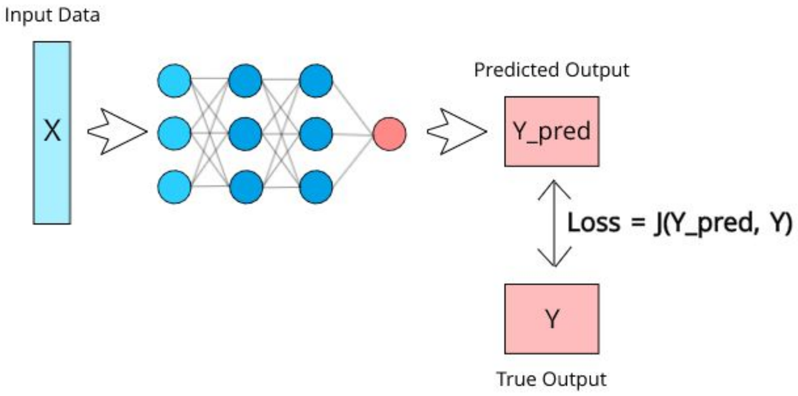
Object detection using Neural Networks

- 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



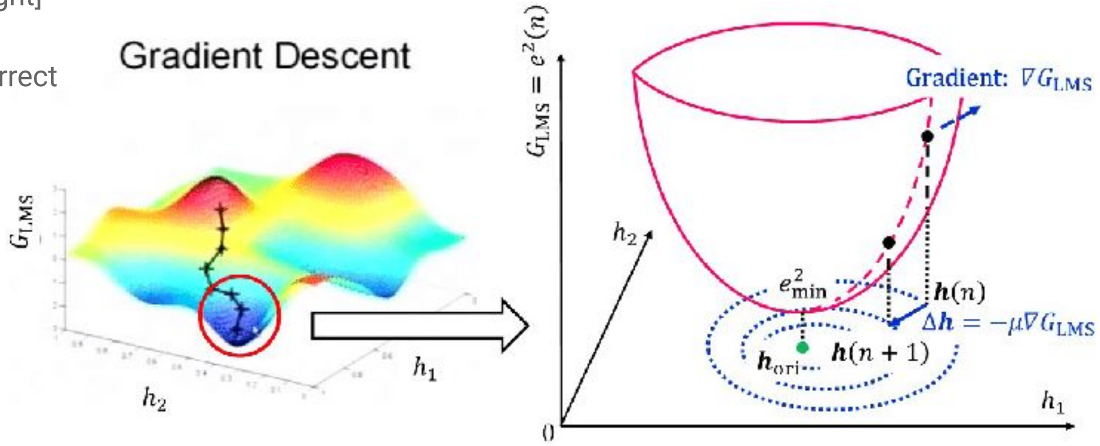
Training a neural Network

- 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



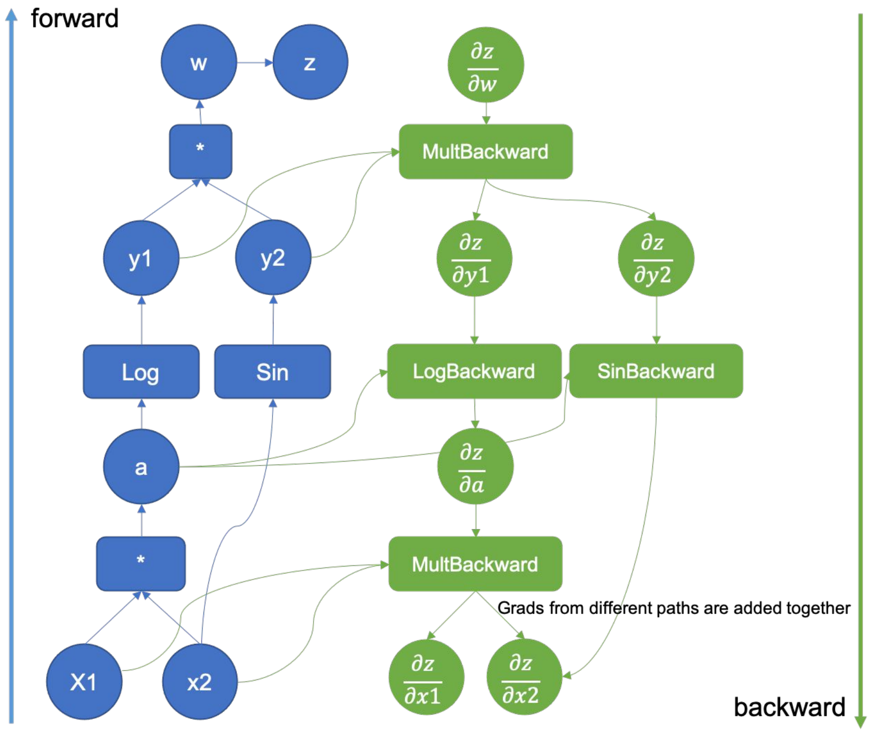
Training a neural Network

- 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

```python
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()
```

