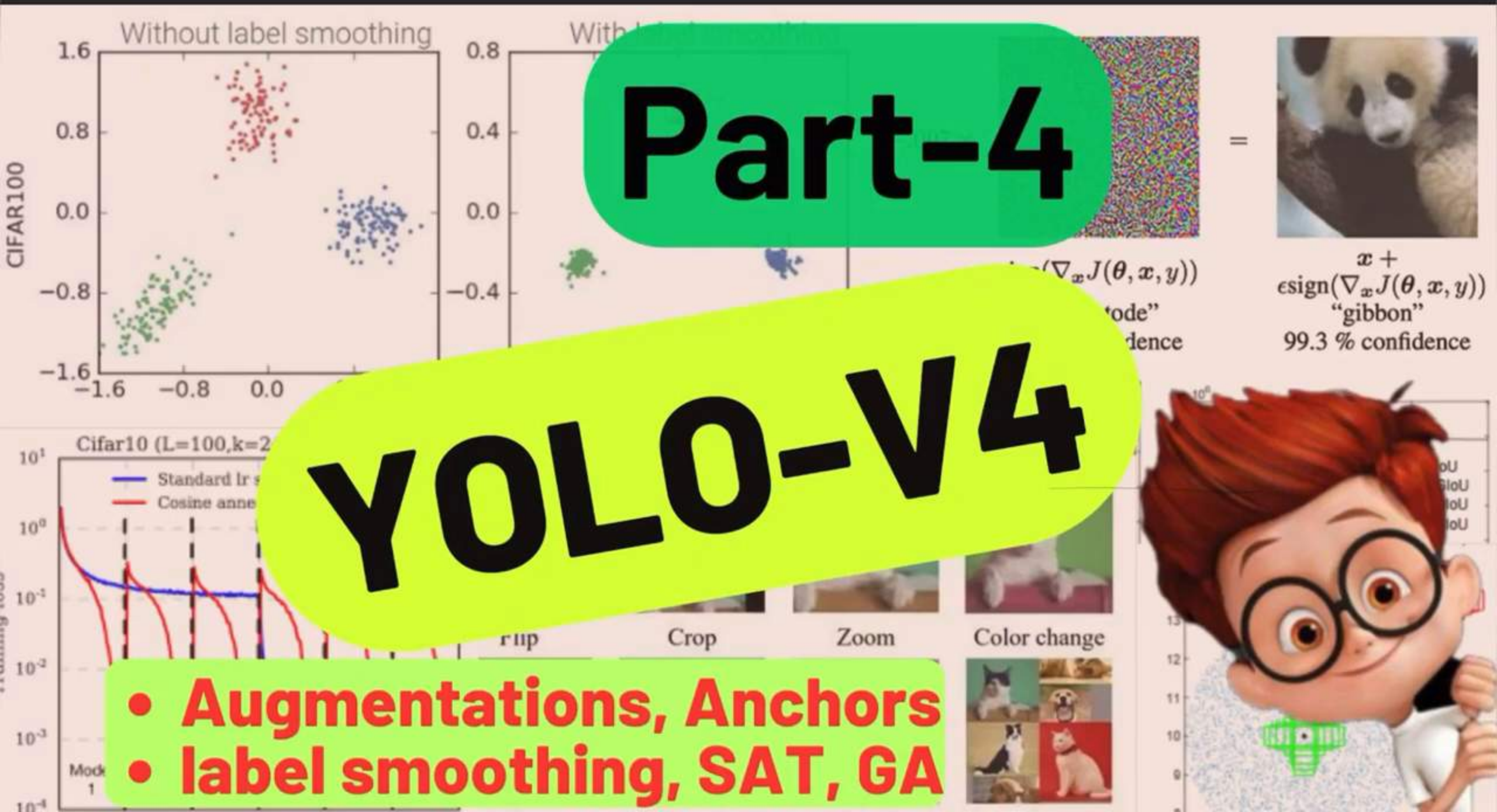
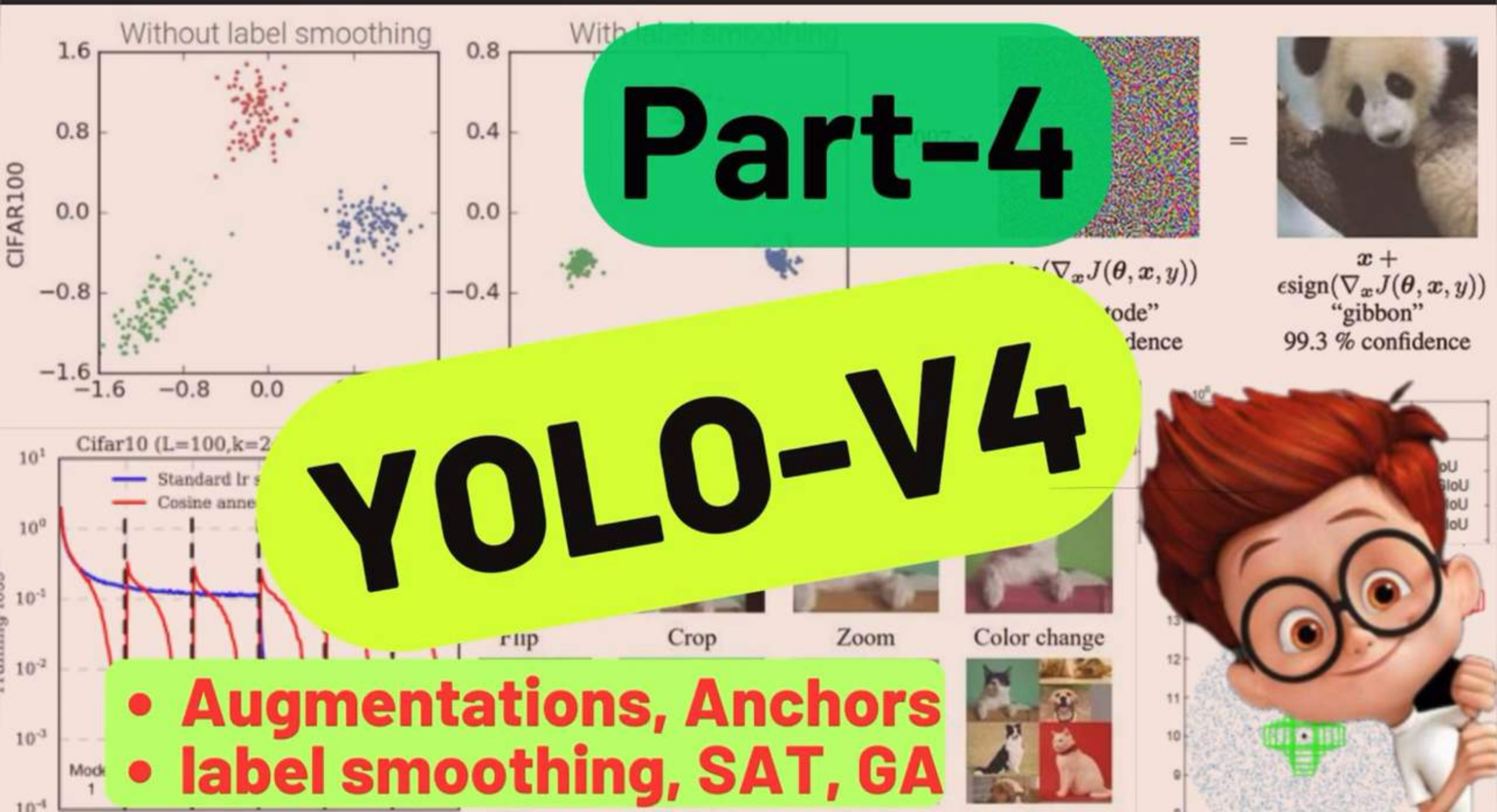

Without label smoothing
CIFAR100
Part-4
YOLO-V4
Cifar10 (L=100,k=2
Standard lr
Cosine anne
Flip
Crop
Zoom
Color change
• Augmentations, Anchors
• label smoothing, SAT, GA


Without label smoothing
CIFAR100
Part-4
YOLO-V4
Cifar10 (L=100,k=2
Standard lr
Cosine anne
Flip
Crop
Zoom
Color change
• Augmentations, Anchors
• label smoothing, SAT, GA

BoF & BoS - covered

	Backbone	Detector
Bag of Freebies (BoF)	• CutMix • Mosaic data augmentation • DropBlock • Class label smoothing	• CIoU-loss • Cross mini-Batch Normalization • DropBlock • Mosaic data augmentation • Self-Adversarial Training • Multiple anchors for a single ground truth • Cosine annealing scheduler • Optimal hyperparameters • Random training shapes
Bag of Specials (BoS)	• Mish activation • Cross-stage partial connections (CSP) • Multi-input weighted residual connections (MiWRC)	• Mish activation • SPP-block • SAM-block • PAN path-aggregation block • DIoU-NMS

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

- Methods to increase data
  - Volume
  - Variations

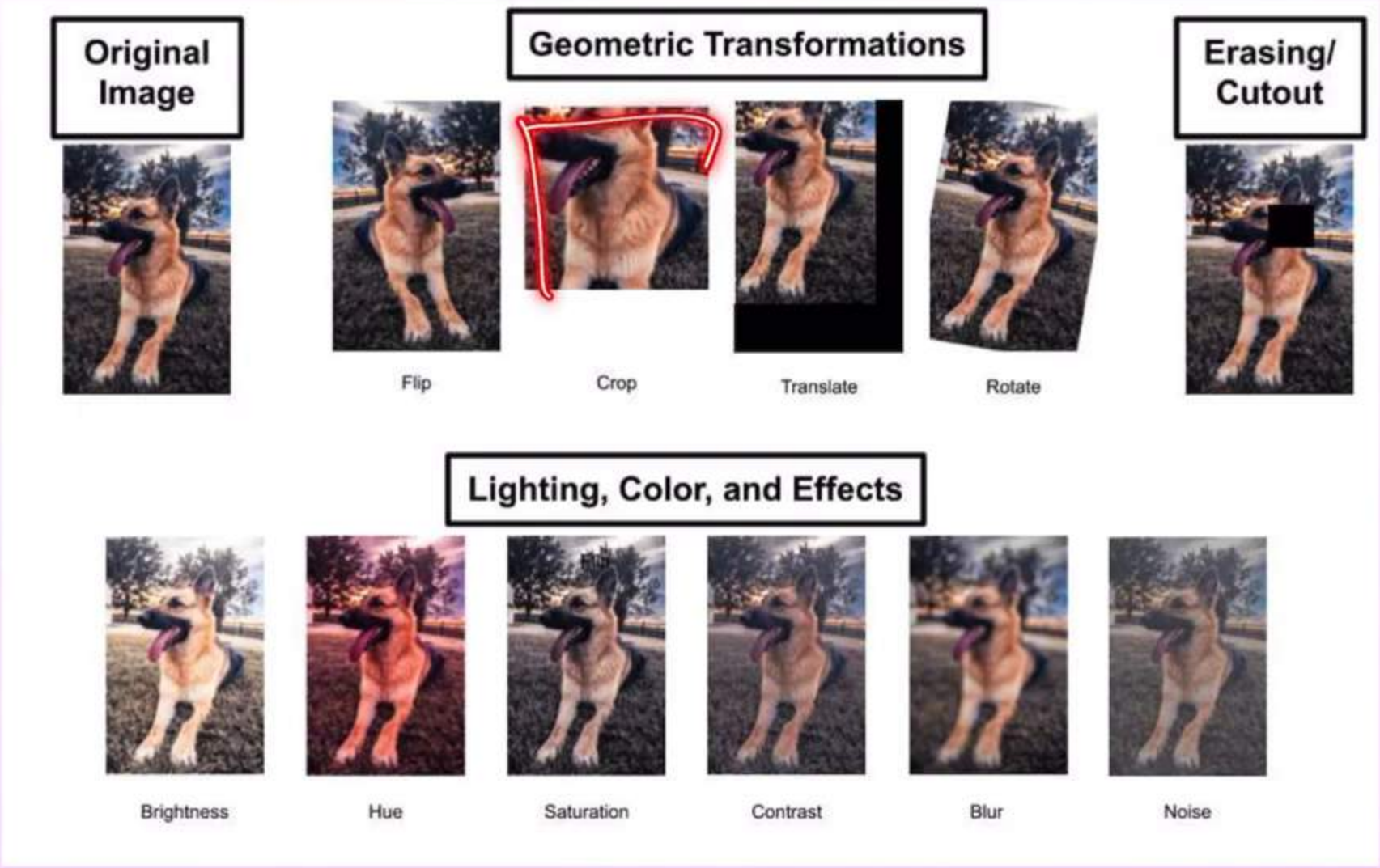
- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

Data Augmentations

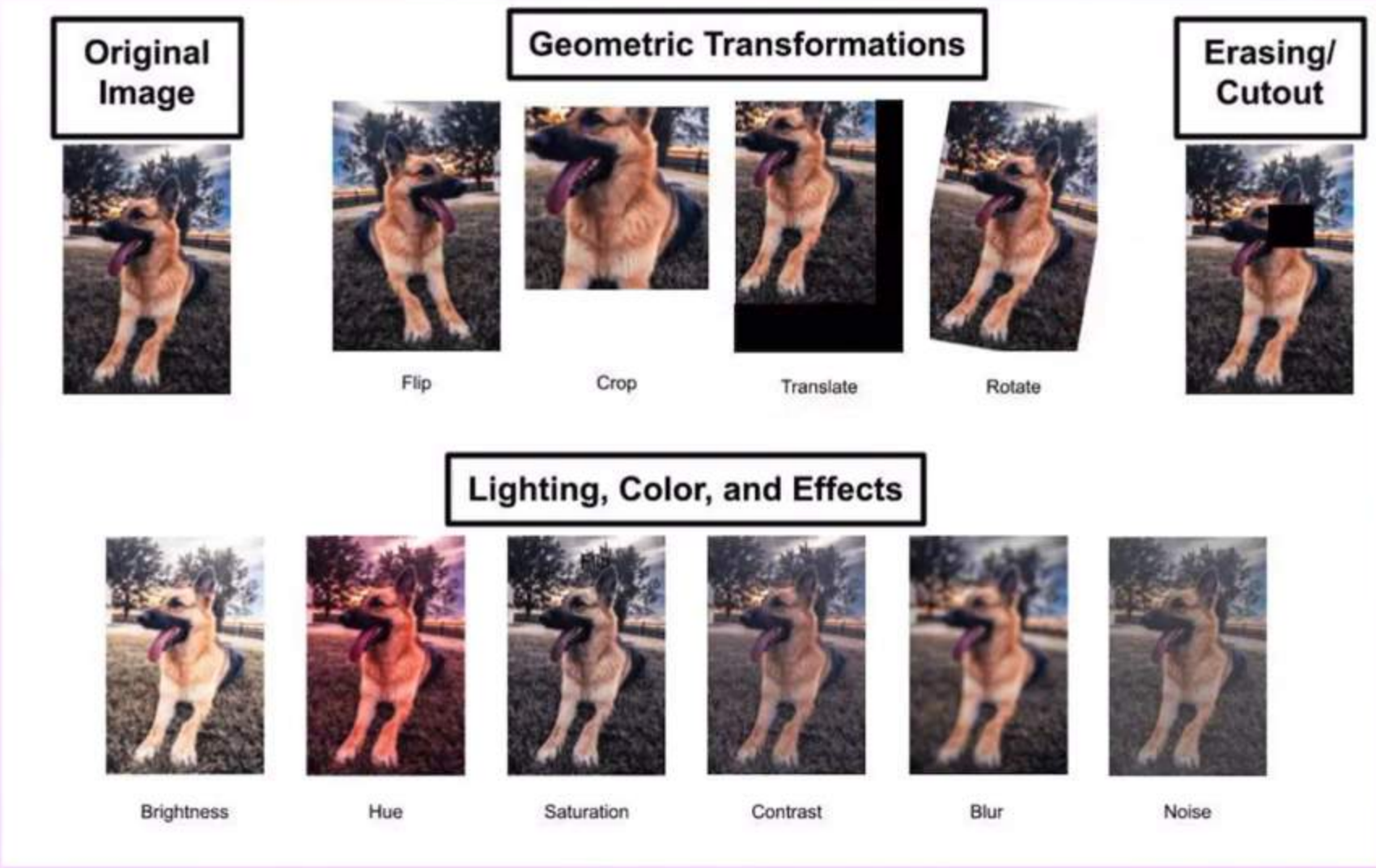
- Methods to increase data
  - Volume
  - Variations

- Methods:
  - Photometrics Distortions:
    - Noise, Brightness, Contrast, Hue etc.,
  - Geometric Distortions:
    - Flip
    - Rotate
    - Scaling
    - Cropping
  - Object Occlusions:
    - Random erase, cutout, grid mask etc.,
    - Clubbing multiple images