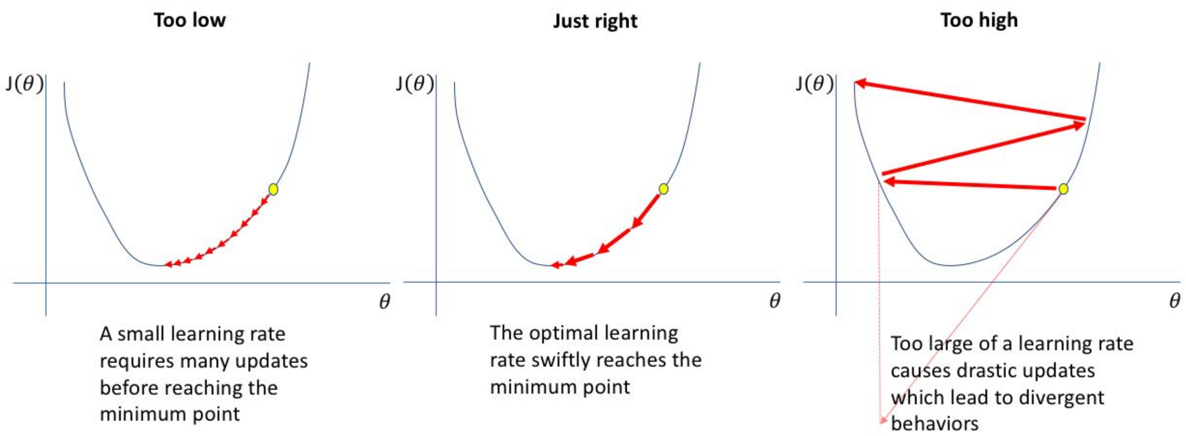
Training Cycle

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()
```

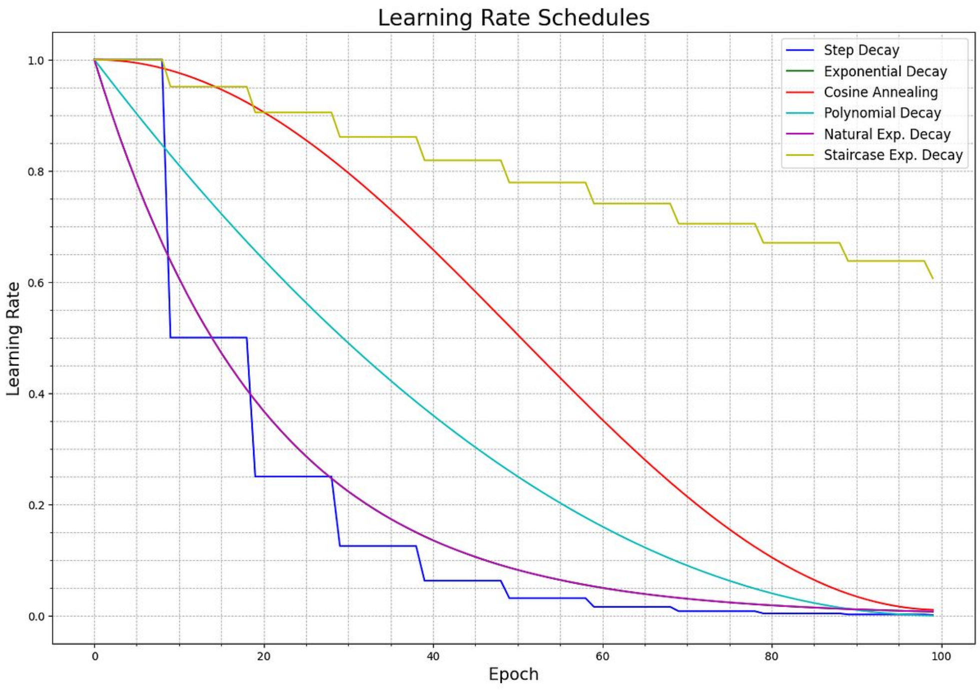
Parameters training

- Learning-rate
  - Learning rate scheduler



Parameters training

- Learning-rate
  - Learning rate scheduler



Parameters training

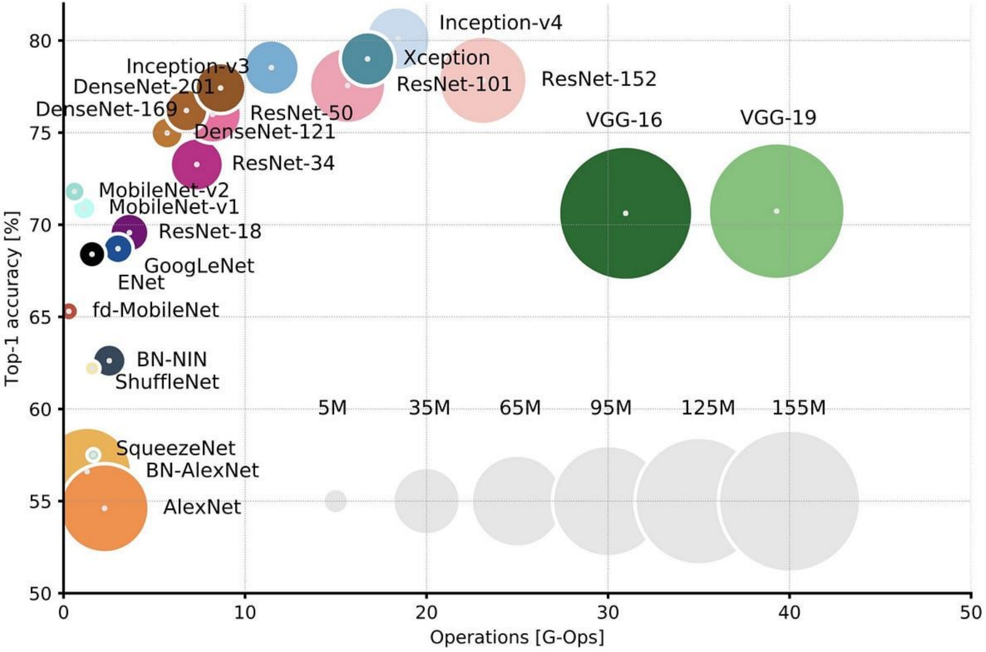
- Learning-rate
  - Learning rate scheduler
- Resolution

Parameters training

- Learning-rate
  - Learning rate scheduler
- Resolution

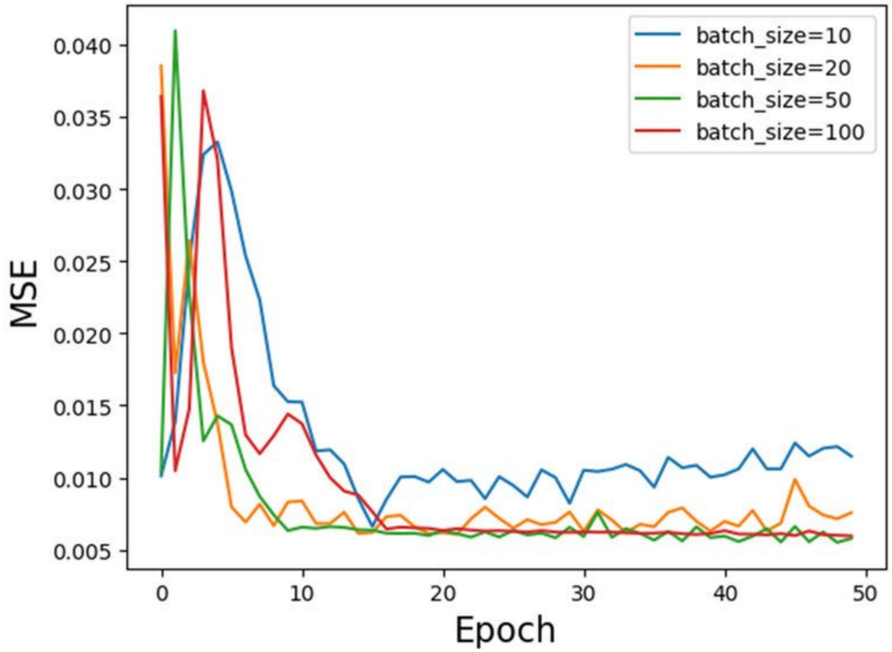
Parameters training

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



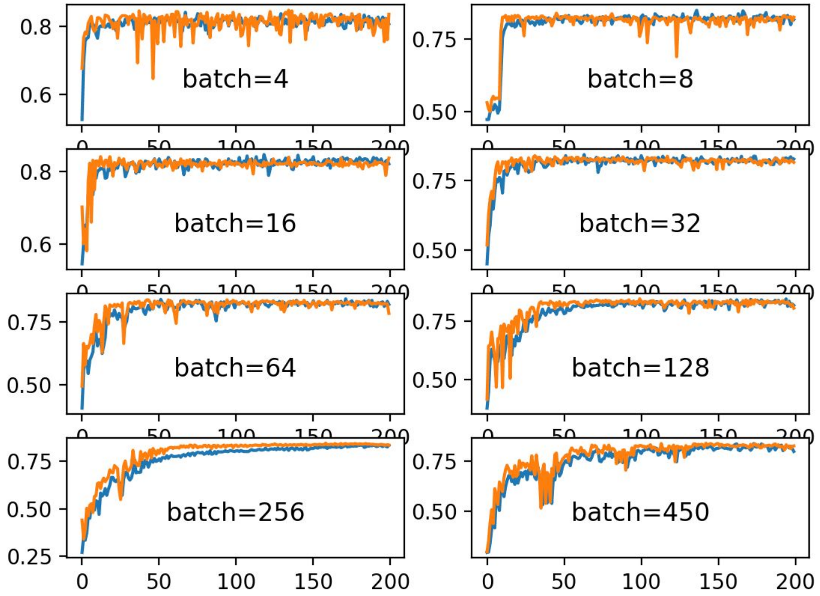