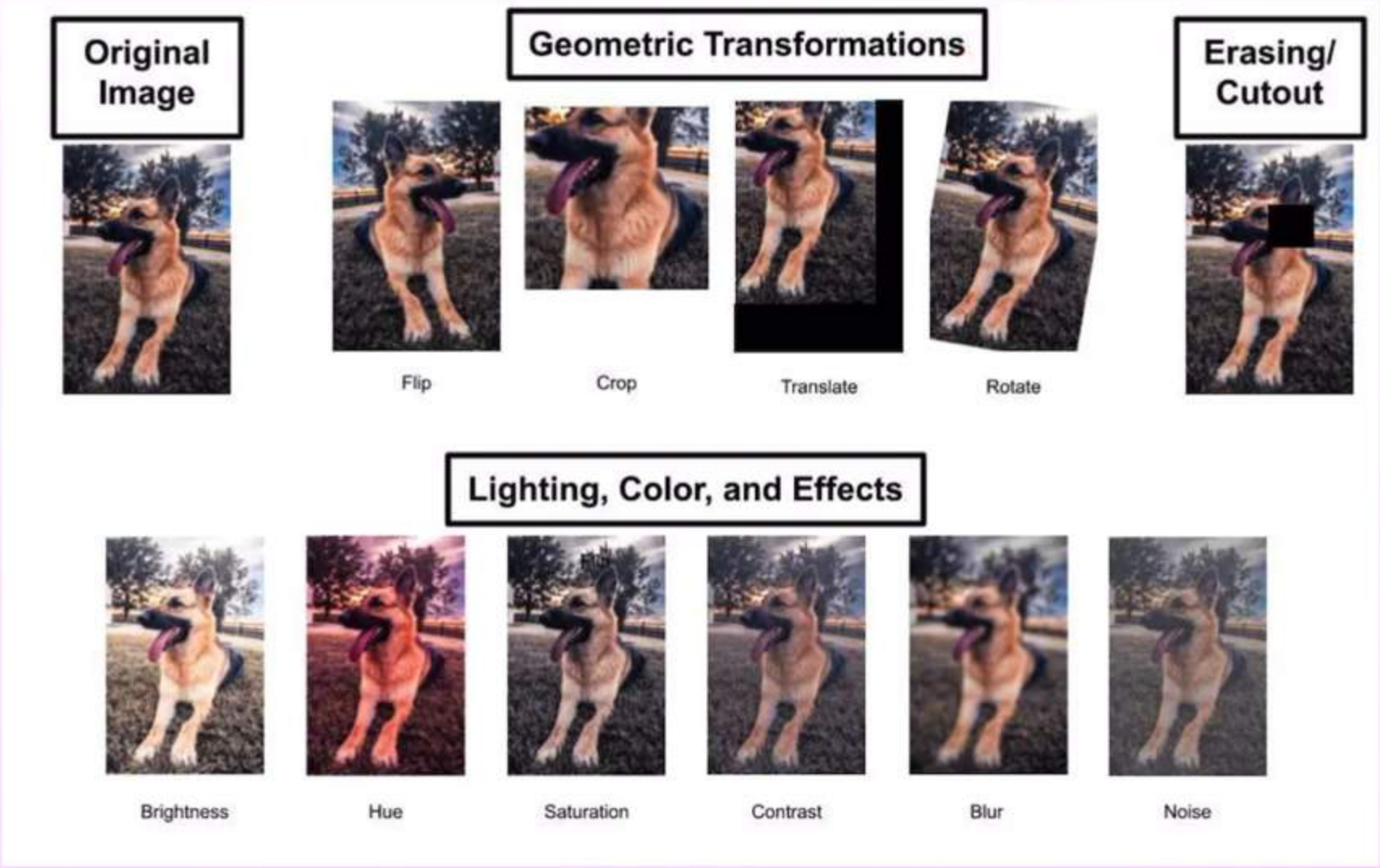
Data Augmentations



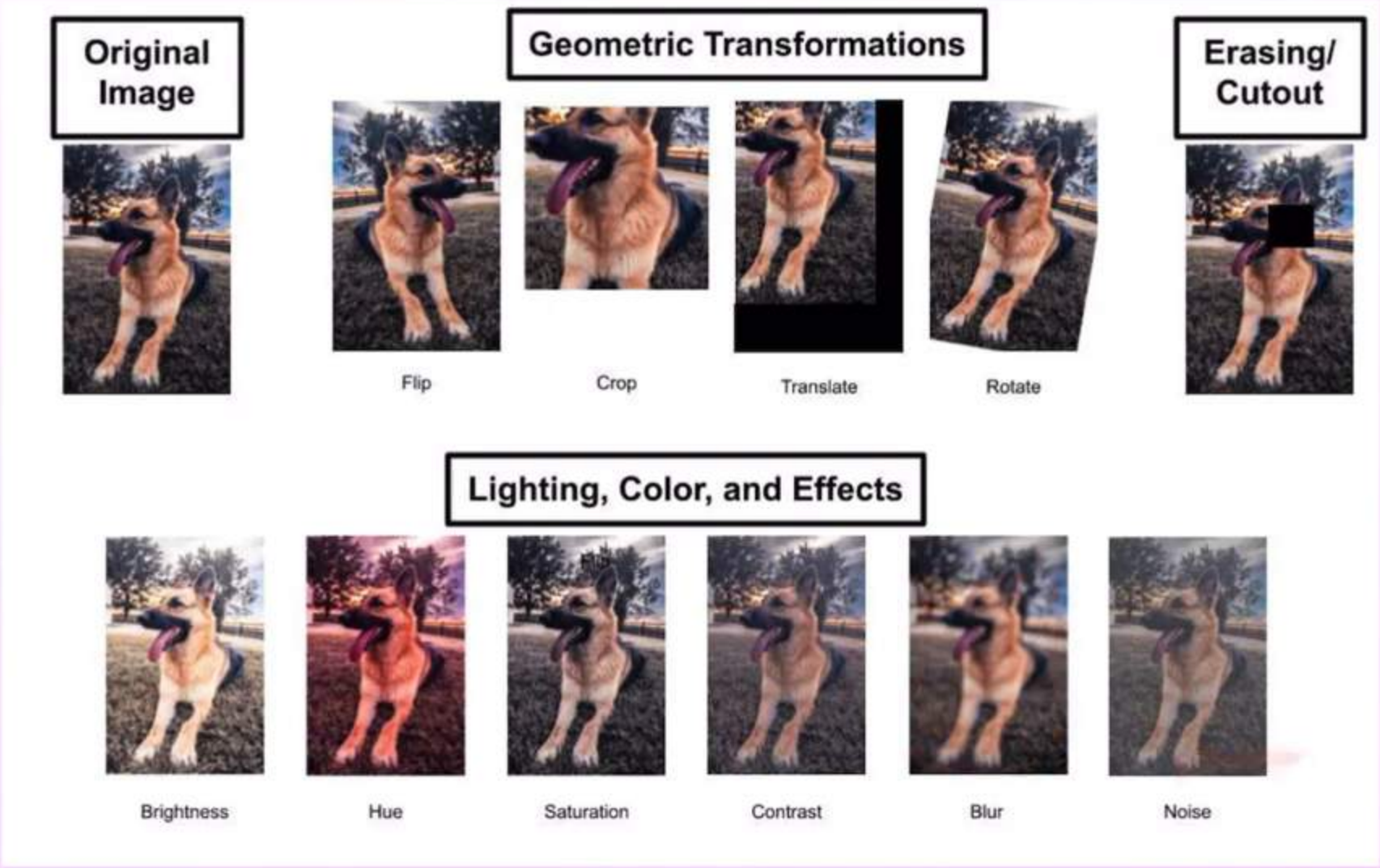
Data Augmentations



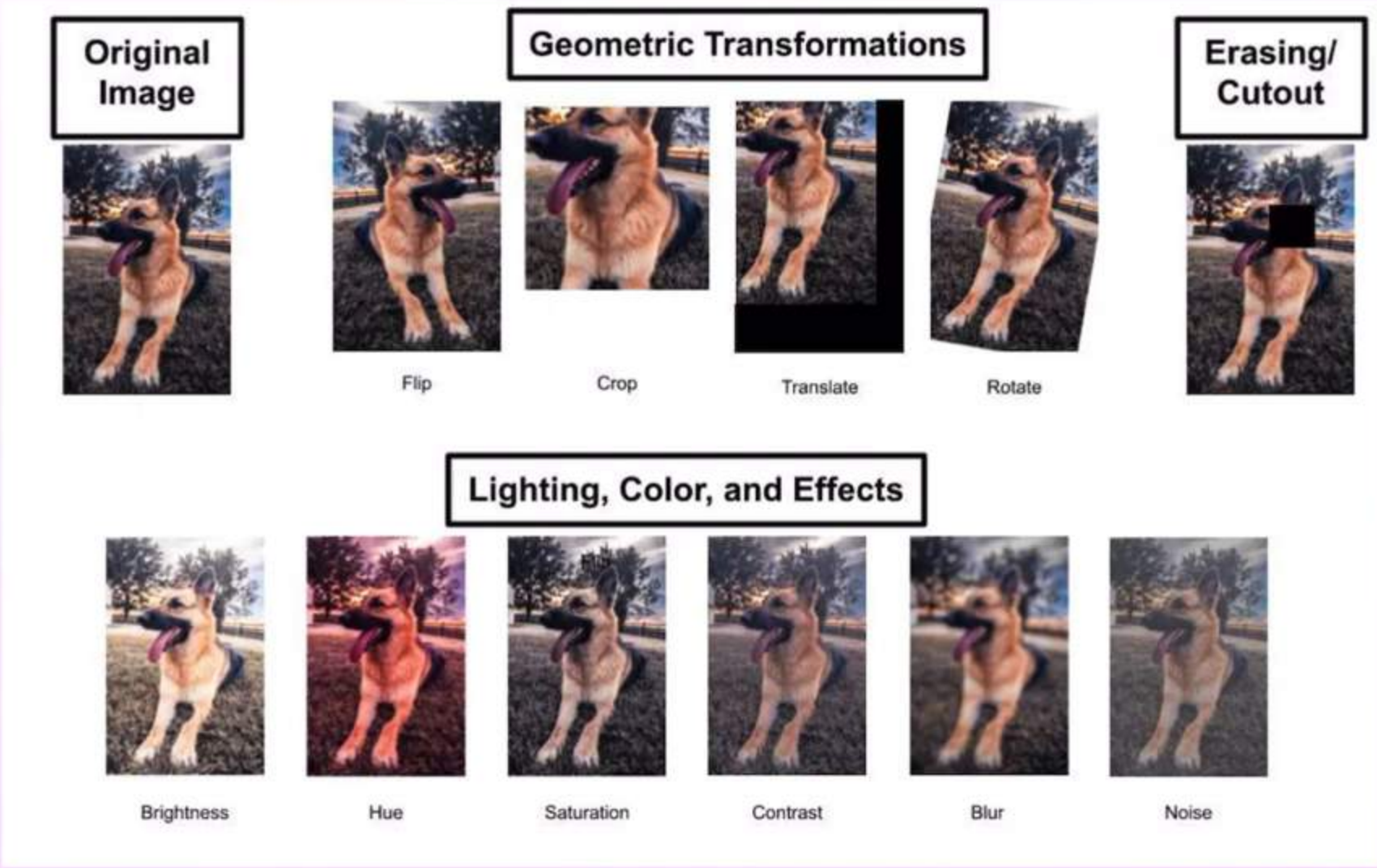
Data Augmentations



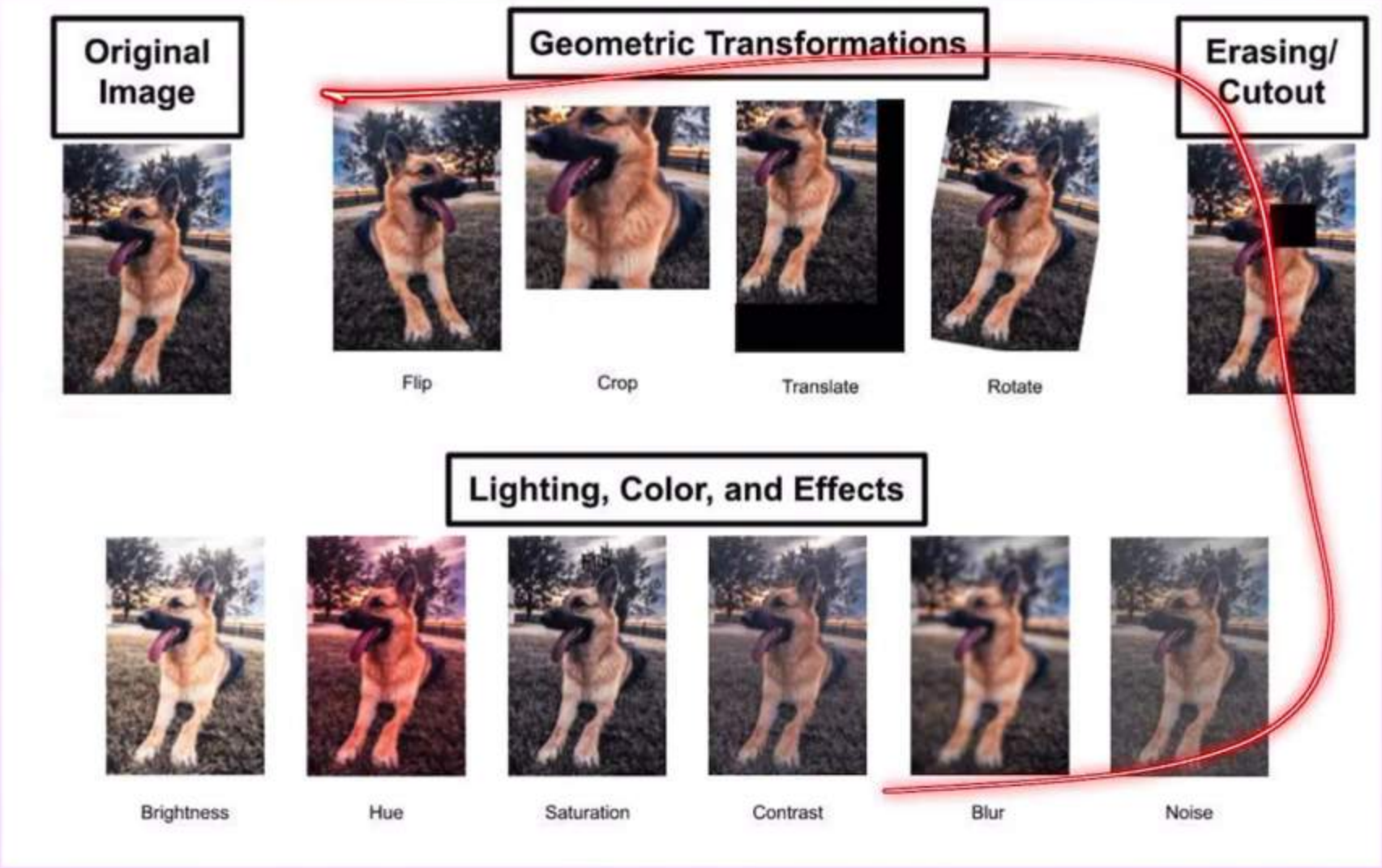
Data Augmentations



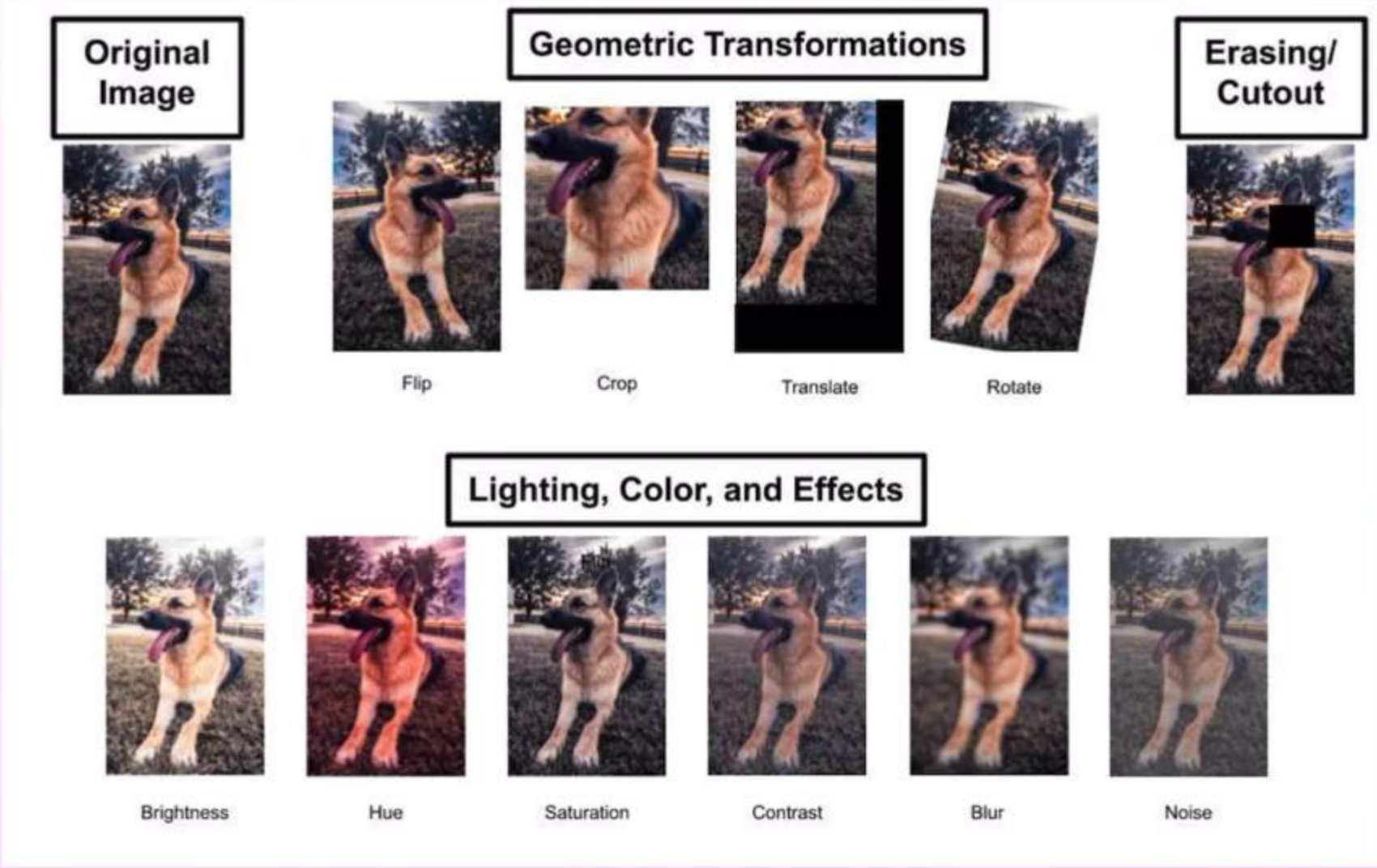
Data Augmentations



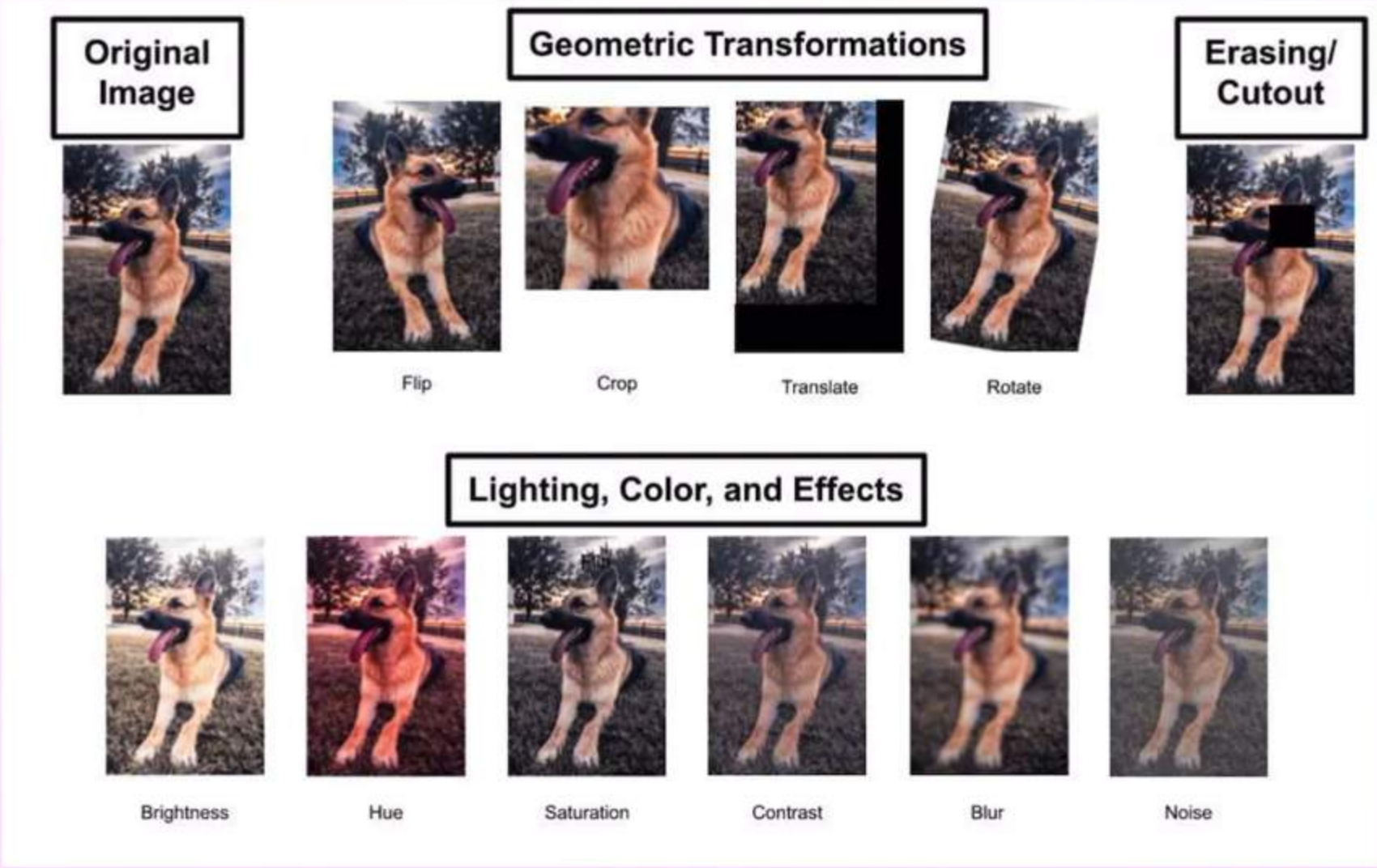
Data Augmentations



Data Augmentations



Data Augmentations



Mixup Augmentation

$$\begin{aligned} \hat{x} &= \lambda x_i + (1-\lambda)x_j, \\ \hat{y} &= \lambda y_i + (1-\lambda)y_j, \end{aligned}$$

where $\lambda \in [0, 1]$ is a random number

x0.52

x0.48

Mixup Augmentation

$$\begin{aligned} \hat{x} &= \lambda x_i + (1-\lambda)x_j, \\ \hat{y} &= \lambda y_i + (1-\lambda)y_j, \end{aligned}$$

where $\lambda \in [0, 1]$ is a random number

x0.52

x0.48

Mixup Augmentation

$$\begin{aligned} \hat{x} &= \lambda x_i + (1-\lambda) x_j, \\ \hat{y} &= \lambda y_i + (1-\lambda) y_j, \end{aligned}$$

where $\lambda \in [0, 1]$ is a random number

x0.52

x0.48

Mixup Augmentation

$$\begin{aligned} \hat{x} &= \lambda x_i + (1-\lambda) x_j, \\ \hat{y} &= \lambda y_i + (1-\lambda) y_j, \end{aligned}$$

where $\lambda \in [0, 1]$ is a random number

x0.52

+

x0.48

=

Mixup Augmentation

$$\begin{aligned} \hat{x} &= \lambda x_i + (1-\lambda)x_j, \\ \hat{y} &= \lambda y_i + (1-\lambda)y_j, \end{aligned}$$

where $\lambda \in [0, 1]$ is a random number

x0.52

x0.48

Mixup Augmentation

$$\begin{aligned} \hat{x} &= \lambda x_i + (1-\lambda)x_j, \\ \hat{y} &= \lambda y_i + (1-\lambda)y_j, \end{aligned}$$

where $\lambda \in [0, 1]$ is a random number

x0.52

x0.48

Mixup Augmentation

$$\begin{aligned} \hat{x} &= \lambda x_i + (1-\lambda) x_j, \\ \hat{y} &= \lambda y_i + (1-\lambda) y_j, \end{aligned}$$