Parameters training

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



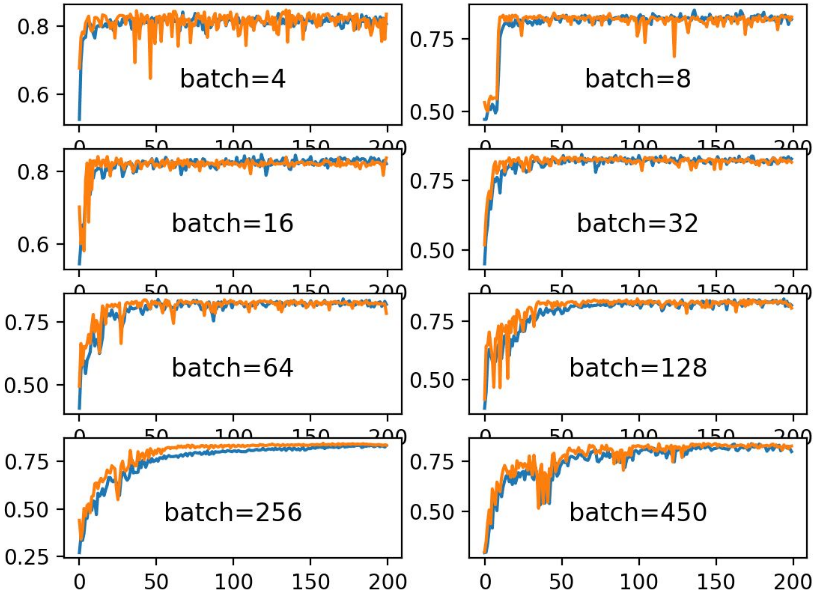
Parameters training

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



Parameters training

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



Detection

- Init Method
  - Setup training parameters