where $\lambda \in [0, 1]$ is a random number

x0.52

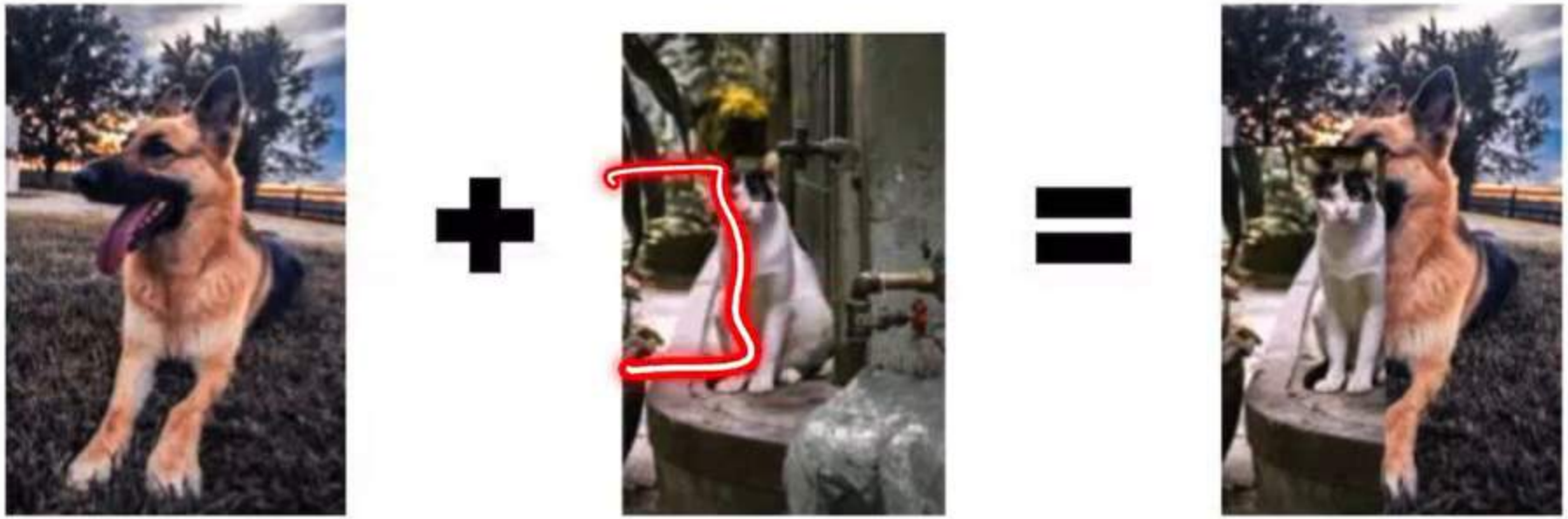
+

x0.48

=

Cutmix Augmentation

Cutmix Augmentation



Cutmix Augmentation

Cutmix Augmentation

Mosaic Augmentation

aug_-319215602_0_-238783579.jpg

aug_-1271888501_0_-749611674.jpg

aug_1462167959_0_-1659206634.jpg

aug_1474493600_0_-45389312.jpg

aug_1715045541_0_603913529.jpg

aug_1779424844_0_-589696888.jpg

Mosaic Augmentation

4 images

aug_-319215602_0_-238783579.jpg

aug_-1271888501_0_-749611674.jpg

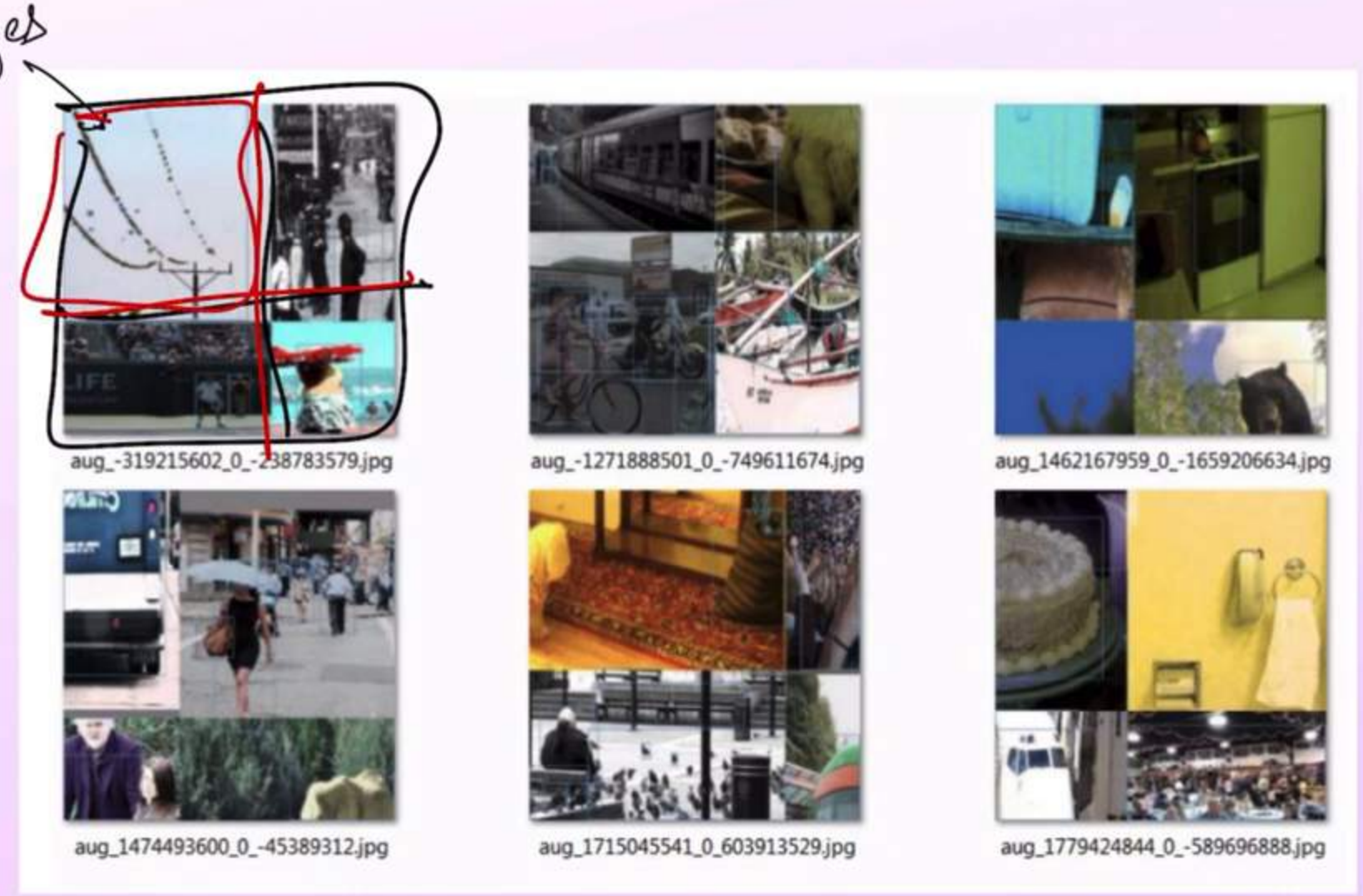
aug_1462167959_0_-1659206634.jpg

aug_1474493600_0_-45389312.jpg

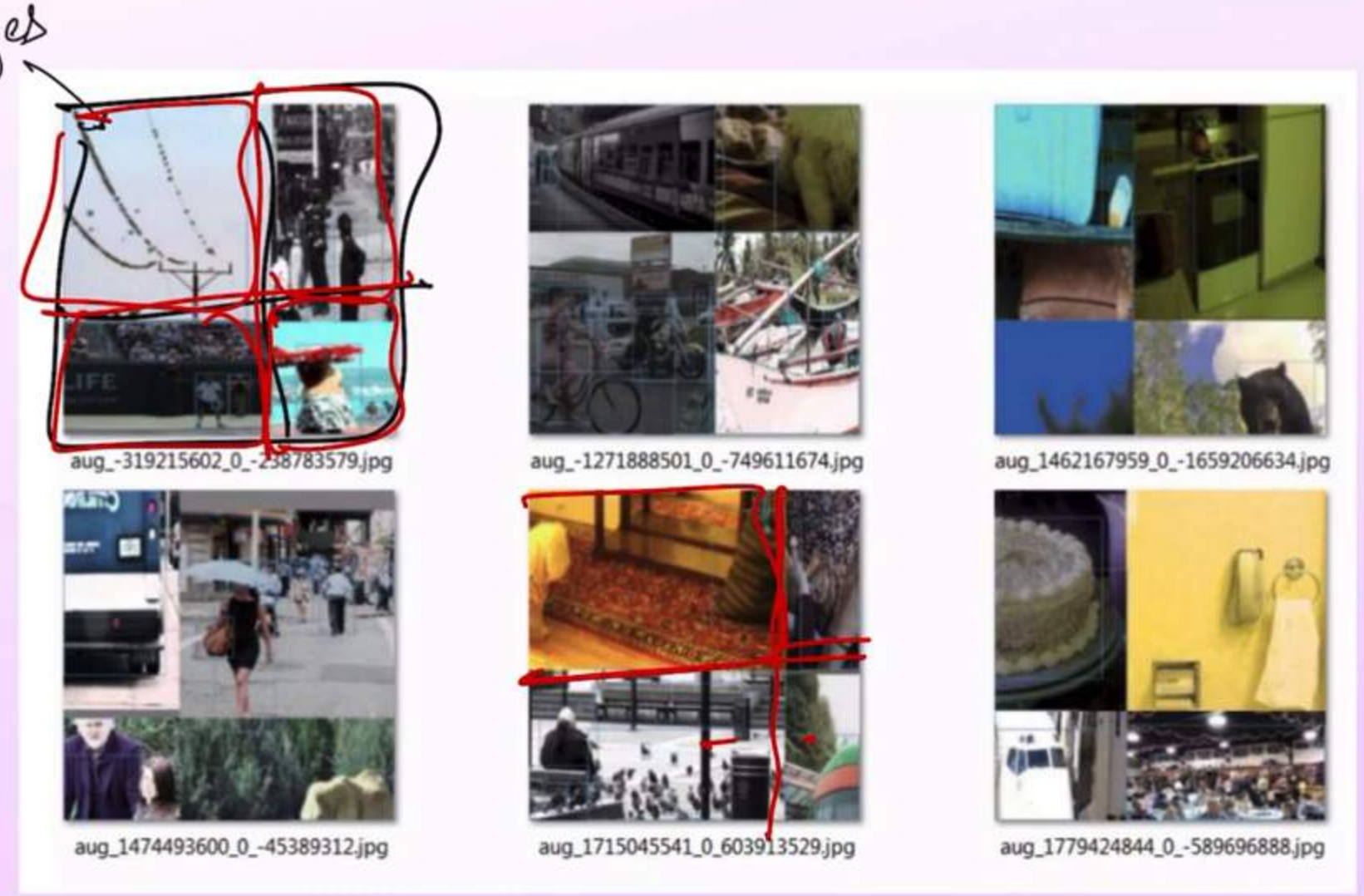
aug_1715045541_0_603913529.jpg

aug_1779424844_0_-589696888.jpg

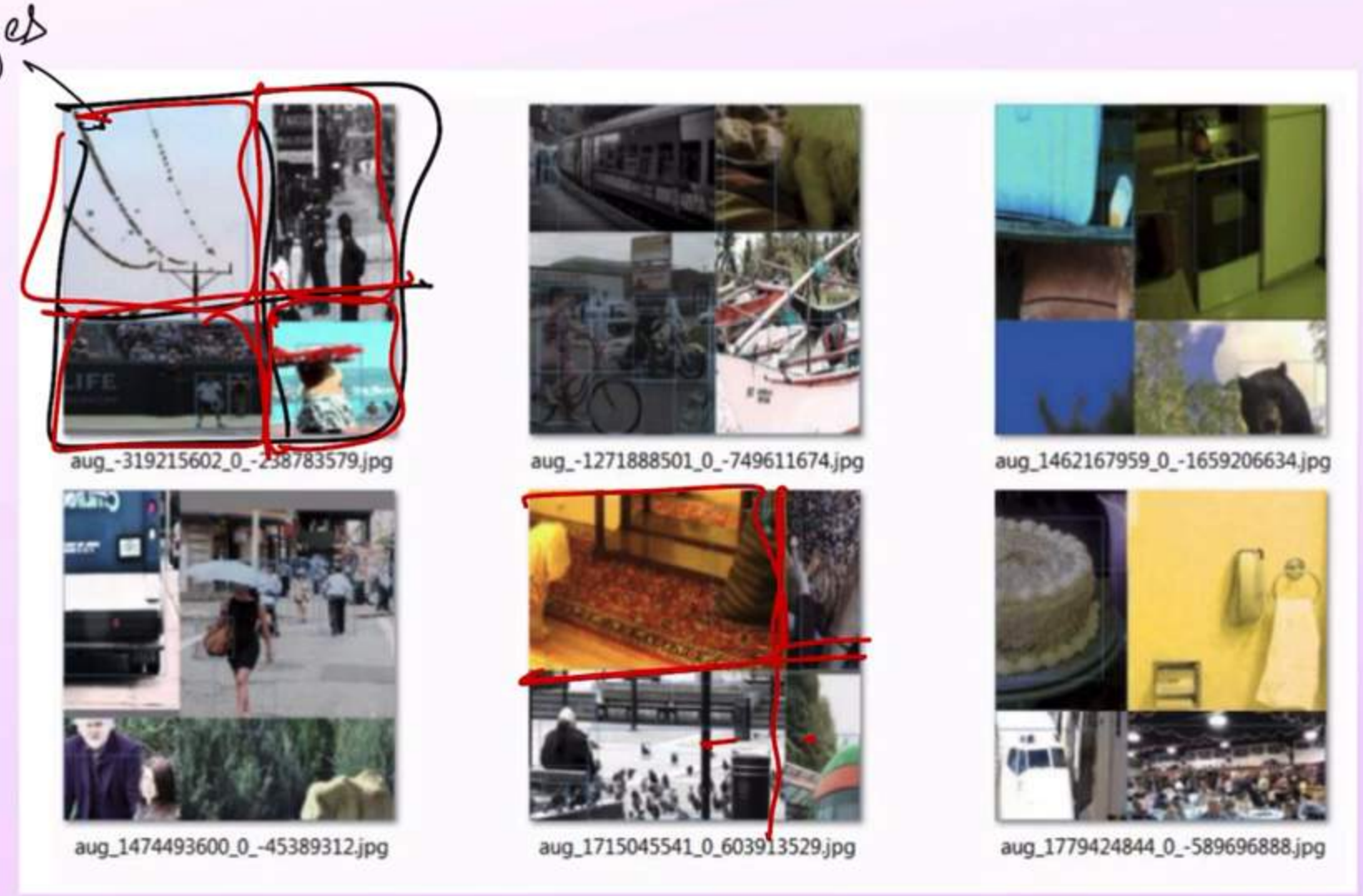
Mosaic Augmentation



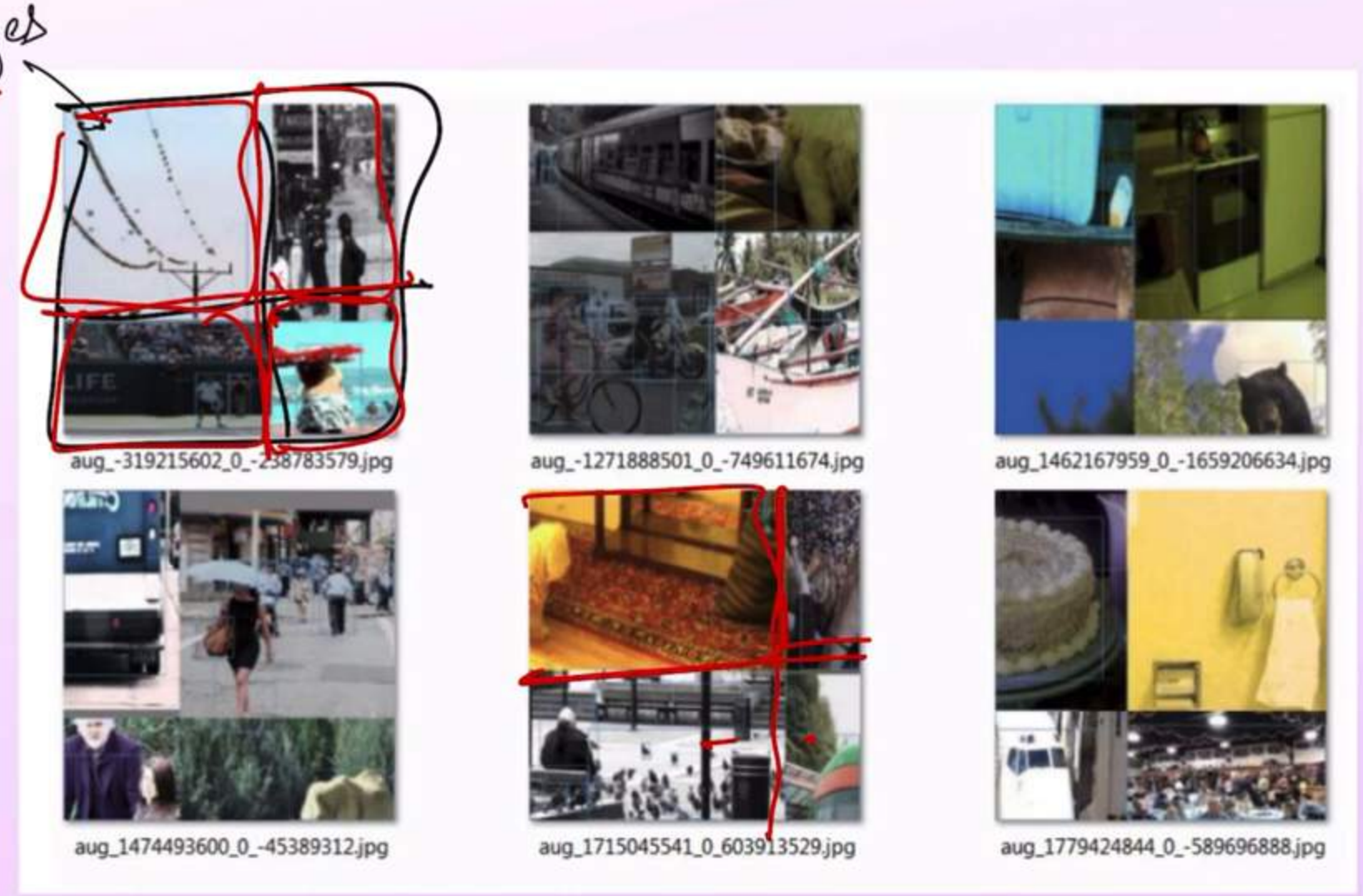
Mosaic Augmentation



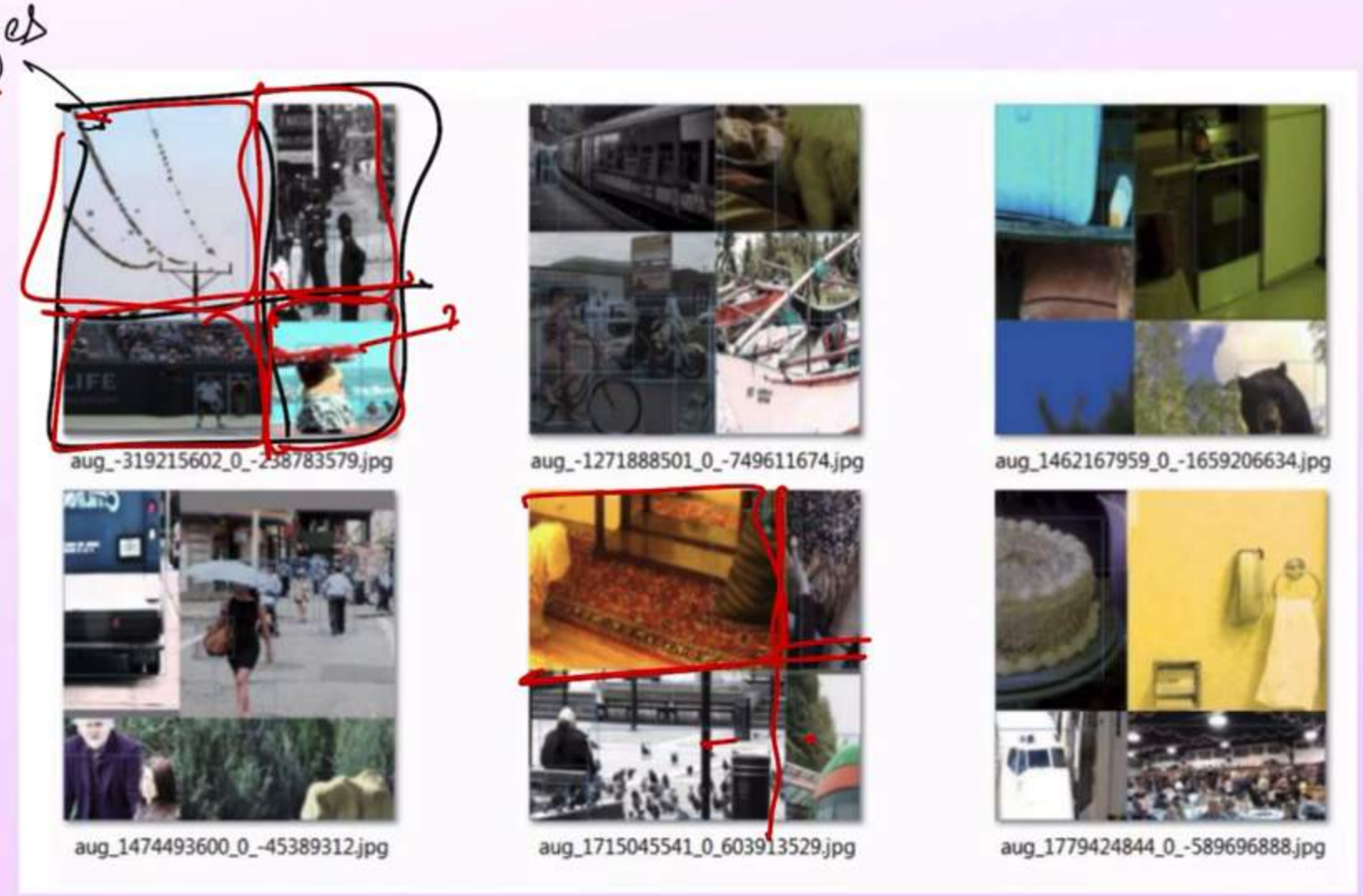
Mosaic Augmentation



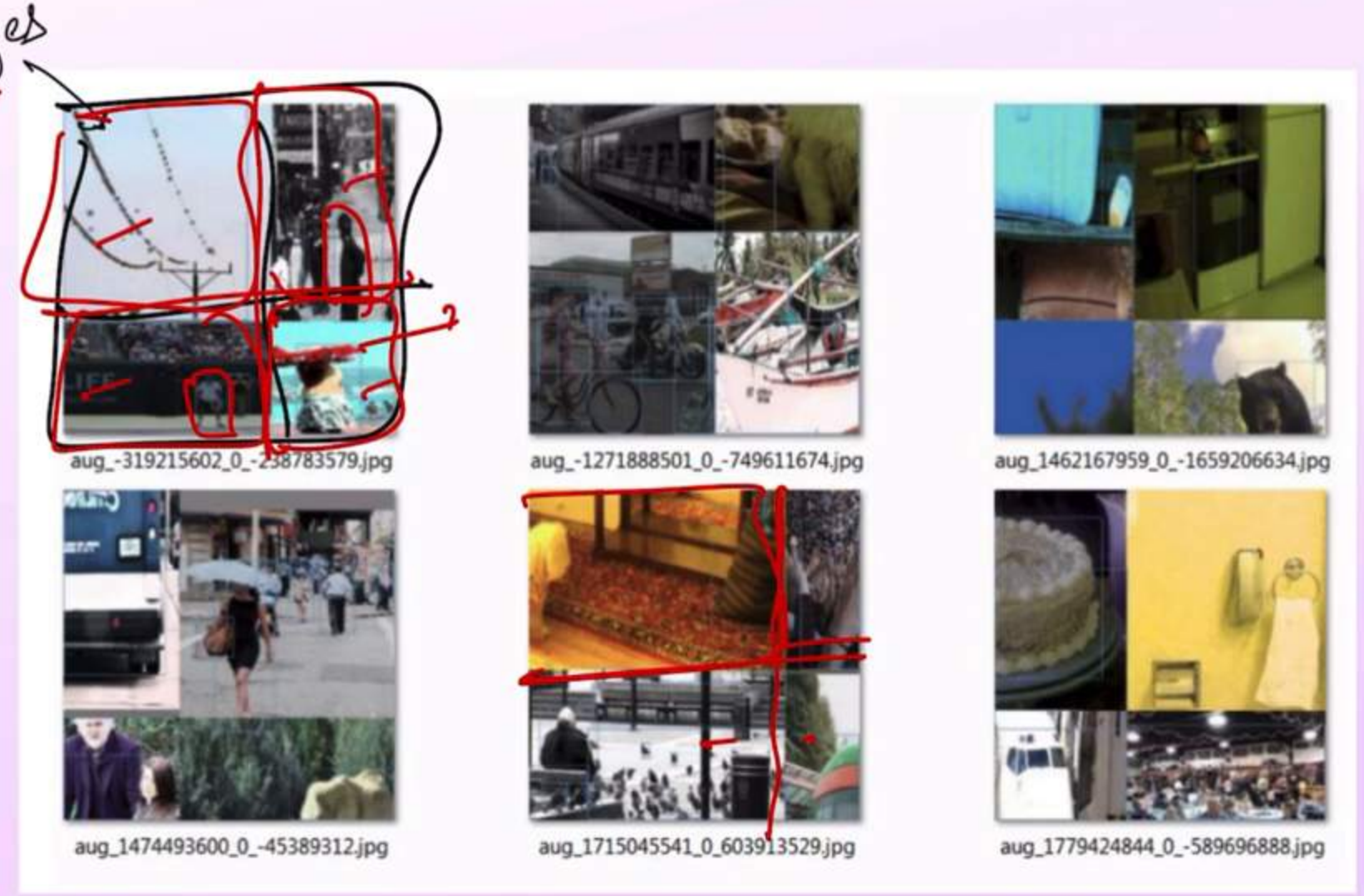
Mosaic Augmentation



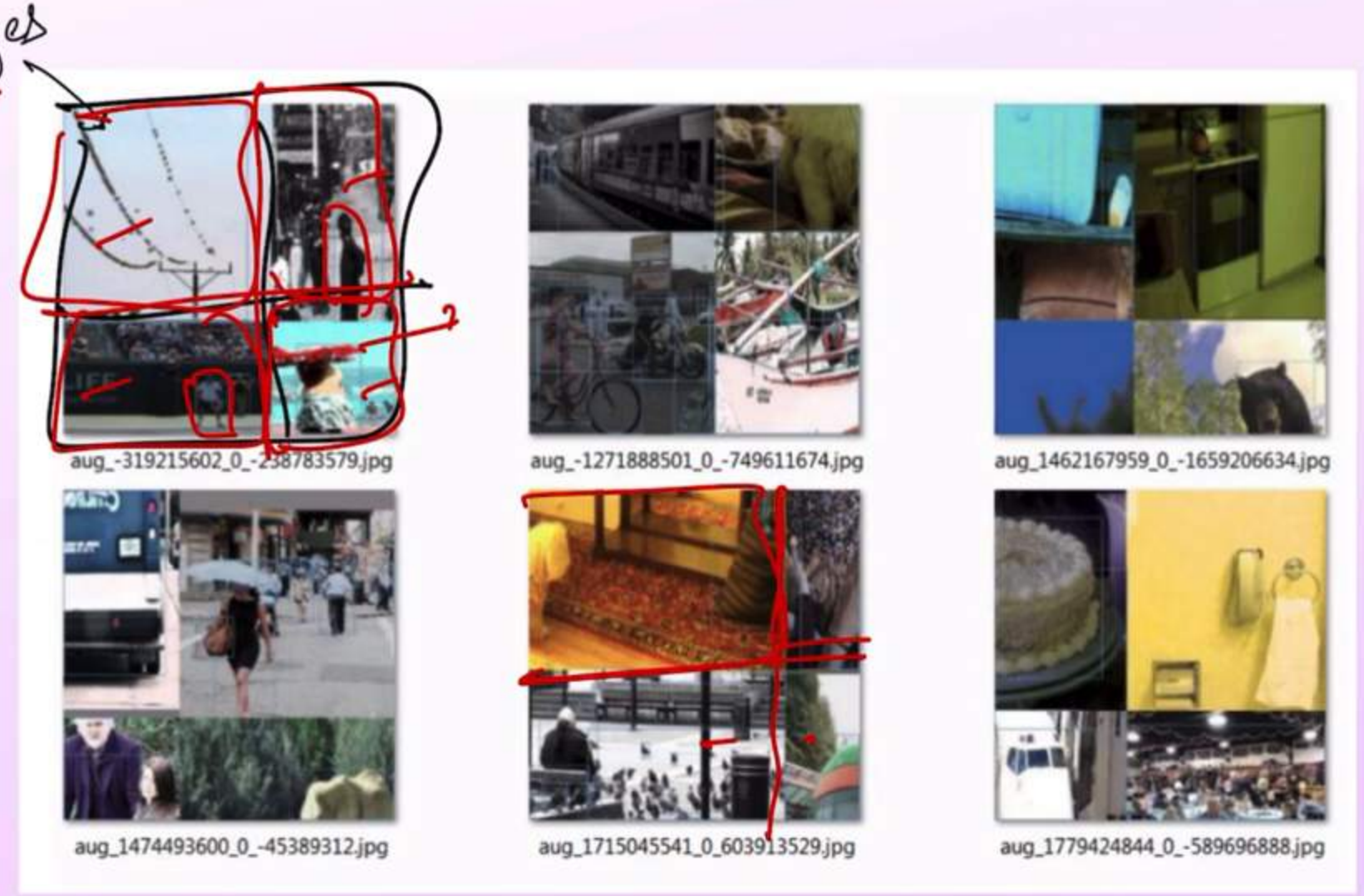
Mosaic Augmentation



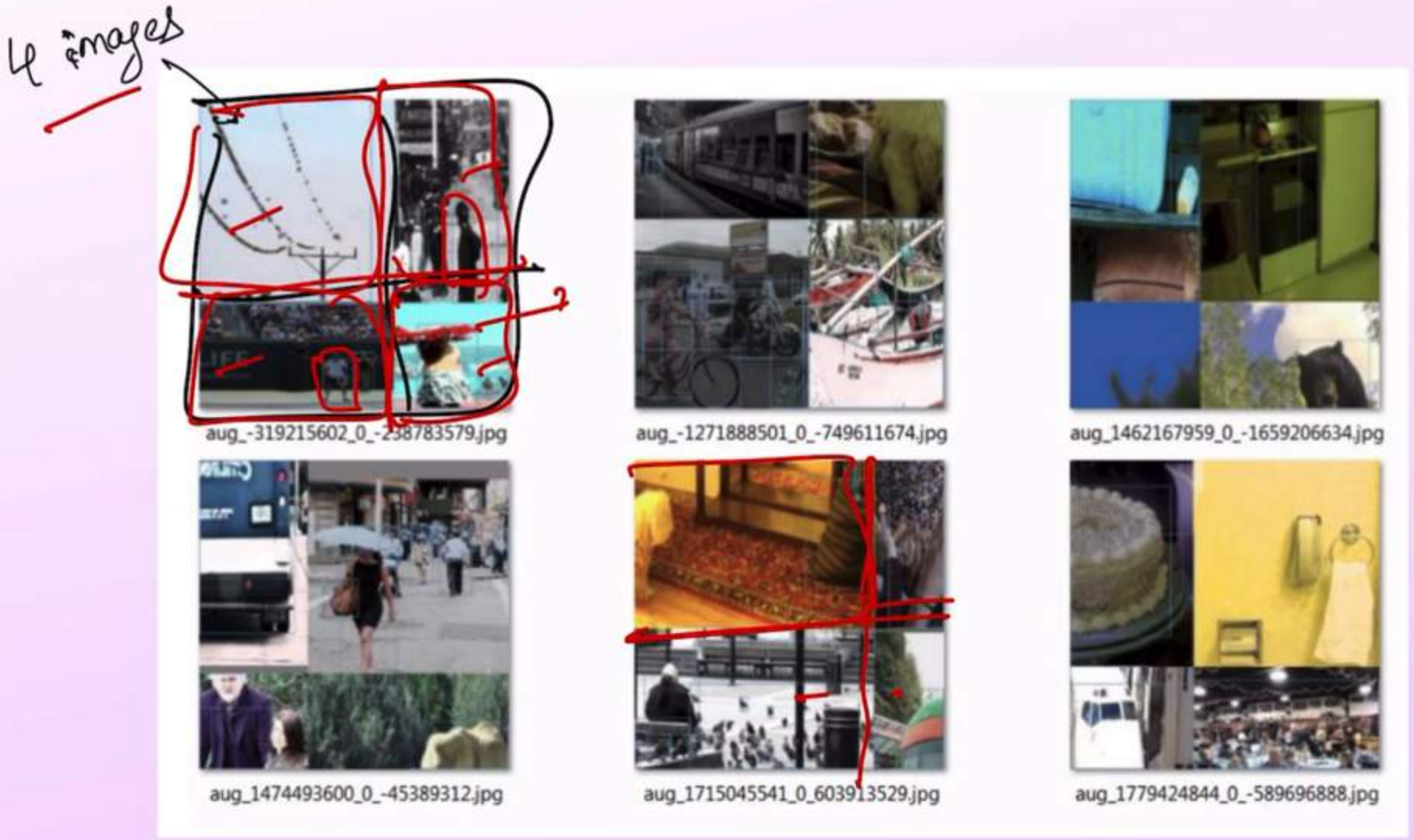
Mosaic Augmentation



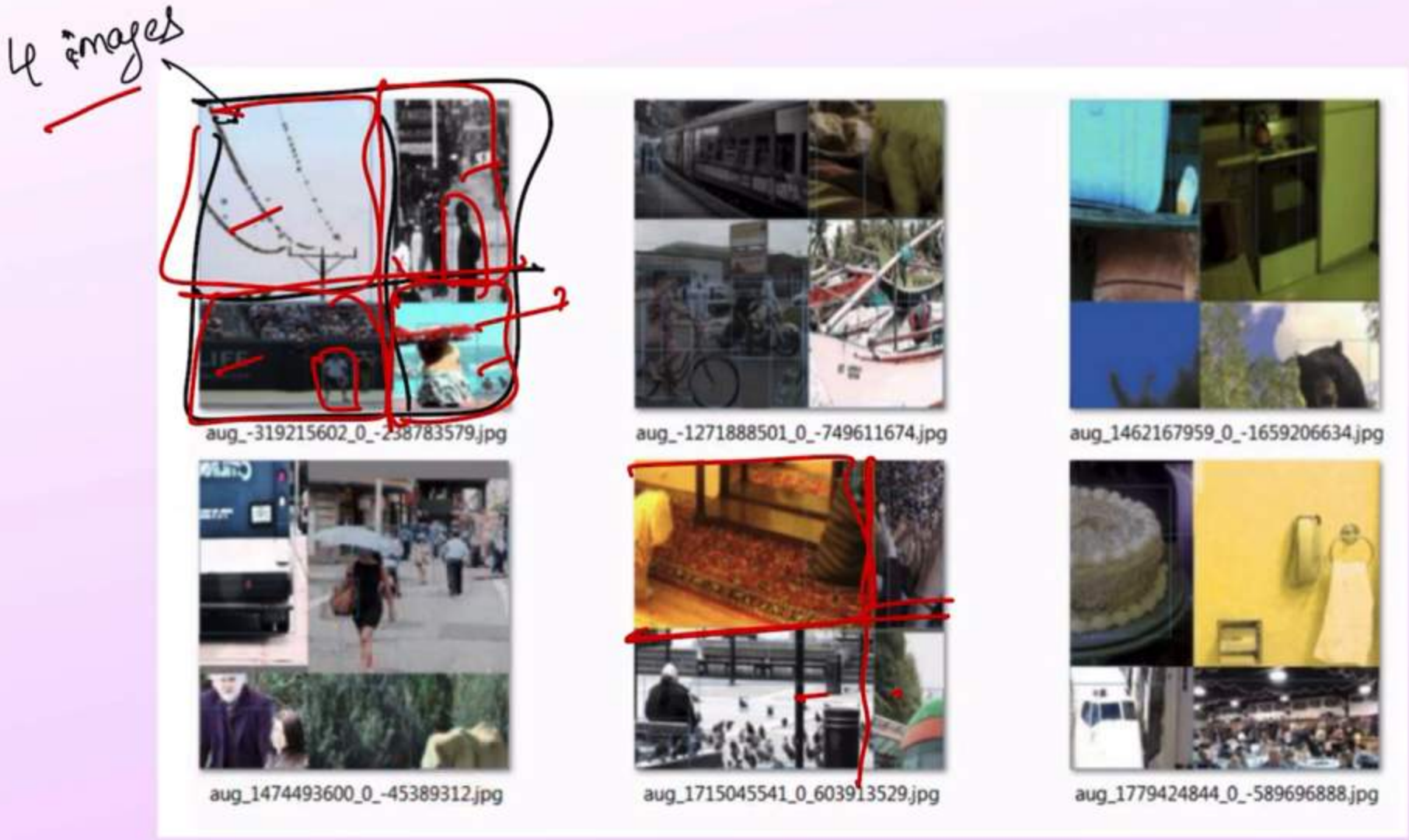
Mosaic Augmentation



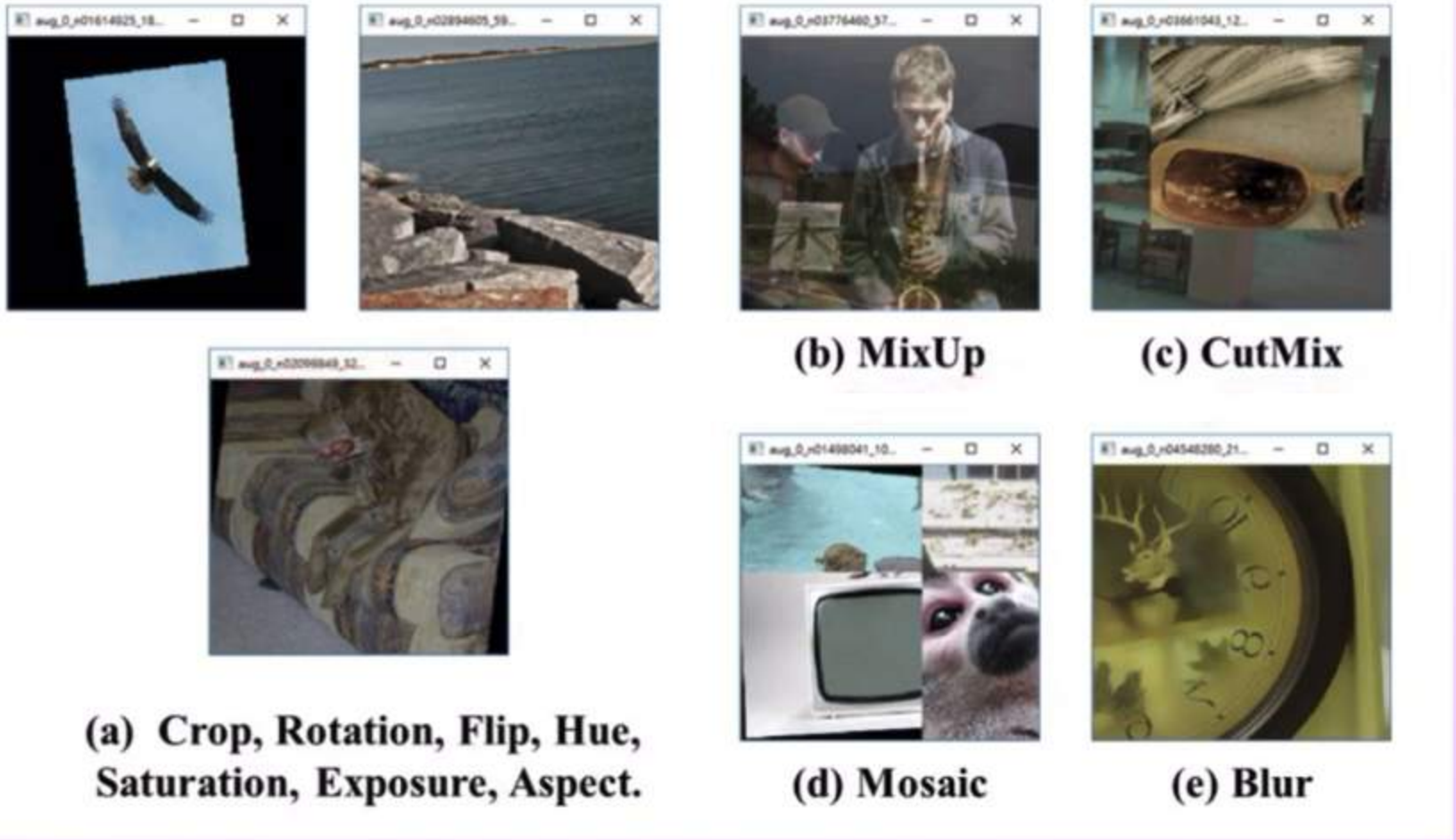
Mosaic Augmentation



Mosaic Augmentation



Data Augmentations



(a) Crop, Rotation, Flip, Hue, Saturation, Exposure, Aspect.

(b) MixUp

(c) CutMix

(d) Mosaic

(e) Blur

Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes

Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes

Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes

Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes

Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes

Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes

Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes

Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes

Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes

16 GB

320 x 320

↓

10 images

Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes

16 GB

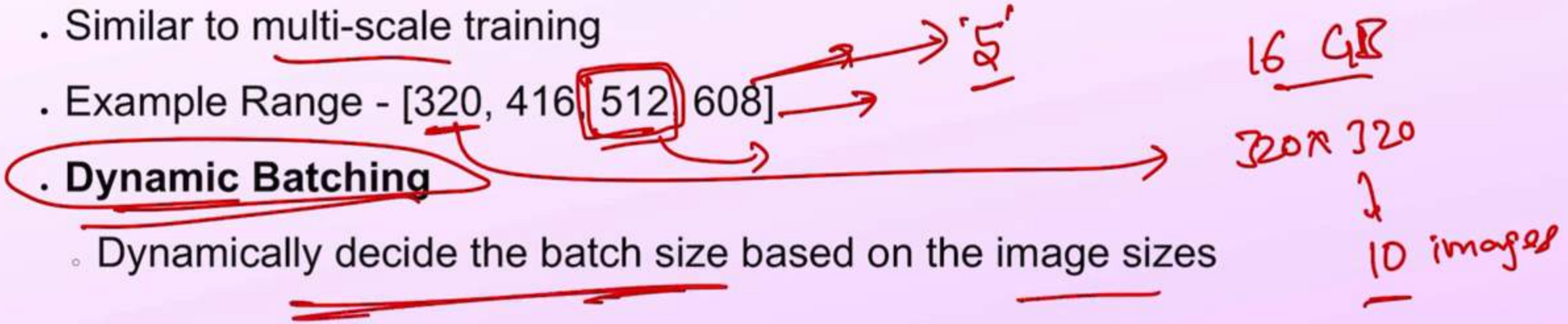
320 x 320

↓

10 images

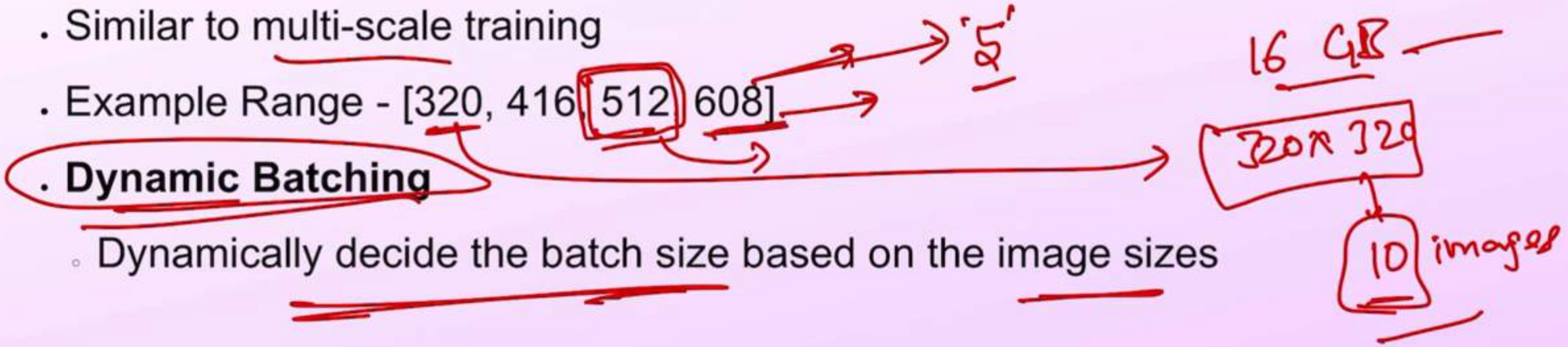
Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes



Random Training Shapes

- Randomly selecting the shape of the images during training
- Similar to multi-scale training
- Example Range - [320, 416, 512, 608]
- **Dynamic Batching**
  - Dynamically decide the batch size based on the image sizes



Random Training Shapes

Example:

Suppose the predefined image sizes are [320, 416, 512, 608].

- During training, for a particular iteration:
- The image size 416×416 is randomly selected.
- The batch size is set to 32 for this iteration (assuming 416×416 allows for this batch size given the GPU memory constraints).
- Images are resized to 416×416, and training proceeds with this batch.

In the next iteration, a different image size might be selected, and the batch size will be adjusted accordingly.

Random Training Shapes

Example:

Suppose the predefined image sizes are [320, 416, 512, 608].

- During training, for a particular iteration:
- The image size 416×416 is randomly selected.
- The batch size is set to 32 for this iteration (assuming 416×416 allows for this batch size given the GPU memory constraints).
- Images are resized to 416×416, and training proceeds with this batch.

In the next iteration, a different image size might be selected, and the batch size will be adjusted accordingly.

Random Training Shapes

Example:

Suppose the predefined image sizes are [320, 416, 512, 608].

- During training, for a particular iteration:
- The image size 416×416 is randomly selected.
- The batch size is set to 32 for this iteration (assuming 416×416 allows for this batch size given the GPU memory constraints).
- Images are resized to 416×416, and training proceeds with this batch.

In the next iteration, a different image size might be selected, and the batch size will be adjusted accordingly.

Random Training Shapes

Example:

Suppose the predefined image sizes are [320, 416, 512, 608].

- During training, for a particular iteration:
- The image size 416×416 is randomly selected.
- The batch size is set to 32 for this iteration (assuming 416×416 allows for this batch size given the GPU memory constraints).
- Images are resized to 416×416, and training proceeds with this batch.

In the next iteration, a different image size might be selected, and the batch size will be adjusted accordingly.