```python
self.is_coco = data_dict['nc'] == 80

# Logging- Doing this before checking the dataset. Might update data_dict
self.loggers = {'wandb': None}  # loggers dict

opt.hyp = hyp  # add hyperparameters
weights, epochs, hyp = opt.weights, opt.epochs, opt.hyp  # WandbLogger might update weights, epochs if resuming

# self.wandb_logger = wandb_logger
self.epochs = opt.epochs
self.batch_size = batch_size
self.nc = 1 if opt.single_cls else int(data_dict['nc'])  # number of classes
self.names = ['item'] if opt.single_cls and len(data_dict['names']) != 1 else data_dict['names']  # class names
self.data_dict = data_dict
assert len(self.names) == self.nc, '%g names found for nc=%g dataset in %s' % (
    len(self.names), self.nc, opt.data)  # check
```

Detection

- Init Method
  - Setup training parameters
  - Load model

```python
pretrained = weights.endswith('.pt')
if pretrained:
    ckpt = torch.load(weights, map_location=device)  # load checkpoint
    self.model = Model(opt.cfg or ckpt['model'].yaml, ch=3, nc=self.nc, anchors=hyp.get('anchors')).to(
        device)  # create
    exclude = ['anchor'] if (opt.cfg or hyp.get('anchors')) and not opt.resume else []  # exclude keys
    state_dict = ckpt['model'].float().state_dict()  # to FP32
    state_dict = intersect_dicts(state_dict, self.model.state_dict(), exclude=exclude)  # intersect
    self.model.load_state_dict(state_dict, strict=False)  # load
    logger.info(
        'Transferred %g/%g items from %s' % (len(state_dict), len(self.model.state_dict()), weights))
else:
    self.model = Model(opt.cfg, ch=3, nc=self.nc, anchors=hyp.get('anchors')).to(device)  # create
```

Detection

```
if opt.adam:
    self.optimizer = optim.Adam(pg0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999))  # adj
else:
    self.optimizer = optim.SGD(pg0, lr=hyp['lr0'], momentum=hyp['momentum'], nesterov=True

self.optimizer.add_param_group(
    {'params': pg1, 'weight_decay': hyp['weight_decay']})  # add pg1 with weight_decay
self.optimizer.add_param_group({'params': pg2})  # add pg2 (biases)
logger.info('Optimizer groups: %g .bias, %g conv.weight, %g other' % (len(pg2), len(pg1),
del pg0, pg1, pg2
self.hyp = hyp
lf = one_cycle(1, self.hyp['lrf'], self.epochs)  # cosine 1->hyp['lrf']
```

- Init Method
  - Setup training parameters
  - Load model
  - Define optimization objects

Detection

- Setup training Method
    - Configure loss related hyper-parameters
    -

```
# Model parameters
self.hyp['box'] *= 3. / self.number_layers  # scale to layers
self.hyp['cls'] *= self.nc / 80. * 3. / self.number_layers  # scale to classes and layers
self.hyp['obj'] *= (self.imgsz / 640) ** 2 * 3. / self.number_layers  # scale to image size and la
self.hyp['label_smoothing'] = opt.label_smoothing
self.model.nc = self.nc  # attach number of classes to model
self.model.hyp = self.hyp  # attach hyperparameters to model
self.model.gr = 1.0  # iou loss ratio (obj_loss = 1.0 or iou)
self.model.class_weights = labels_to_class_weights(self.dataset_train.labels, self.nc).to(
    self.device) * self.nc  # attach class weights
self.model.names = self.names
```

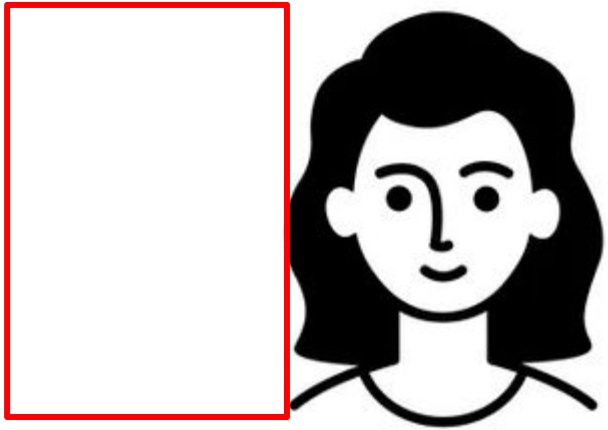
Detection

- Setup training Method
  - Configure loss related hyper-parameters
  - Define loss function

```
self.scheduler.last_epoch = self.start_epoch - 1  # do not move

logger.info(f'Image sizes {self.imgsz} train, {self.imgsz_test} test\n'
            f'Using {self.loader_train.num_workers} dataloader workers\n'
            f'Logging results to {self.save_dir}\n'
            f'Starting training for {self.epochs} epochs...')