Multiple Anchors

- Multiple Anchor boxes for a single Groundtruth
- Usually one anchor box is assigned to groundtruth based on IOU
- In YOLOV4:
  - Take an IOU threshold
  - keep all the anchors which has IOU greater than the threshold
  - Calculate predictions and loss for all of these

- Improves the learning capability of the model and speedup the learning process
- Better handling of different object sizes

Multiple Anchors

- Multiple Anchor boxes for a single Groundtruth
- Usually one anchor box is assigned to groundtruth based on IOU
- In YOLOV4:
  - Take an IOU threshold
  - keep all the anchors which has IOU greater than the threshold
  - Calculate predictions and loss for all of these

- Improves the learning capability of the model and speedup the learning process
- Better handling of different object sizes

Multiple Anchors

- Multiple Anchor boxes for a single Groundtruth
- Usually one anchor box is assigned to groundtruth based on IOU
- In YOLOV4:
  - Take an IOU threshold
  - keep all the anchors which has IOU greater than the threshold
  - Calculate predictions and loss for all of these

- Improves the learning capability of the model and speedup the learning process
- Better handling of different object sizes

Multiple Anchors

- Multiple Anchor boxes for a single Groundtruth
- Usually one anchor box is assigned to groundtruth based on IOU
- In YOLOV4:
  - Take an IOU threshold
  - keep all the anchors which has IOU greater than the threshold
  - Calculate predictions and loss for all of these

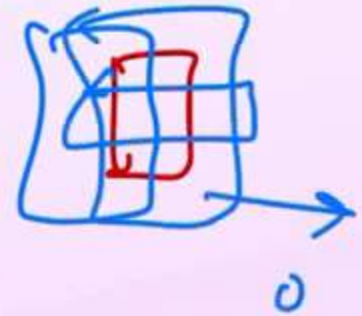
- Improves the learning capability of the model and speedup the learning process
- Better handling of different object sizes

Multiple Anchors

- Multiple Anchor boxes for a single Groundtruth
- Usually one anchor box is assigned to groundtruth based on IOU
- In YOLOV4:
  - Take an IOU threshold
  - keep all the anchors which has IOU greater than the threshold
  - Calculate predictions and loss for all of these

- Improves the learning capability of the model and speedup the learning process
- Better handling of different object sizes

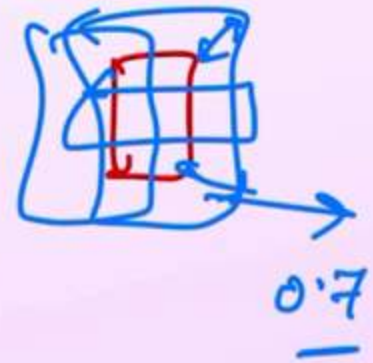
Multiple Anchors



- Multiple Anchor boxes for a single Groundtruth
- Usually one anchor box is assigned to groundtruth based on IOU
- In YOLOV4:
  - Take an IOU threshold
  - keep all the anchors which has IOU greater than the threshold
  - Calculate predictions and loss for all of these

- Improves the learning capability of the model and speedup the learning process
- Better handling of different object sizes

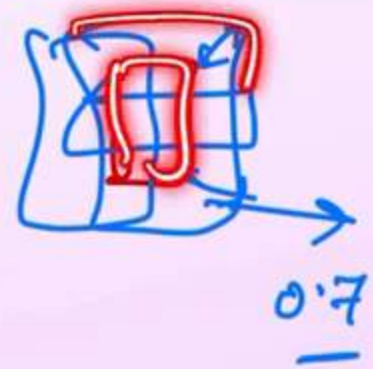
Multiple Anchors



- Multiple Anchor boxes for a single Groundtruth
- Usually one anchor box is assigned to groundtruth based on IOU
- In YOLOV4:
  - Take an IOU threshold
  - keep all the anchors which has IOU greater than the threshold
  - Calculate predictions and loss for all of these

- Improves the learning capability of the model and speedup the learning process
- Better handling of different object sizes

Multiple Anchors



- Multiple Anchor boxes for a single Groundtruth
- Usually one anchor box is assigned to groundtruth based on IOU
- In YOLOV4:
  - Take an IOU threshold
  - keep all the anchors which has IOU greater than the threshold
  - Calculate predictions and loss for all of these

- Improves the learning capability of the model and speedup the learning process
- Better handling of different object sizes

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]

- Instead of 1 or 0, we give smoothed values

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]

- Instead of 1 or 0, we give smoothed values

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]

- Instead of 1 or 0, we give smoothed values

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]

- Instead of 1 or 0, we give smoothed values

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]

- Instead of 1 or 0, we give smoothed values

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]

- Instead of 1 or 0, we give smoothed values

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]

- Instead of 1 or 0, we give smoothed values

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]

- Instead of 1 or 0, we give smoothed values

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]
- Instead of 1 or 0, we give smoothed values

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]

- Instead of 1 or 0, we give smoothed values
- But why do we need this?

Label Smoothing

- Example:
  - 5 classes - cat, dog, horse, tiger, lion
  - image-1: Tiger, Label - [0,0,0,1,0]
  - image-2: cat, Label - [1,0,0,0,0]
  - etc.,
- Label Smoothing:
  - image-1: Tiger, Label - [0.025, 0.025, 0.025, 0.9, 0.025]
  - image-2: Cat, Label - [0.9, 0.025, 0.025, 0.025, 0.025]

- Instead of 1 or 0, we give smoothed values
- But why do we need this?

Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels

Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels

0.9

Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels

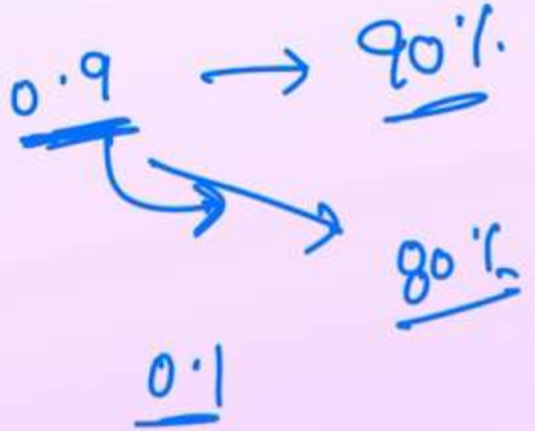
0.9 → 90%

Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels

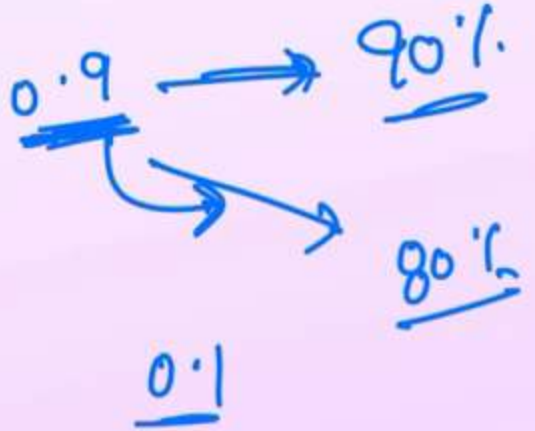
Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels



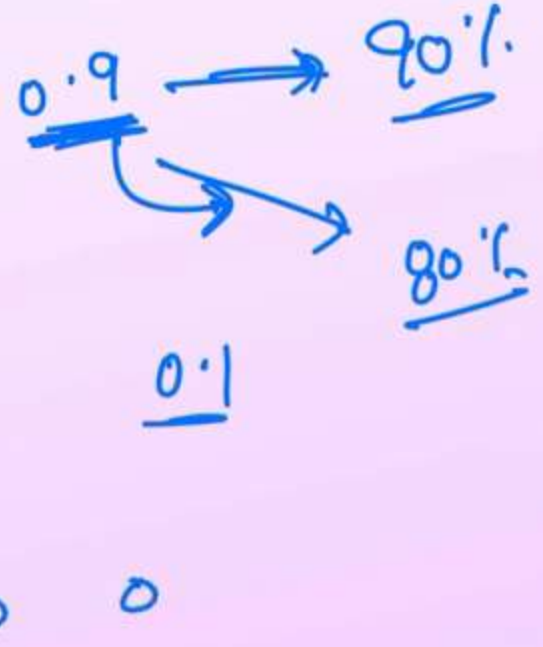
Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels



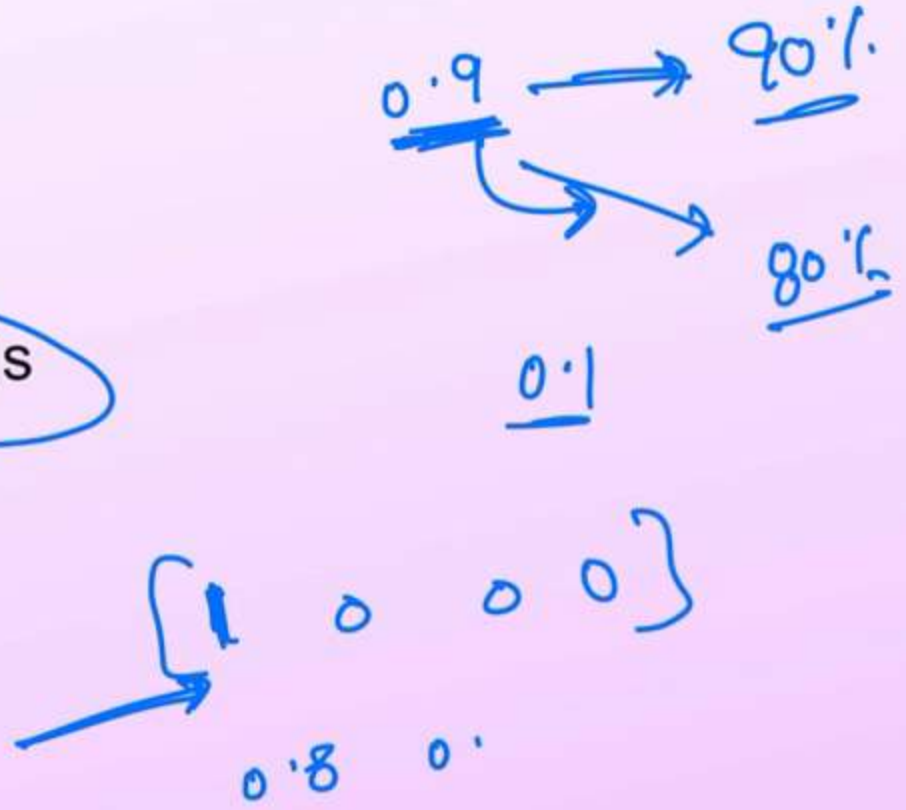
Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels



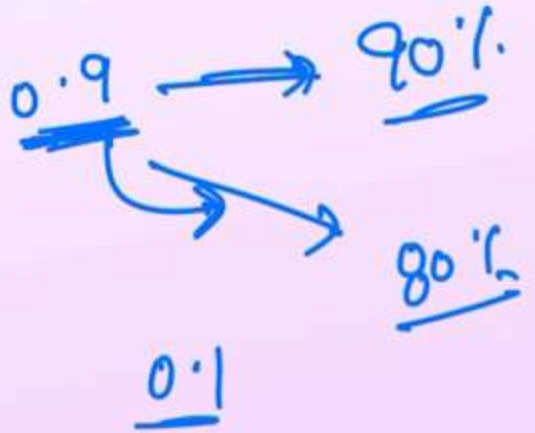
Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels



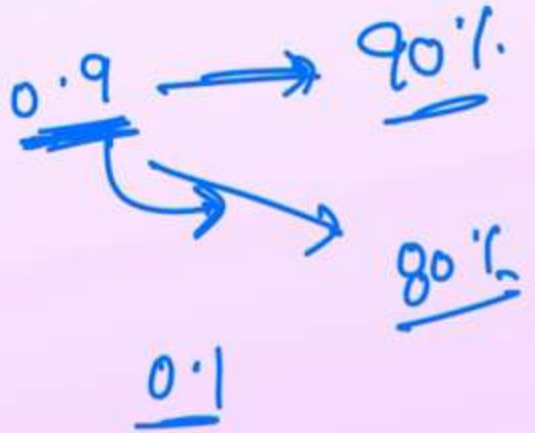
Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels



Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels



Label Smoothing

- Prevents overconfidence
- Improves calibration
- Enhances Regularization
- Mitigates the Impact of Noisy Labels

- How to do this? - [0.025, 0.025, 0.025, 0.9, 0.025]

Label Smoothing

- α - smoothing parameter
- C - total classes
- smoothed label:
  - $y = (1-\alpha) + \alpha/C$ for true class
  - $y = \alpha/C$ for other classes

Label Smoothing

- α - smoothing parameter
- C - total classes
- smoothed label:
  - $y = (1-\alpha) + \alpha/C$ for true class
  - $y = \alpha/C$ for other classes

$\alpha = 0.1$

$C = 5$

Label Smoothing

- α - smoothing parameter
- C - total classes → 5
- smoothed label:
  - $y = (1-\alpha) + \alpha/C$ for true class
  - $y = \alpha/C$ for other classes

$\alpha = 0.1$

[1 0 0 0 0]

Label Smoothing

- α - smoothing parameter
- C - total classes → 5
- smoothed label:
  - $y = (1-\alpha) + \alpha/C$ for true class
  - $y = \alpha/C$ for other classes

$\alpha = 0.1$

[1 0 0 0 0]

↓

1 − 0.1 + 0.1/

Label Smoothing

- α - smoothing parameter
- C - total classes → 5
- smoothed label:
  - $y = (1-\alpha) + \alpha/C$ for true class
  - $y = \alpha/C$ for other classes

$\alpha = 0.1$

[1 0 0 0 0]

↓

$(1 - 0.1 + 0.1/5)$

Label Smoothing

- α - smoothing parameter
- C - total classes → 5
- smoothed label:
  - $y = (1-\alpha) + \alpha/C$ for true class
  - $y = \alpha/C$ for other classes

$\alpha = 0.1$

[1 0 0 0 0]

↓

$(1 - 0.1 + 0.1/5)$

$(0.9 + 0.02)$

[

Label Smoothing

- α - smoothing parameter
- C - total classes → 5
- smoothed label:
  - $y = (1-\alpha) + \alpha/C$ for true class
  - $y = \alpha/C$ for other classes

$\alpha = 0.1$

[1 0 0 0 0]

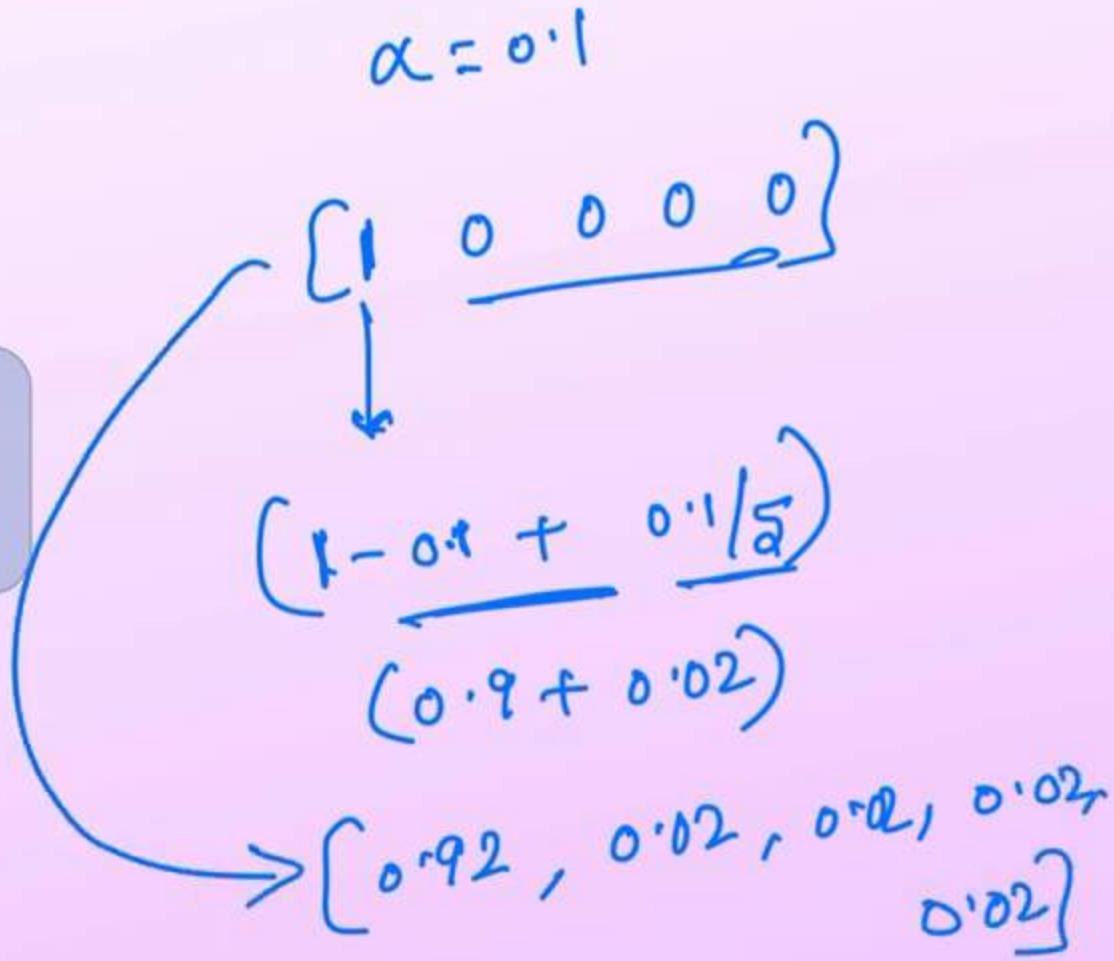
$(1 - 0.1 + 0.1/5)$

$(0.9 + 0.02)$

[0.92,

Label Smoothing

- α - smoothing parameter
- C - total classes → 5
- smoothed label:
  - $y = (1-\alpha) + \alpha/C$ for true class
  - $y = \alpha/C$ for other classes



Label Smoothing

```
import tensorflow as tf
from tensorflow.keras.losses import CategoricalCrossentropy

# Define label smoothing factor
label_smoothing = 0.1

# Initialize the loss function with label smoothing
loss_fn = CategoricalCrossentropy(label_smoothing=label_smoothing)

# Example true labels and predictions
y_true = tf.constant([[1.0, 0.0, 0.0]])  # One-hot encoded label for class A
y_pred = tf.constant([[0.8, 0.1, 0.1]])  # Model's predictions

# Compute the loss
loss = loss_fn(y_true, y_pred)
print('Loss with label smoothing:', loss.numpy())
```

Label Smoothing

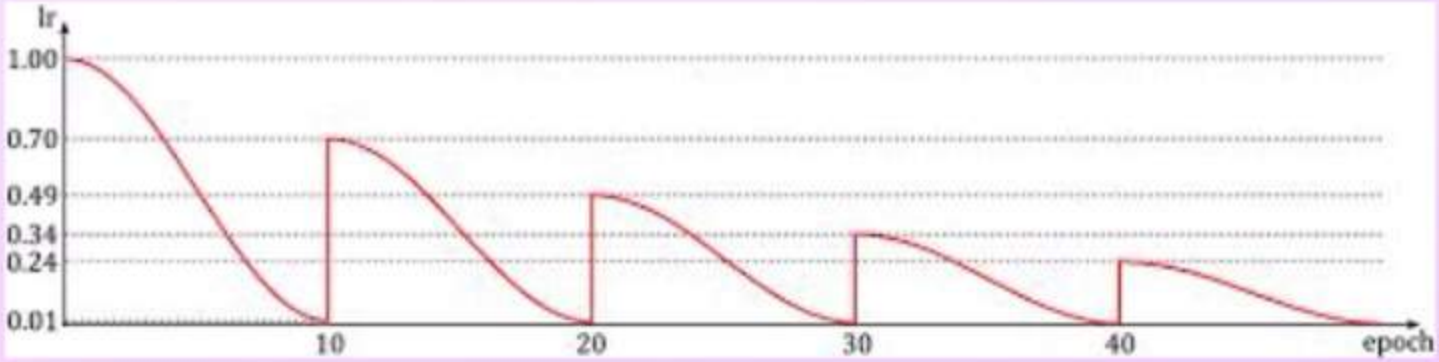
```
import tensorflow as tf
from tensorflow.keras.losses import CategoricalCrossentropy

# Define label smoothing factor
label_smoothing = 0.1

# Initialize the loss function with label smoothing
loss_fn = CategoricalCrossentropy(label_smoothing=label_smoothing)

# Example true labels and predictions
y_true = tf.constant([[1.0, 0.0, 0.0]])  # One-hot encoded label for class A
y_pred = tf.constant([[0.8, 0.1, 0.1]])  # Model's predictions

# Compute the loss
loss = loss_fn(y_true, y_pred)
print('Loss with label smoothing:', loss.numpy())
```

Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```

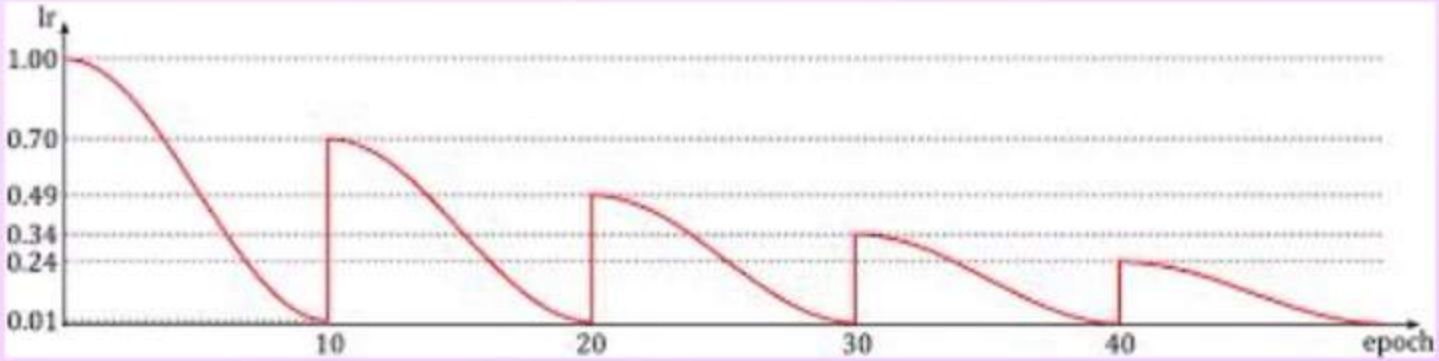


Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```

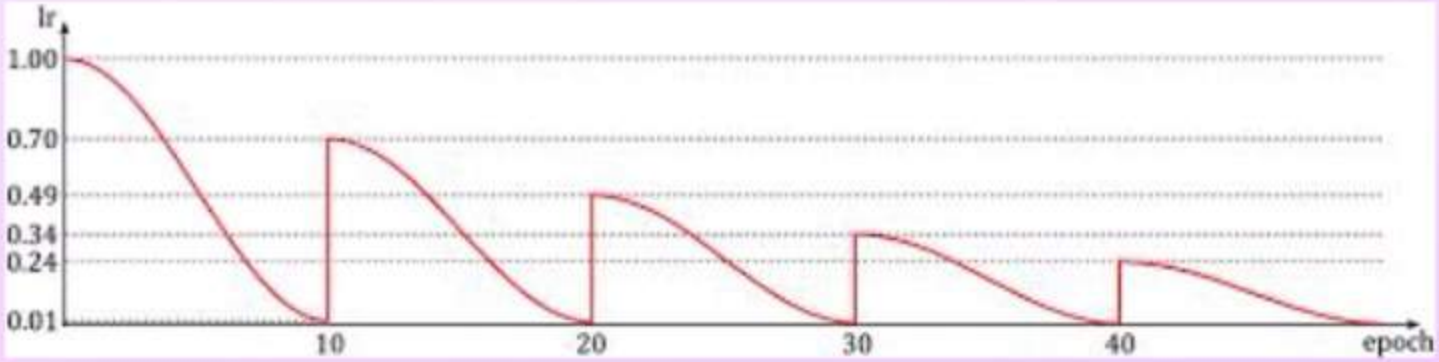


Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```

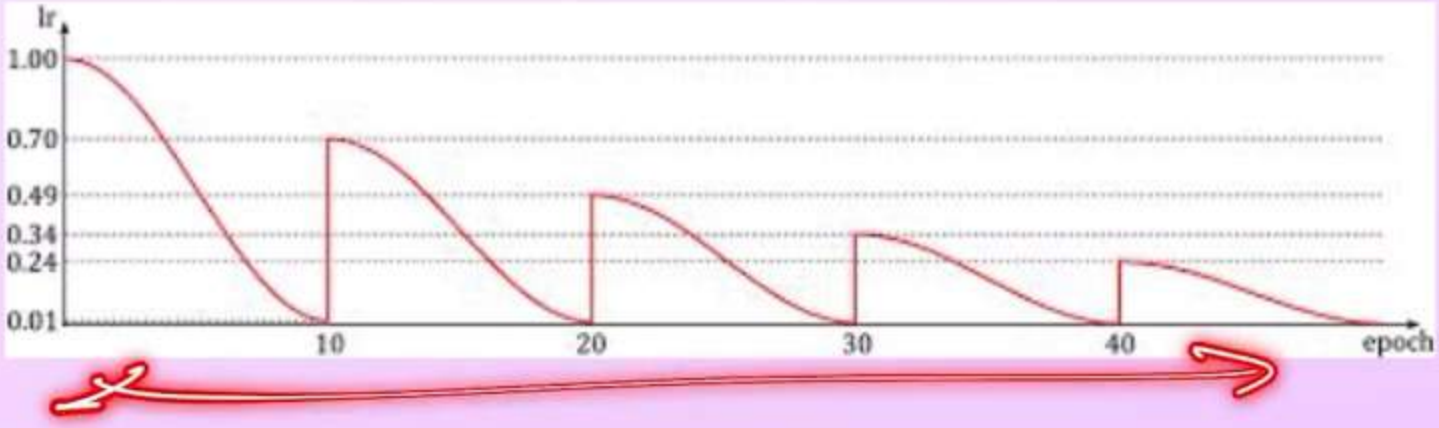


Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```

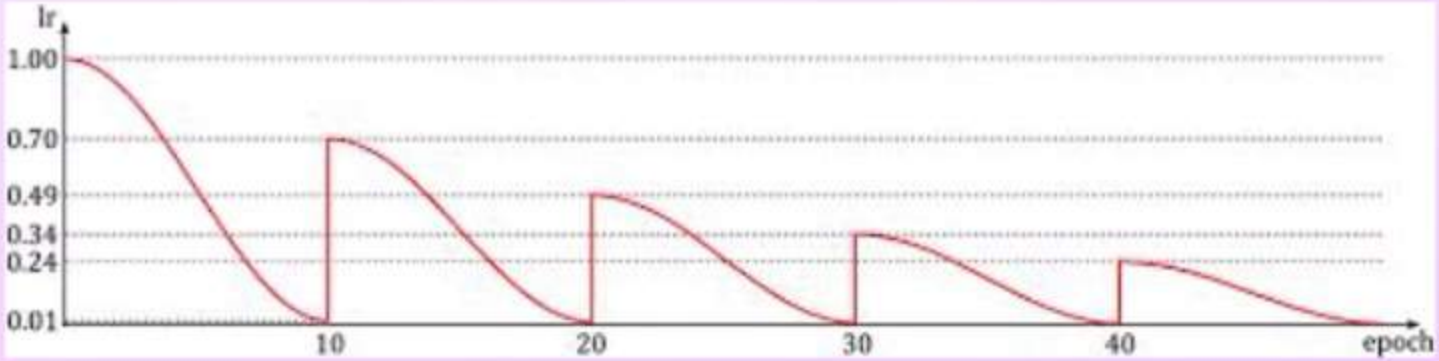


Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1,
min_lr=0.001, gamma=0.7)
```

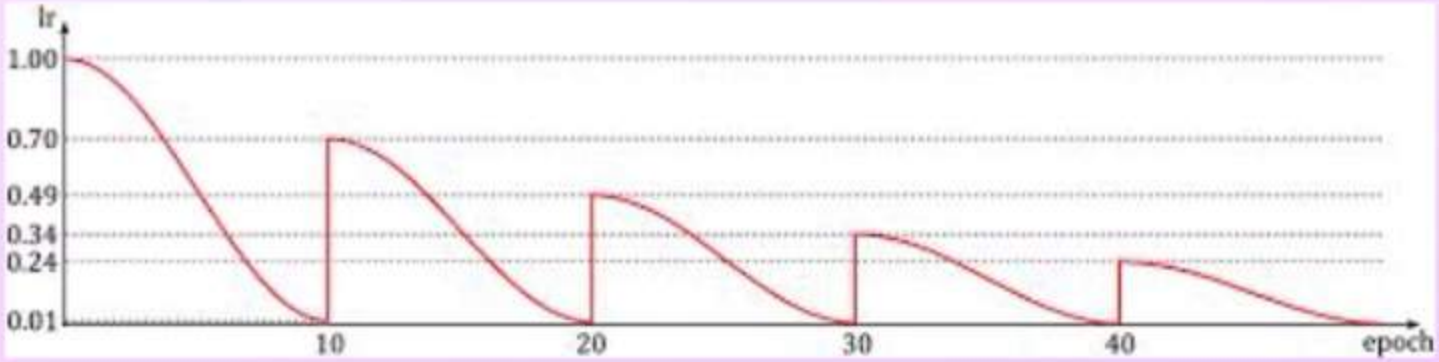


Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```

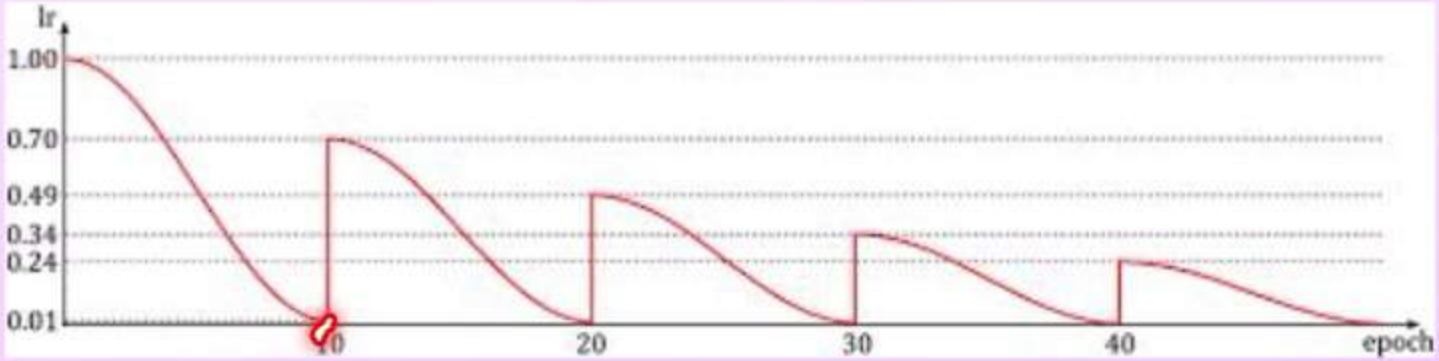


Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```

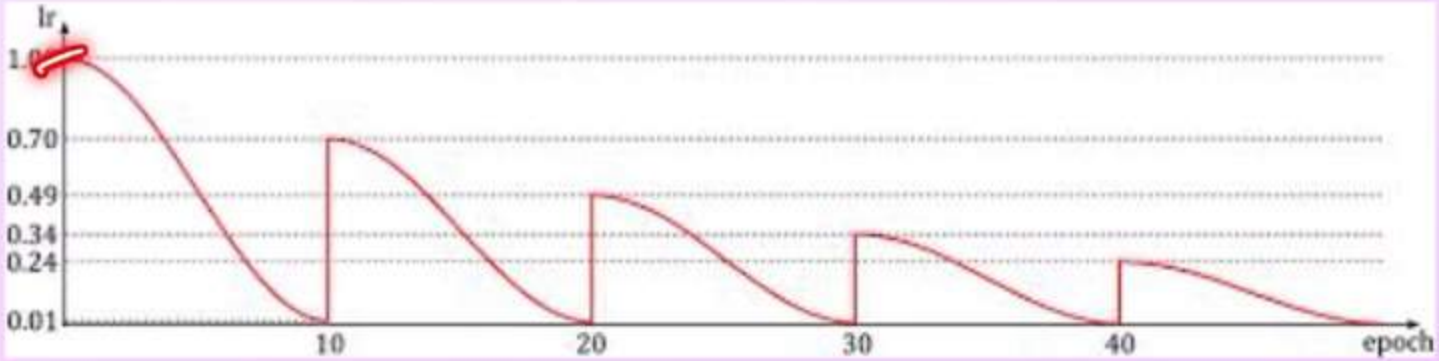


Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```

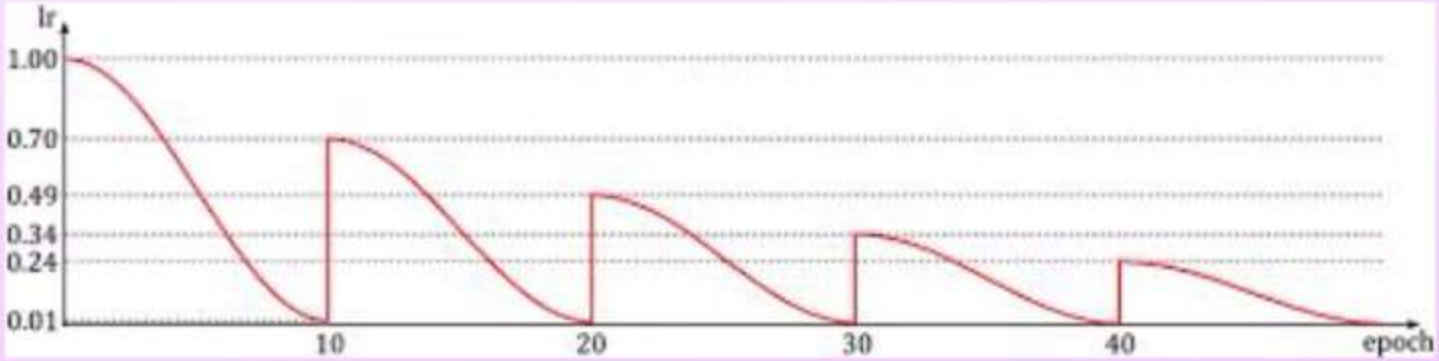


Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```

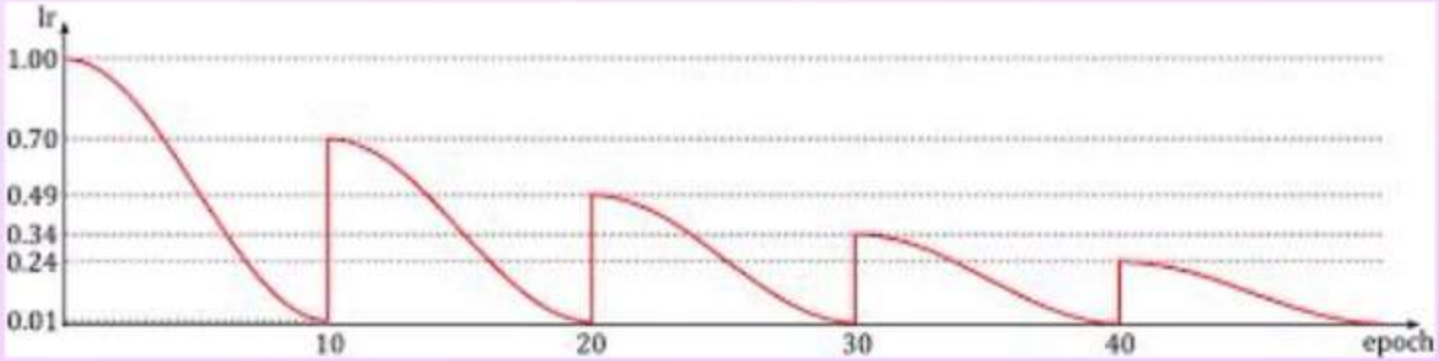


Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```



Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

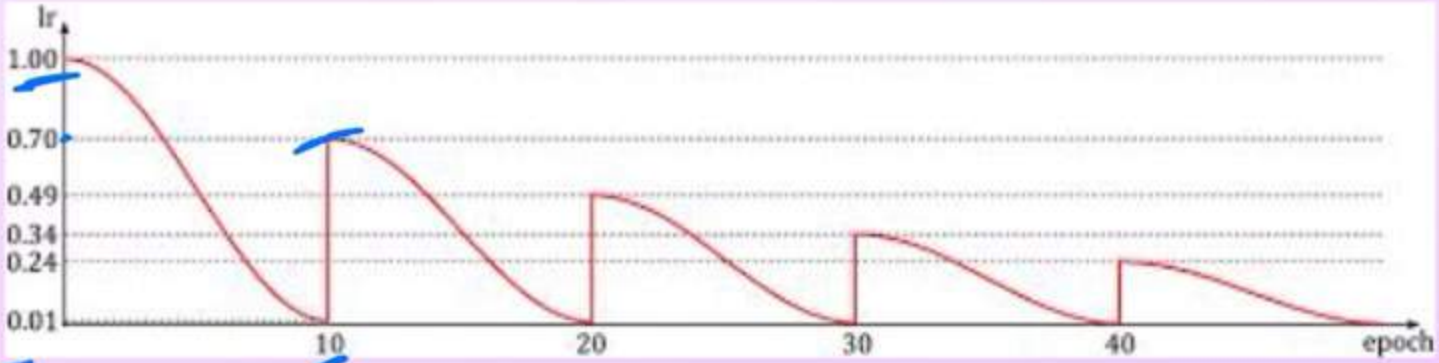
```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```

Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```



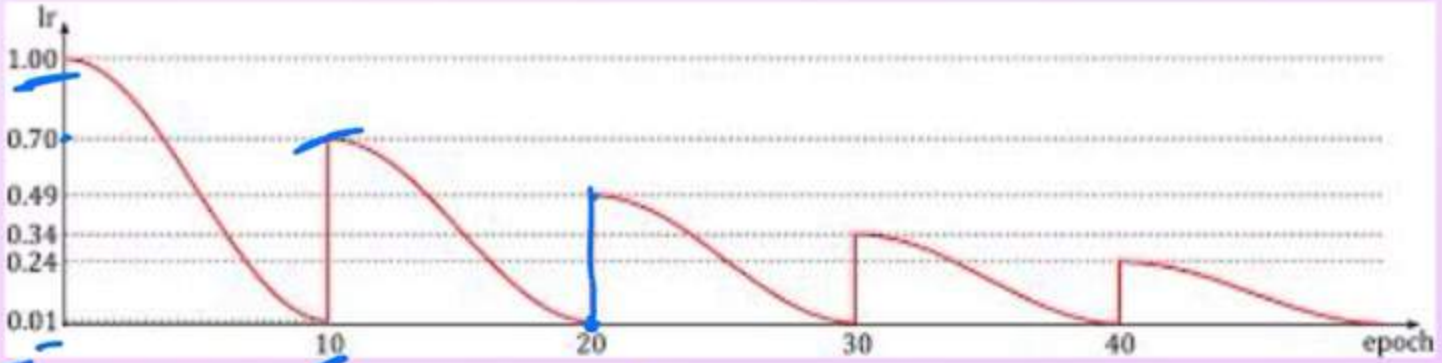
1 × 0.7 ⇒ 0.7

Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)
```



1 × 0.7 ⇒ 0.7

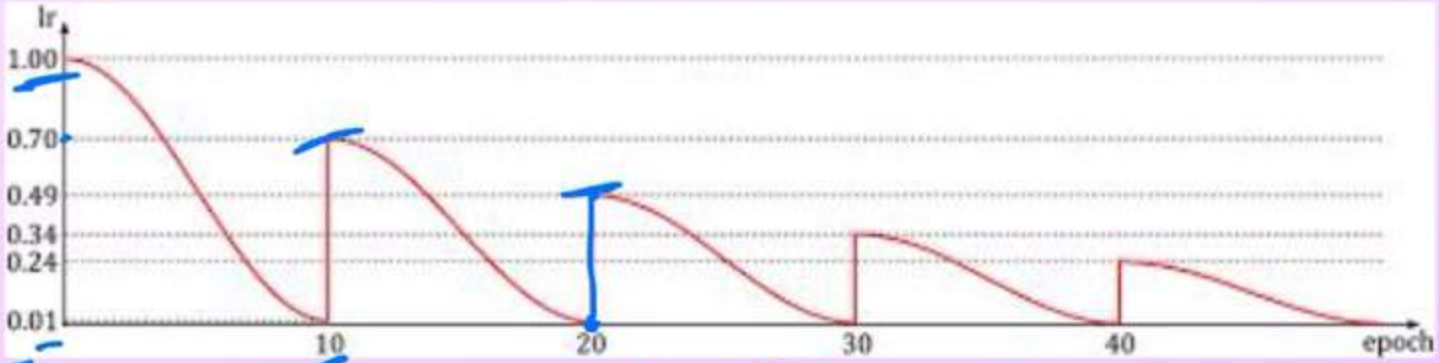
0.7

Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1, min_lr=0.001, gamma=0.7)



1 x 0.7 ⇒ 0.7

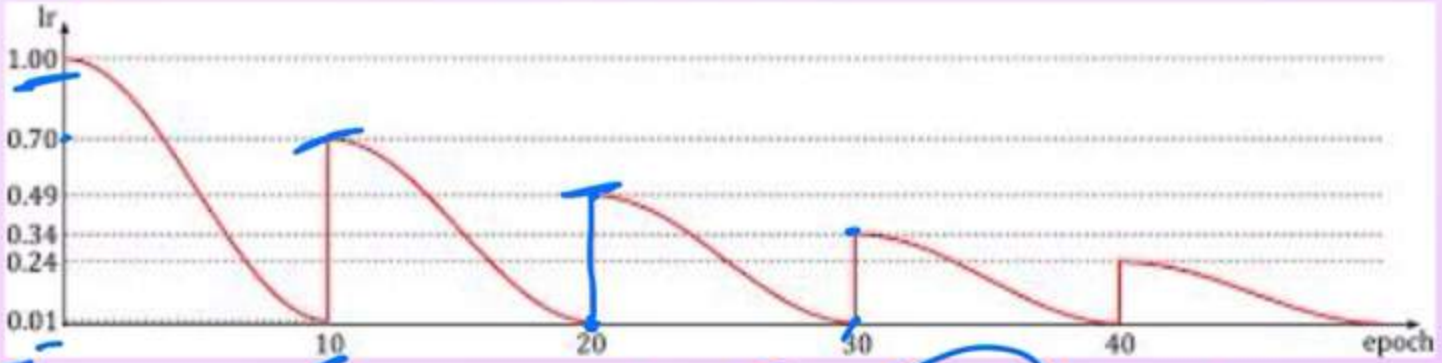
0.7 x 0.7 = 0.49

Cosine Annealing Lr Scheduler

- Lr Scheduler - adjusts the learning rate during training
- Starts with high value and decreases over time
- Cosine Annealing scheduler

  - follows cosine wave pattern of reducing values