self.cuda = self.device.type != 'cpu'
self.scaler = amp.GradScaler(enabled=self.cuda)
self.compute_loss_ota = ComputeLossAuxOTA(self.model)  # init loss class
self.compute_loss = ComputeLoss(self.model)  # init loss class
```

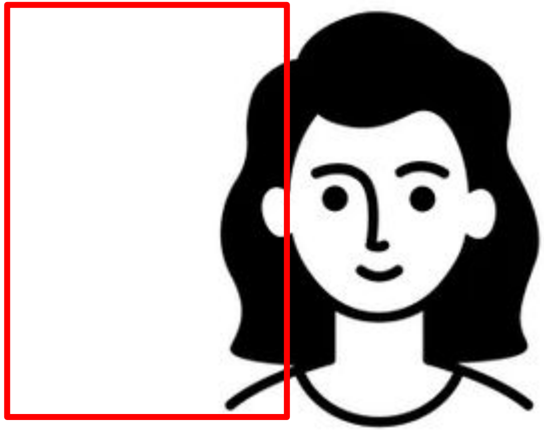
Metrics and losses

- Box loss
  - IoU loss



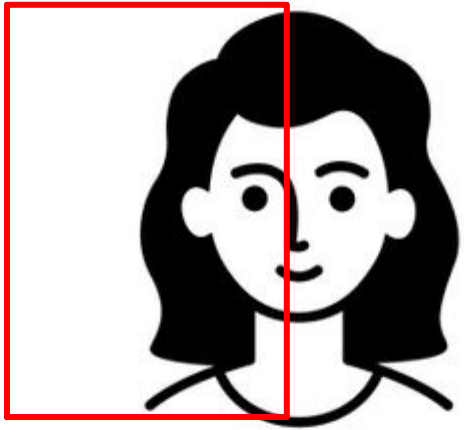
Metrics and losses

- Box loss
  - IoU loss



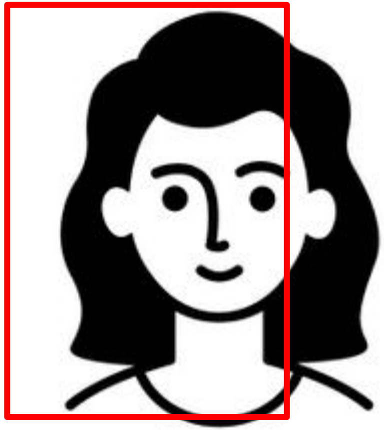
Metrics and losses

- Box loss
  - IoU loss

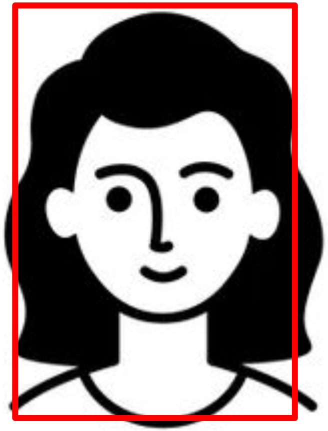