```
CosineAnnealingWarmupRestarts(optimizer, first_cycle_steps=10, cycle_mult=1.0, max_lr=0.1,
min_lr=0.001, gamma=0.7)
```

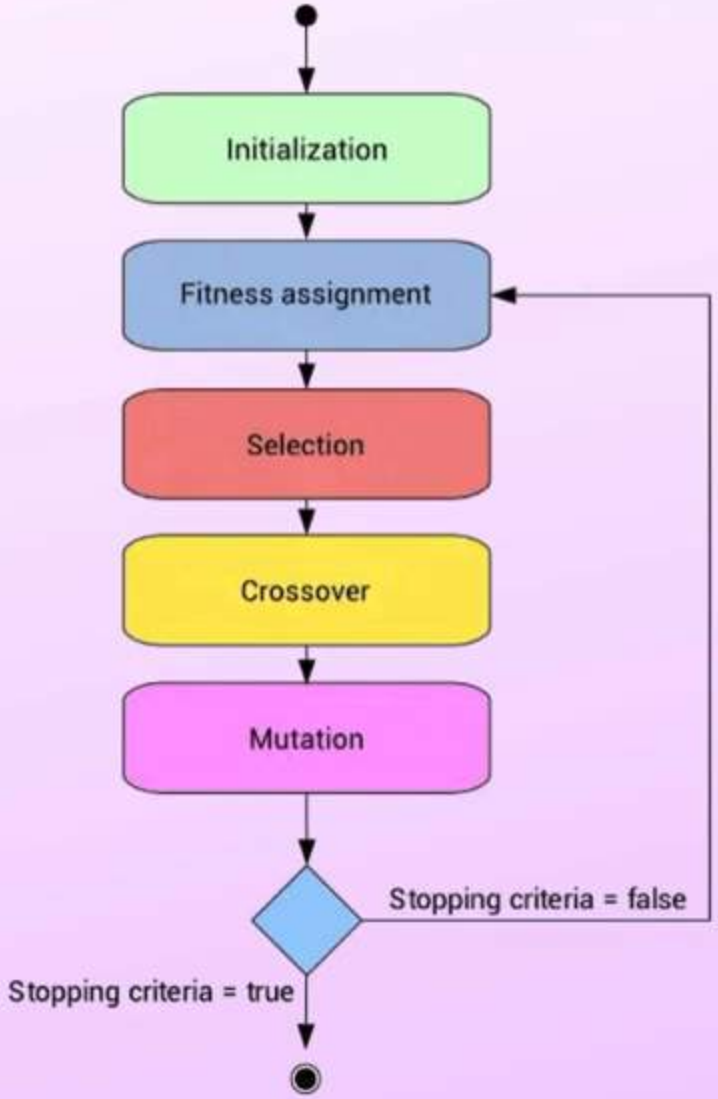


1 x 0.7 ⇒ 0.7

0.7 x 0.7 = 0.49 → gamma →

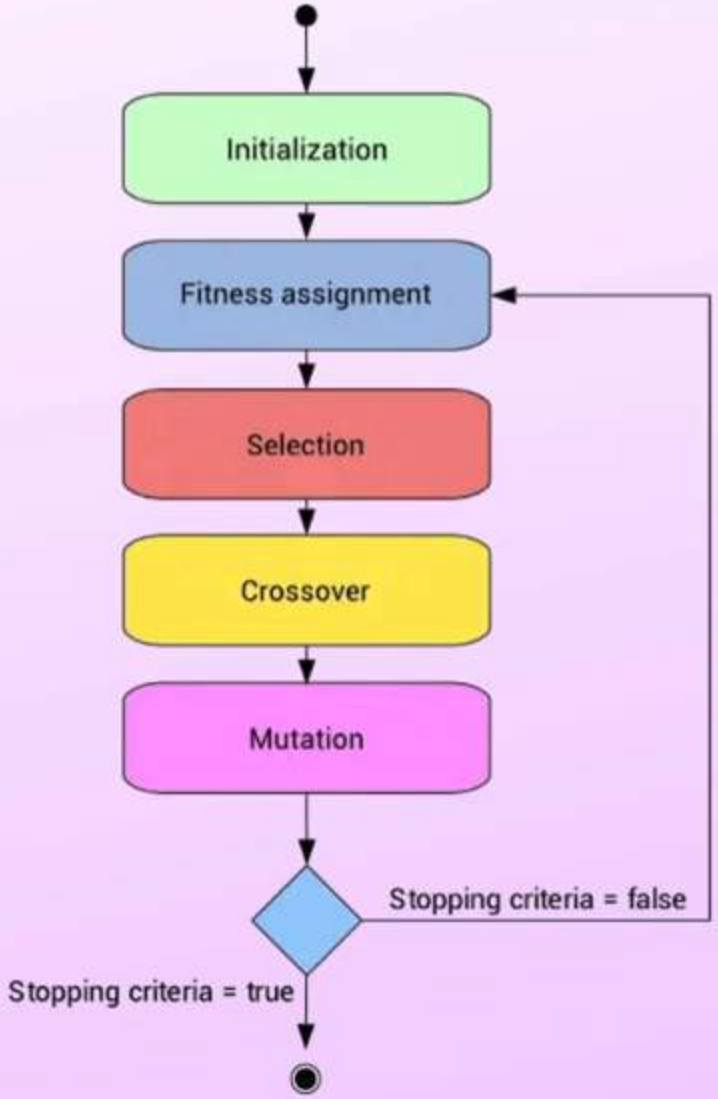
Genetic Algorithms

- Optimal Hyperparameters



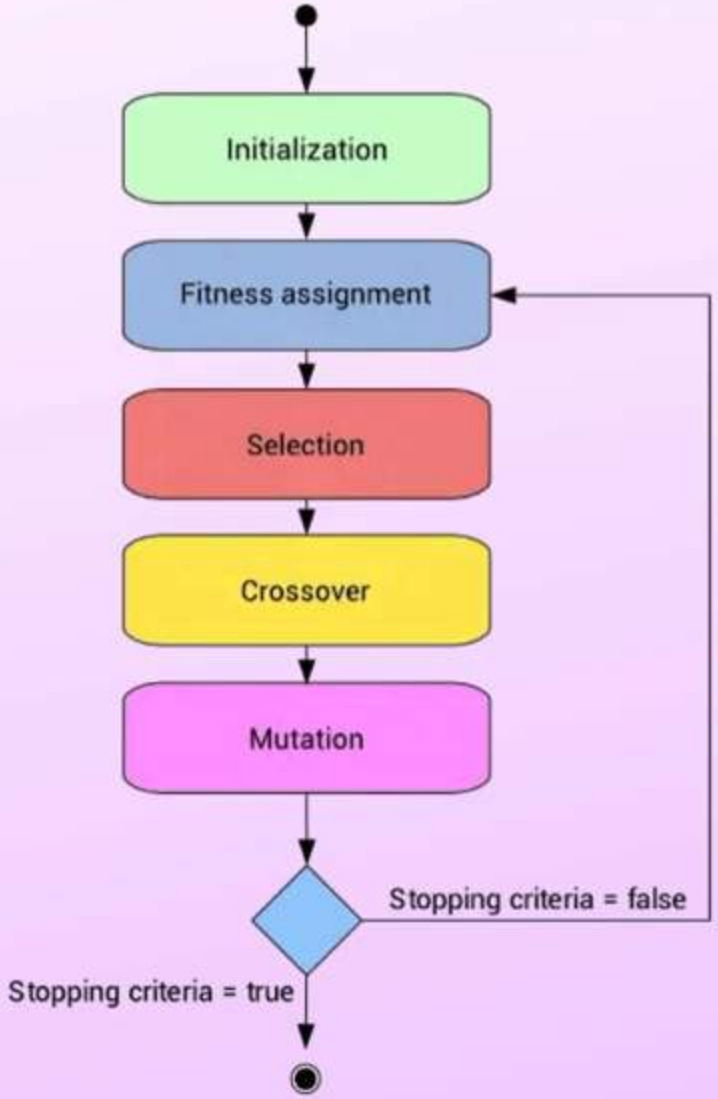
Genetic Algorithms

- Optimal Hyperparameters



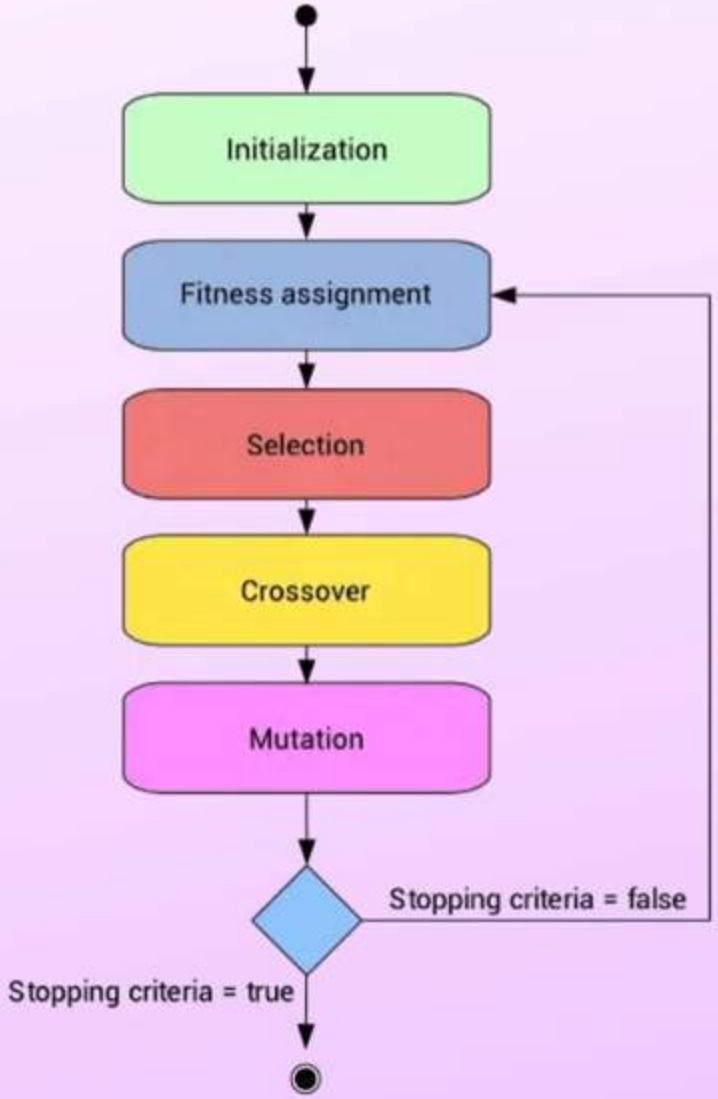
Genetic Algorithms

- Optimal Hyperparameters



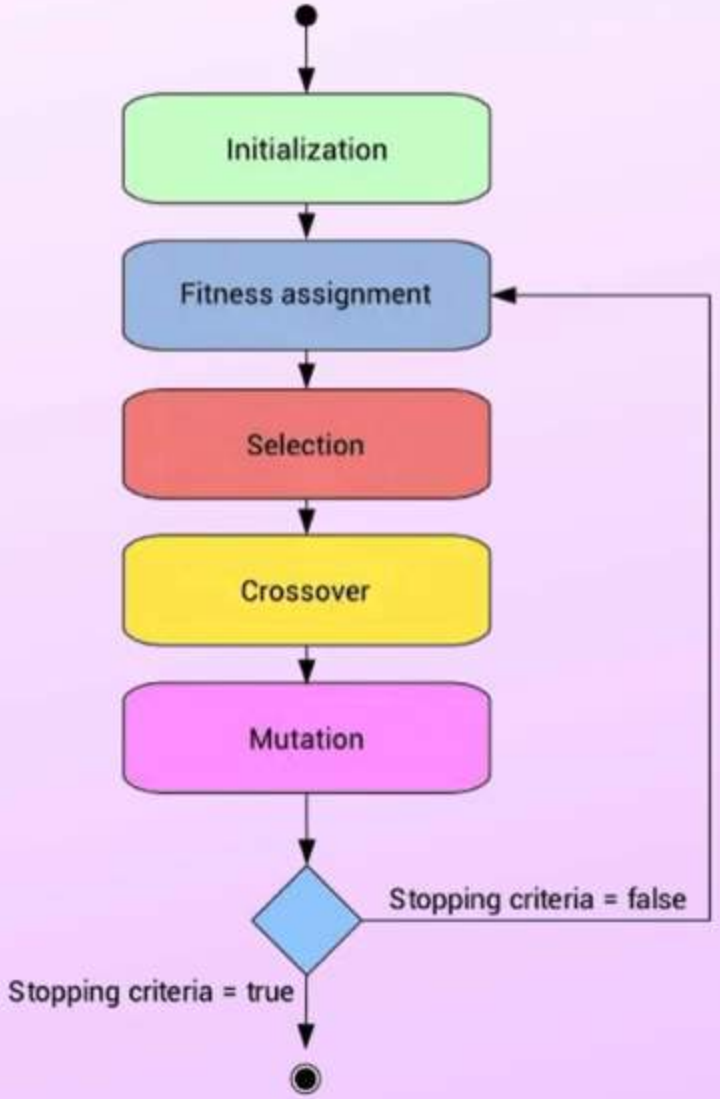
Genetic Algorithms

- Optimal Hyperparameters



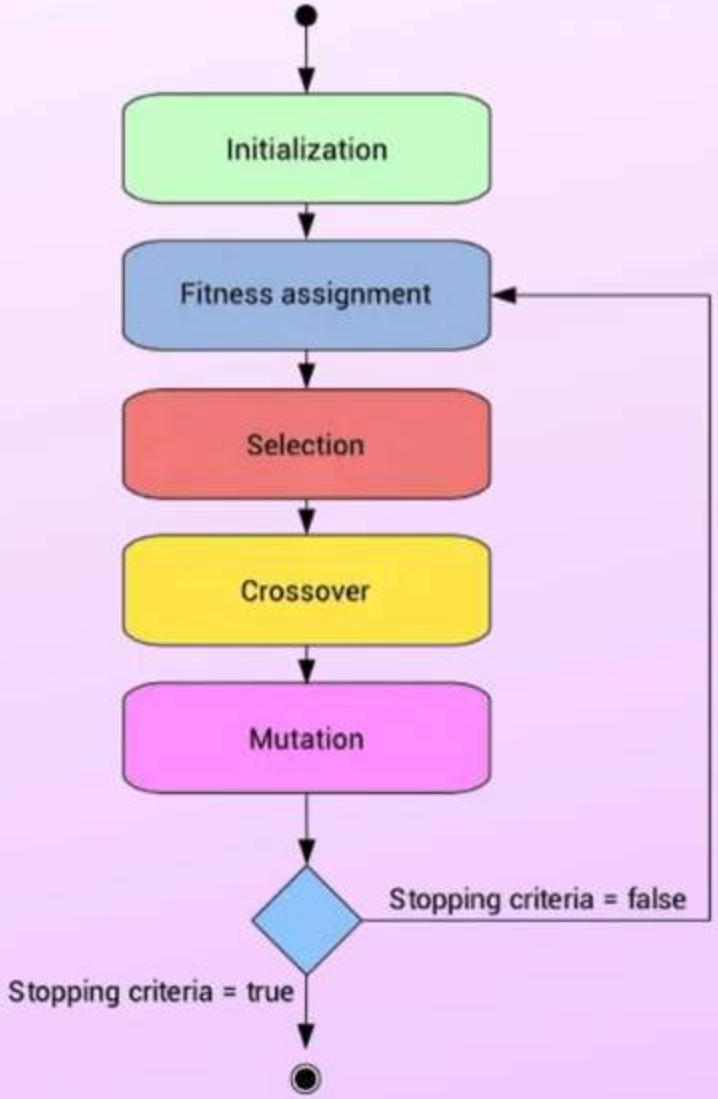
Genetic Algorithms

- Optimal Hyperparameters



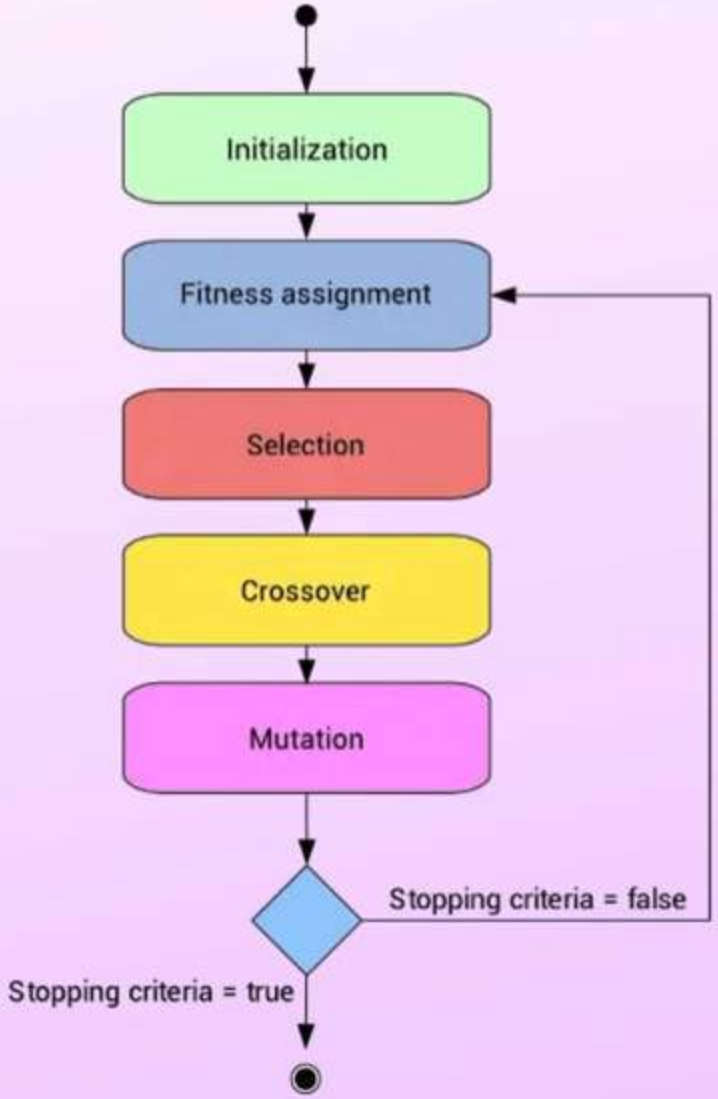
Genetic Algorithms

- Optimal Hyperparameters



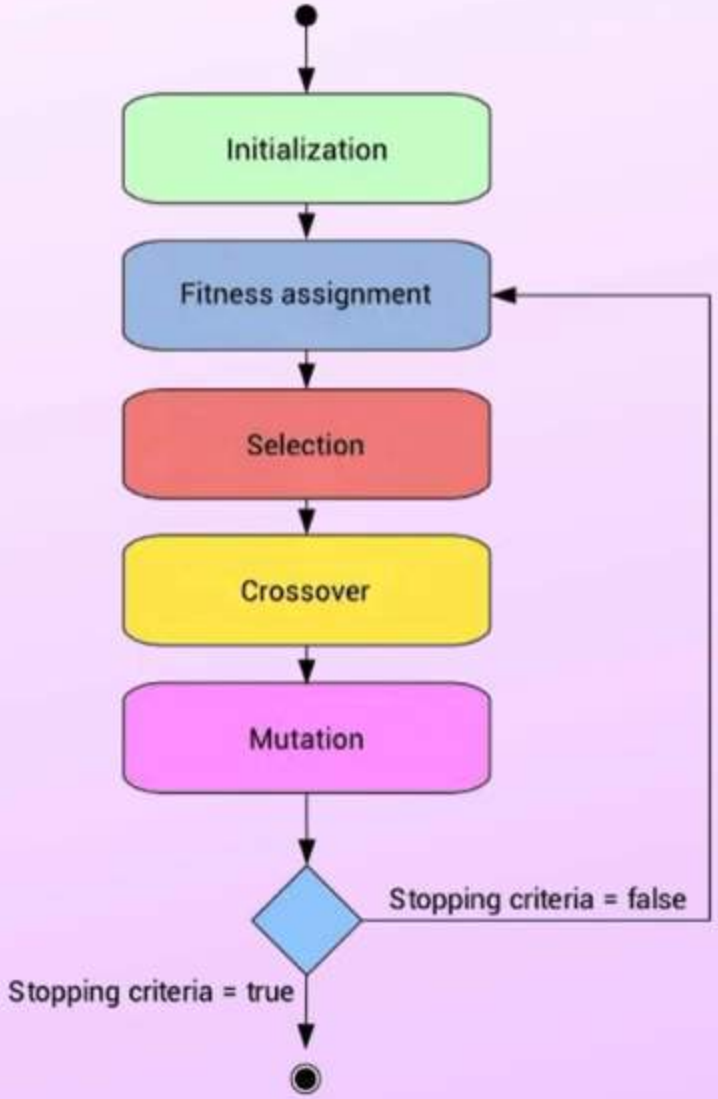
Genetic Algorithms

- Optimal Hyperparameters



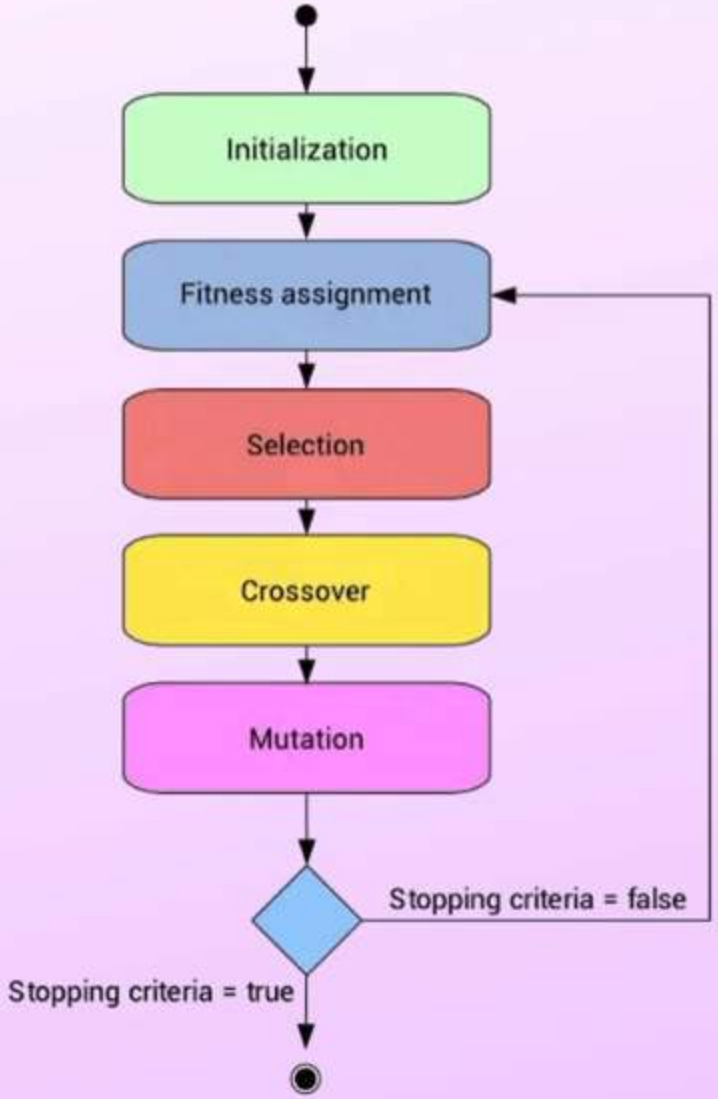
Genetic Algorithms

- Optimal Hyperparameters



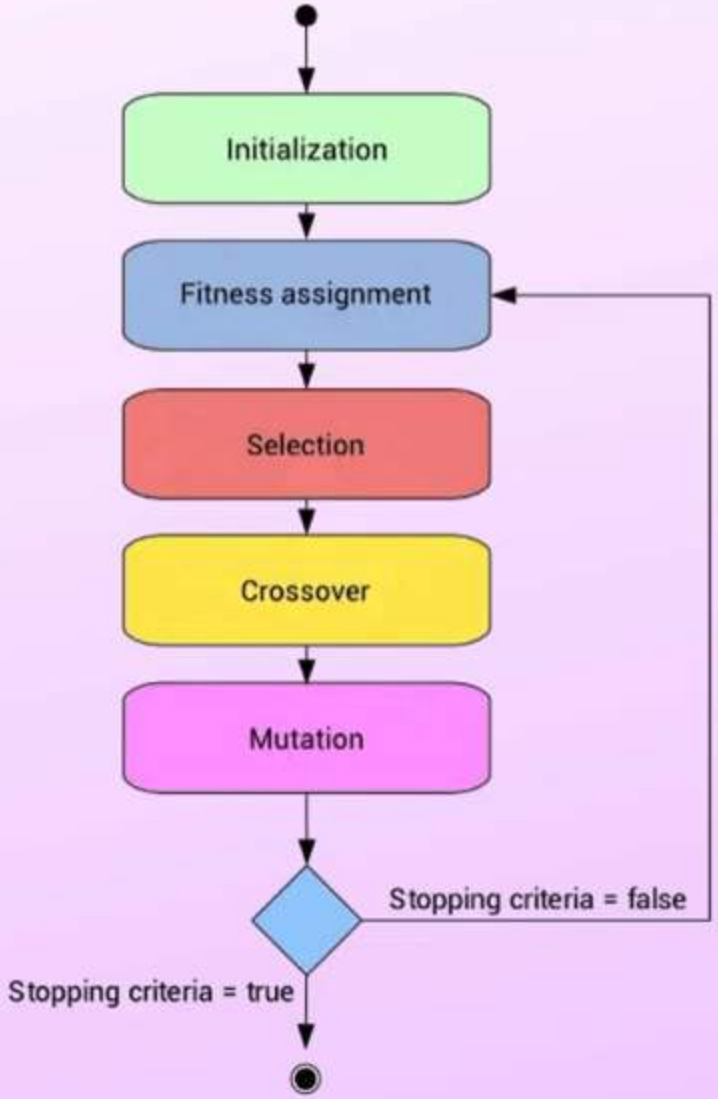
Genetic Algorithms

- Optimal Hyperparameters



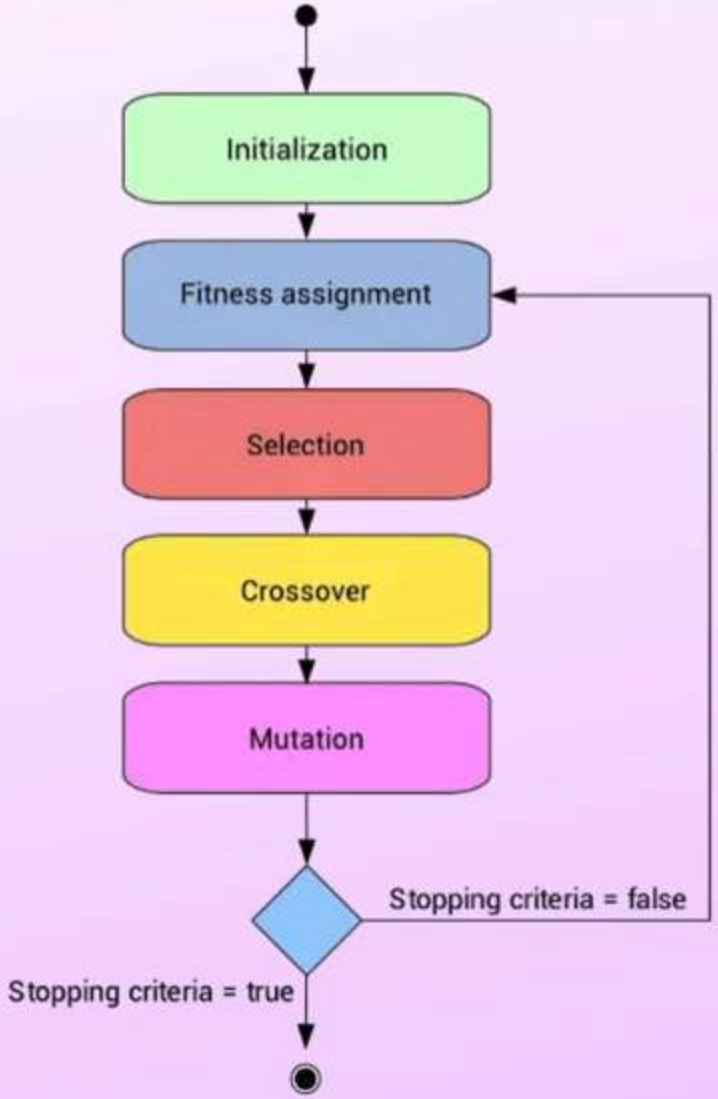