Metrics and losses

- Box loss
    - IoU loss



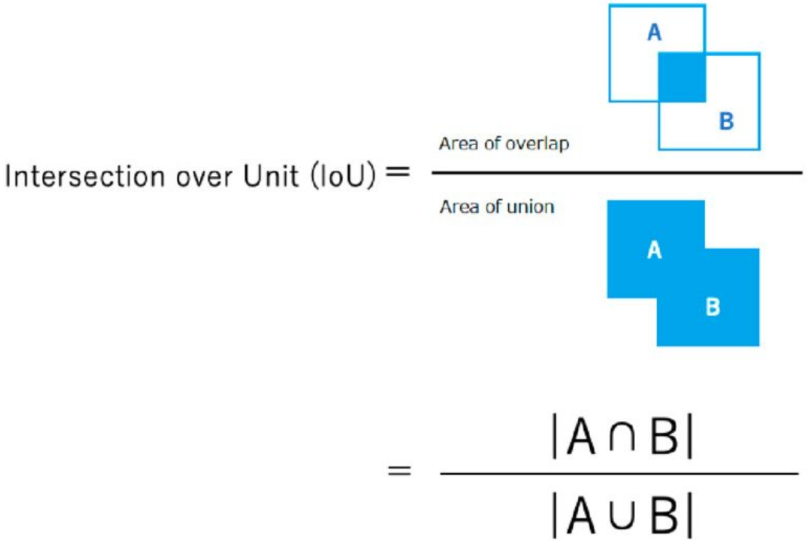
Metrics and losses



- Box loss
  - IoU loss

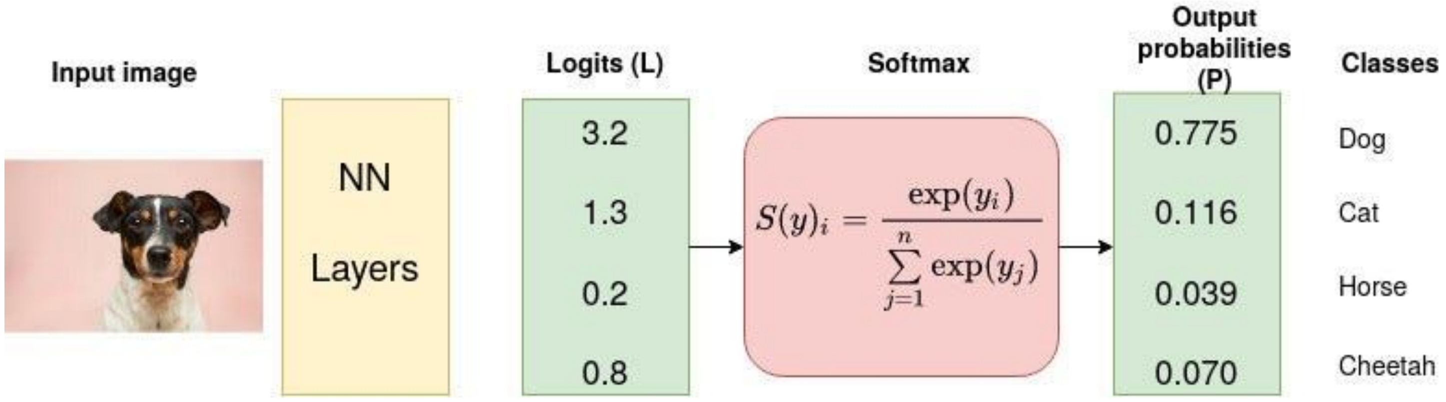
Metrics and losses

- Box loss
  - IoU loss

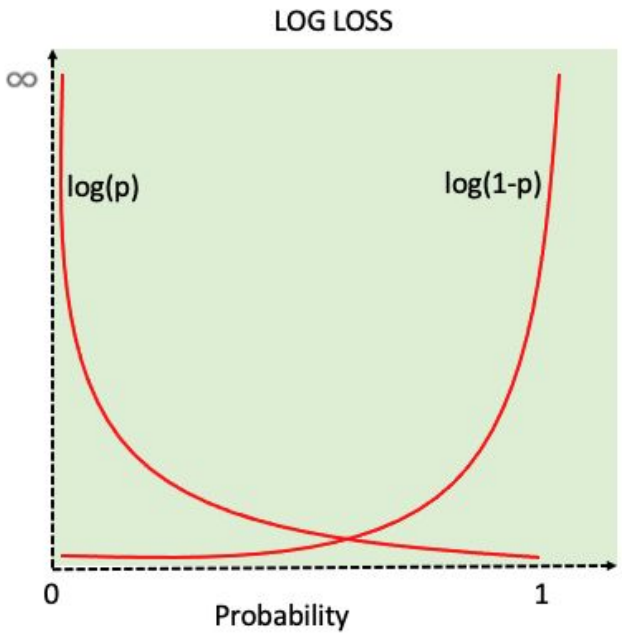


Metrics and losses

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

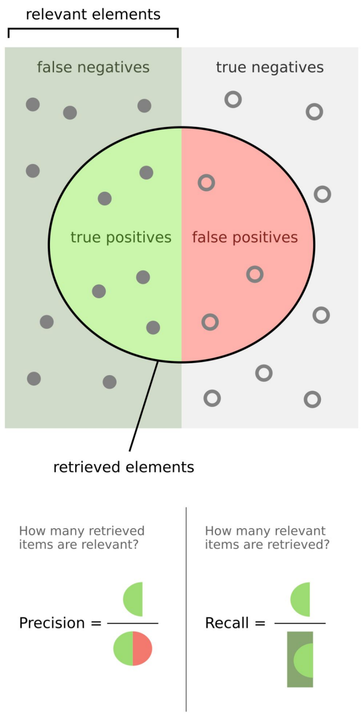


Metrics and losses



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

Metrics and losses



- Box loss
    - IoU loss
- Precision and Recall

Detection

```
def train(self, opt, device):
    results = (0, 0, 0, 0, 0, 0, 0)  # P, R, mAP@.5, mAP@.5-.95, val_loss(box, obj, cls)
    lf = one_cycle(1, self.hyp['lrf'], self.epochs)  # cosine 1->hyp['lrf']

    for epoch in range(self.start_epoch, self.epochs):  # epoch ------------------------------------------------
        self.model.train()
        ############## SETUP PROGRESS BAR VARIABLES ####################
        mloss = torch.zeros(4, device=device)  # mean losses
        pbar = enumerate(self.loader_train)
        logger.info(('\n' + '%10s' * 8) % ('Epoch', 'gpu_mem', 'box', 'obj', 'cls', 'total', 'labels', 'i
        pbar = tqdm(pbar, total=self.nb)  # progress bar
        self.optimizer.zero_grad()
        for i, (imgs, targets, paths, _) in pbar:  # batch ------------------------------------------------
            ni = i + self.nb * epoch  # number integrated batches (since train start)
            imgs = imgs.to(device, non_blocking=True).float() / 255.0  # uint8 to float32, 0-255 to 0.0-1.

            # Warmup
            if ni <= self.nw:
                self.warmup_learning_rate(ni, epoch, lf)

            # Multi-scale
            if opt.multi_scale:
                sz = random.randrange(self.imgsz * 0.5, self.imgsz * 1.5 + self.gs) // self.gs * self.gs
                sf = sz / max(imgs.shape[2:])  # scale factor
                if sf != 1:
                    ns = [math.ceil(x * sf / self.gs) * self.gs for x in
                          imgs.shape[2:]]  # new shape (stretched to gs-multiple)
                    imgs = F.interpolate(imgs, size=ns, mode='bilinear', align_corners=False)

            # Forward
            with amp.autocast(enabled=self.cuda):
                pred = self.model(imgs)  # forward
                loss, loss_items = self.compute_loss_ota(pred, targets.to(device),
                                                         imgs)  # loss scaled by batch_size

            # Backward/load gradients
            self.scaler.scale(loss).backward()
```

- Train function

Detection

- Train function

```python
for epoch in range(self.start_epoch, self.epochs):  # epoch ------------------------------------
    self.model.train()
    ############## SETUP PROGRESS BAR VARIABLES ####################
    mloss = torch.zeros(4, device=device)  # mean losses
    pbar = enumerate(self.loader_train)
    self.optimizer.zero_grad()
    for i, (imgs, targets, paths, _) in pbar:  # batch ------------------------------------------
        ni = i + self.nb * epoch  # number integrated batches (since train start)
        imgs = imgs.to(device, non_blocking=True).float() / 255.0  # uint8 to float32, 0-255 to 0.0-1.
        with amp.autocast(enabled=self.cuda):
            pred = self.model(imgs)  # forward
            loss, loss_items = self.compute_loss_ota(pred, targets.to(device),
                                                     imgs)  # loss scaled by batch_size

        # Backward/load gradients
        self.scaler.scale(loss).backward()
```

Detection

- Train function

```
for epoch in range(self.start_epoch, self.epochs):  # epoch ------------------------------------
    self.model.train()
    ############## SETUP PROGRESS BAR VARIABLES ####################
    mloss = torch.zeros(4, device=device)  # mean losses
    pbar = enumerate(self.loader_train)
    self.optimizer.zero_grad()
    for i, (imgs, targets, paths, _) in pbar:  # batch ------------------------------------
        ni = i + self.nb * epoch  # number integrated batches (since train start)
        imgs = imgs.to(device, non_blocking=True).float() / 255.0  # uint8 to float32, 0-255 to 0.0-1.
        with amp.autocast(enabled=self.cuda):
            pred = self.model(imgs)  # forward
            loss, loss_items = self.compute_loss_ota(pred, targets.to(device),
                                                     imgs)  # loss scaled by batch_size

        # Backward/load gradients
        self.scaler.scale(loss).backward()
```

Detection

- Train function

```python
for epoch in range(self.start_epoch, self.epochs):  # epoch ----------------------------------------
    self.model.train()
    ############## SETUP PROGRESS BAR VARIABLES ####################
    mloss = torch.zeros(4, device=device)  # mean losses
    pbar = enumerate(self.loader_train)
    self.optimizer.zero_grad()
    for i, (imgs, targets, paths, _) in pbar:  # batch ---------------------------------------
        ni = i + self.nb * epoch  # number integrated batches (since train start)
        imgs = imgs.to(device, non_blocking=True).float() / 255.0  # uint8 to float32, 0-255 to 0.0-1.
        with amp.autocast(enabled=self.cuda):
            pred = self.model(imgs)  # forward
            loss, loss_items = self.compute_loss_ota(pred, targets.to(device),
                                                     imgs)  # loss scaled by batch_size

        # Backward/load gradients
        self.scaler.scale(loss).backward()
```

Results

Experiments	Batch-size	Resolution	Epochs	mAP0.5@0.95
1	4	320 x 512	30	0.0003141
2	8	320 x 512	30	0.002141
3	24	320 x 512	90	0.63141