Genetic Algorithms

- Optimal Hyperparameters



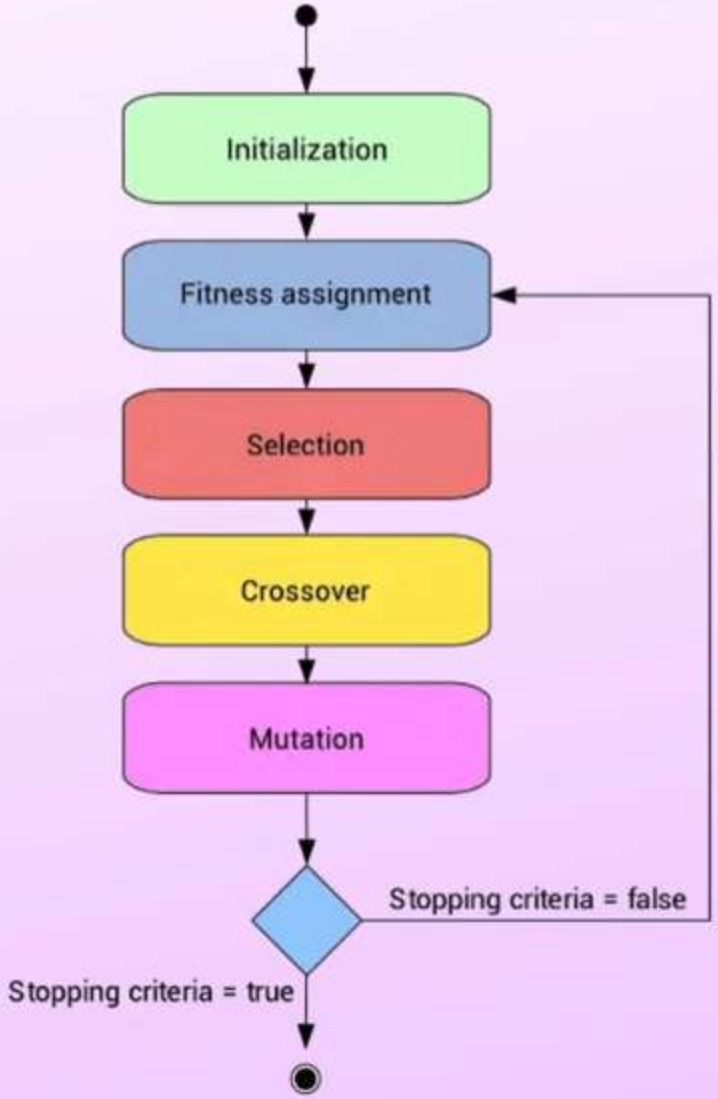
Genetic Algorithms

- Optimal Hyperparameters



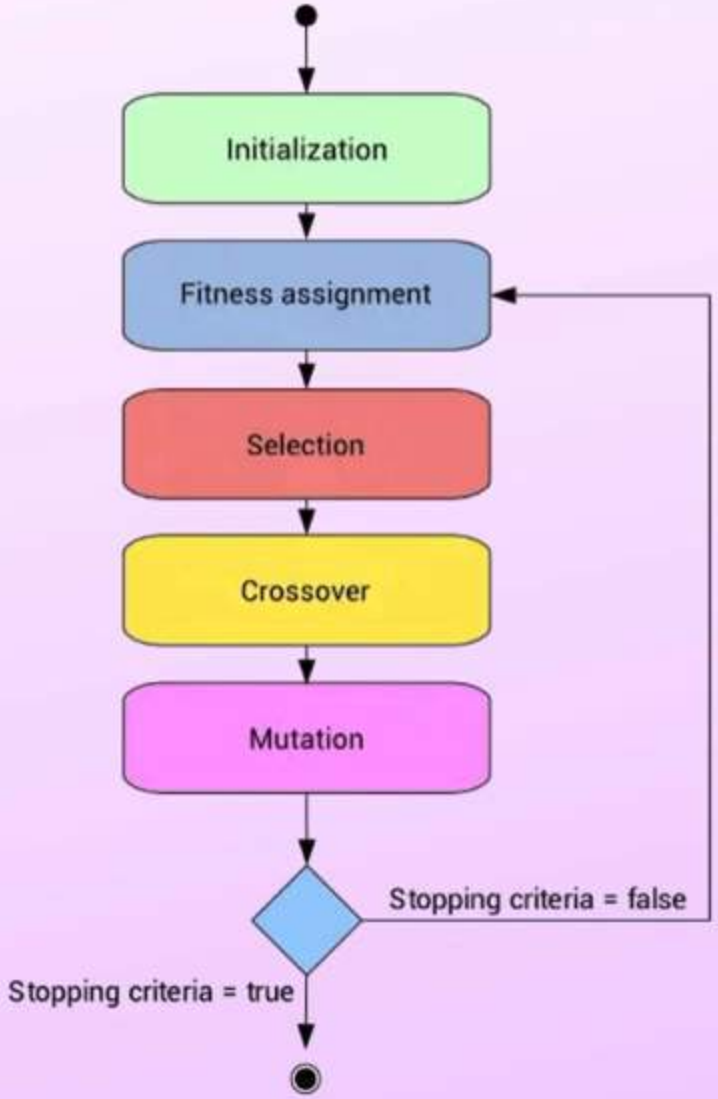
Genetic Algorithms

- Optimal Hyperparameters



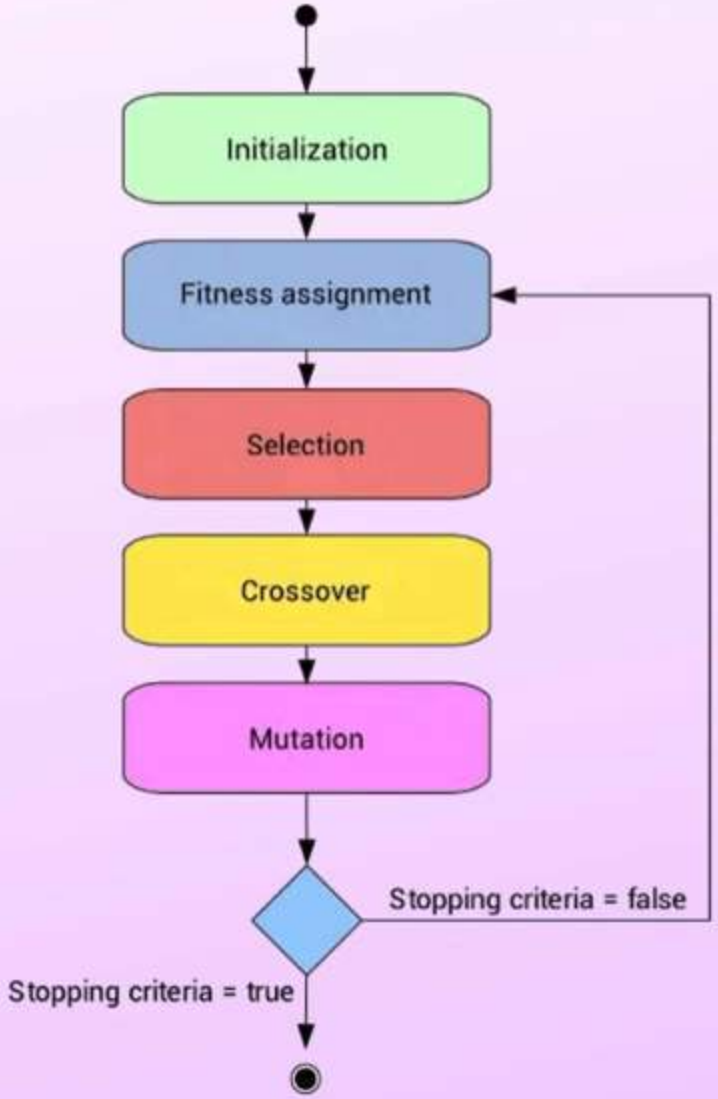
Genetic Algorithms

- Optimal Hyperparameters



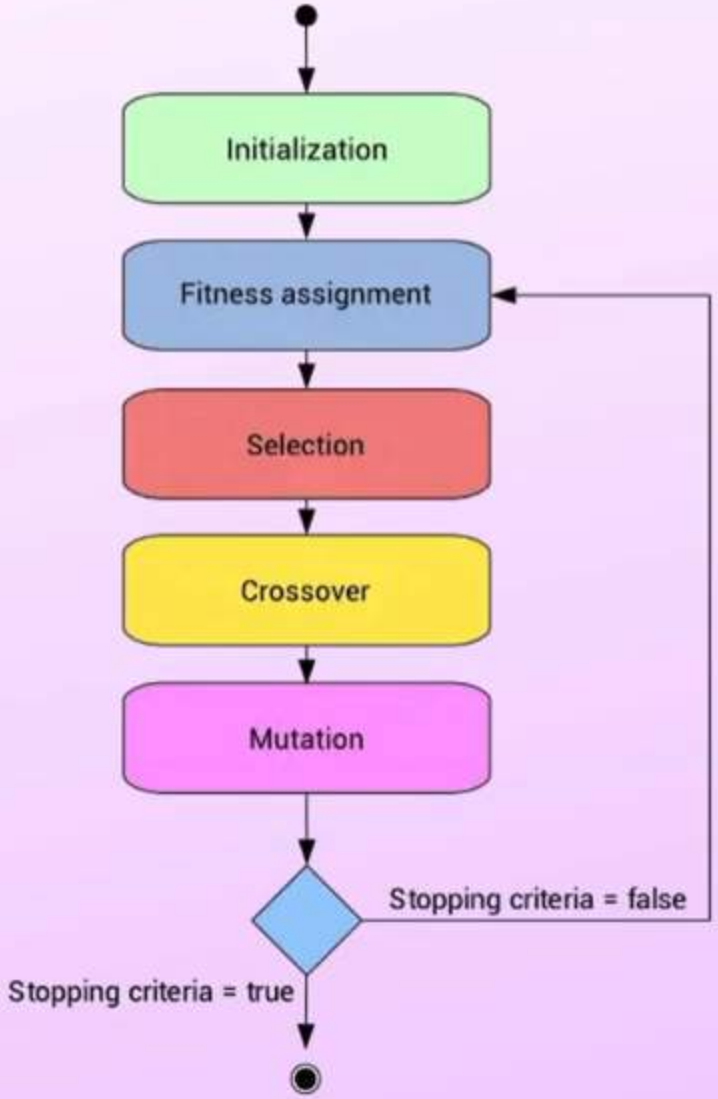
Genetic Algorithms

- Optimal Hyperparameters



Genetic Algorithms

- Optimal Hyperparameters



lr → [0.00001 , 0.1]

Batch

Decay

IOU

lr → [0.00001 , 0.1]

Batch → [4 , 64]

Decay → [0.

IOU

lr → [0.00001 , 0.1]

Batch → [4 , 64]

Decay → [0.0001 , 0.01]

IOU → [0.3 , 0.8]

lr → [0.00001 , 0.1]

Batch → [4 , 64]

Decay → [0.0001 , 0.01]

IOU → [0.3 , 0.8]

lr → [0.00001 , 0.1]

Batch → [4 , 64]

Decay → [0.0001 , 0.01]

IOU → [0.3 , 0.8]

lr → [0.00001 , 0.1]

Batch → [4 , 64]

Decay → [0.0001 , 0.01]

IOU → [0.3 , 0.8]

chromosome → [0.001 ,

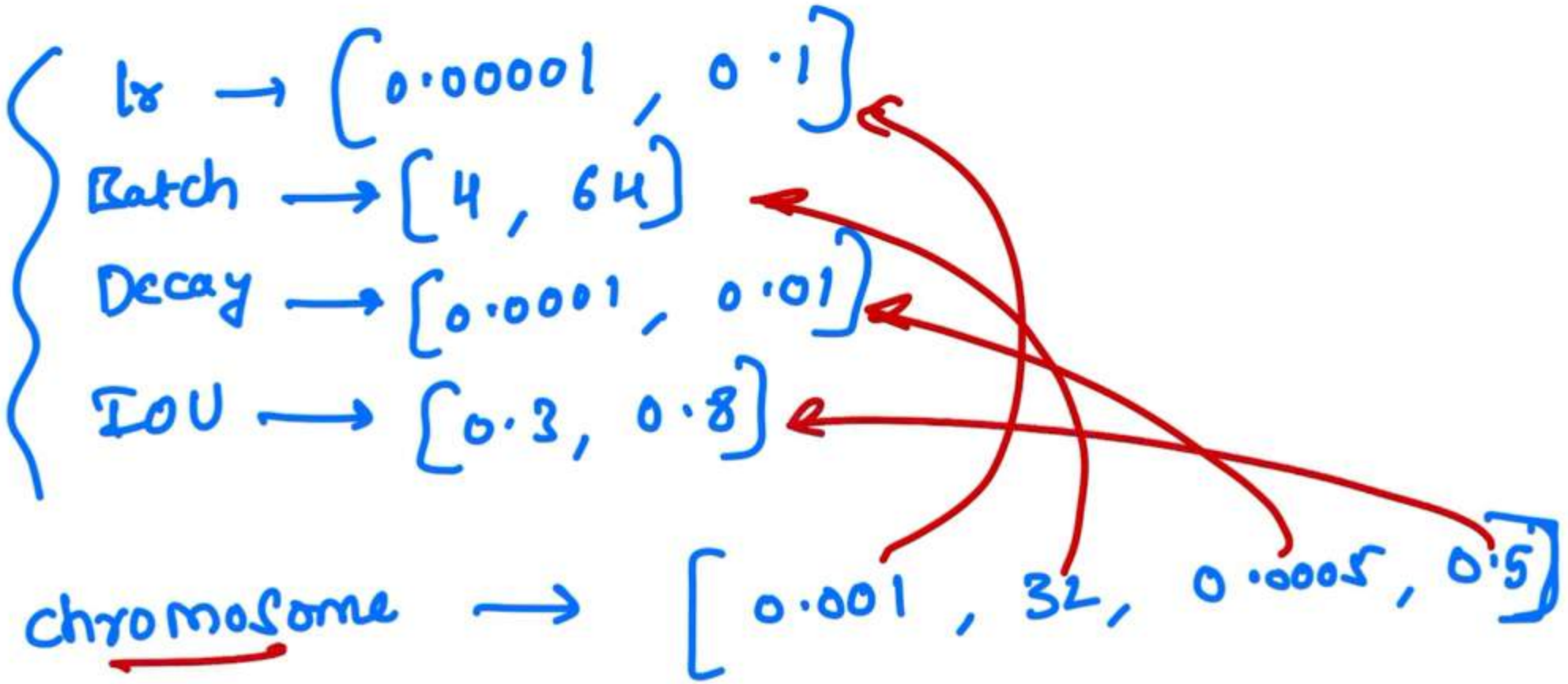
lr → [0.00001 , 0.1]

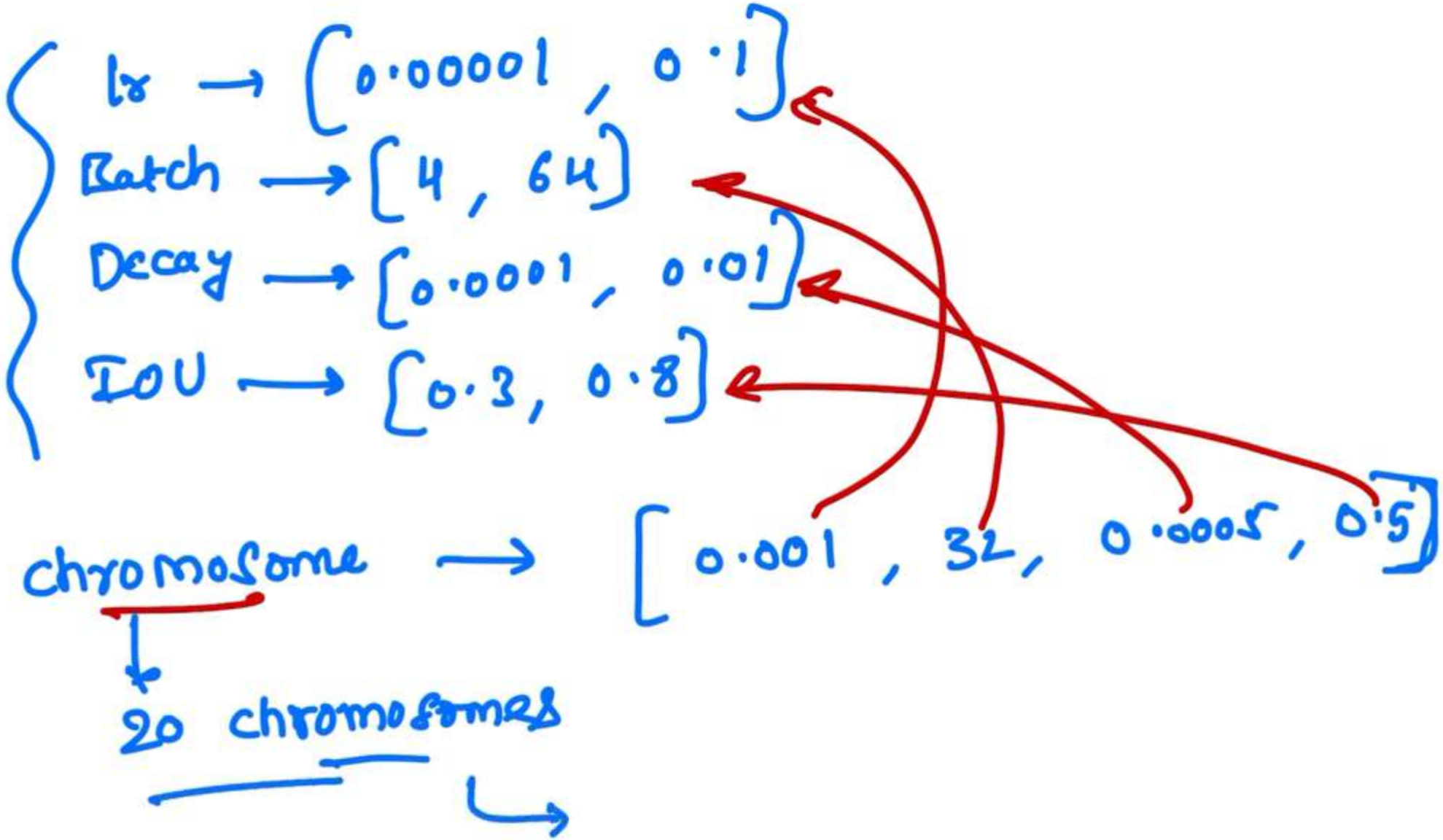
Batch → [4 , 64]

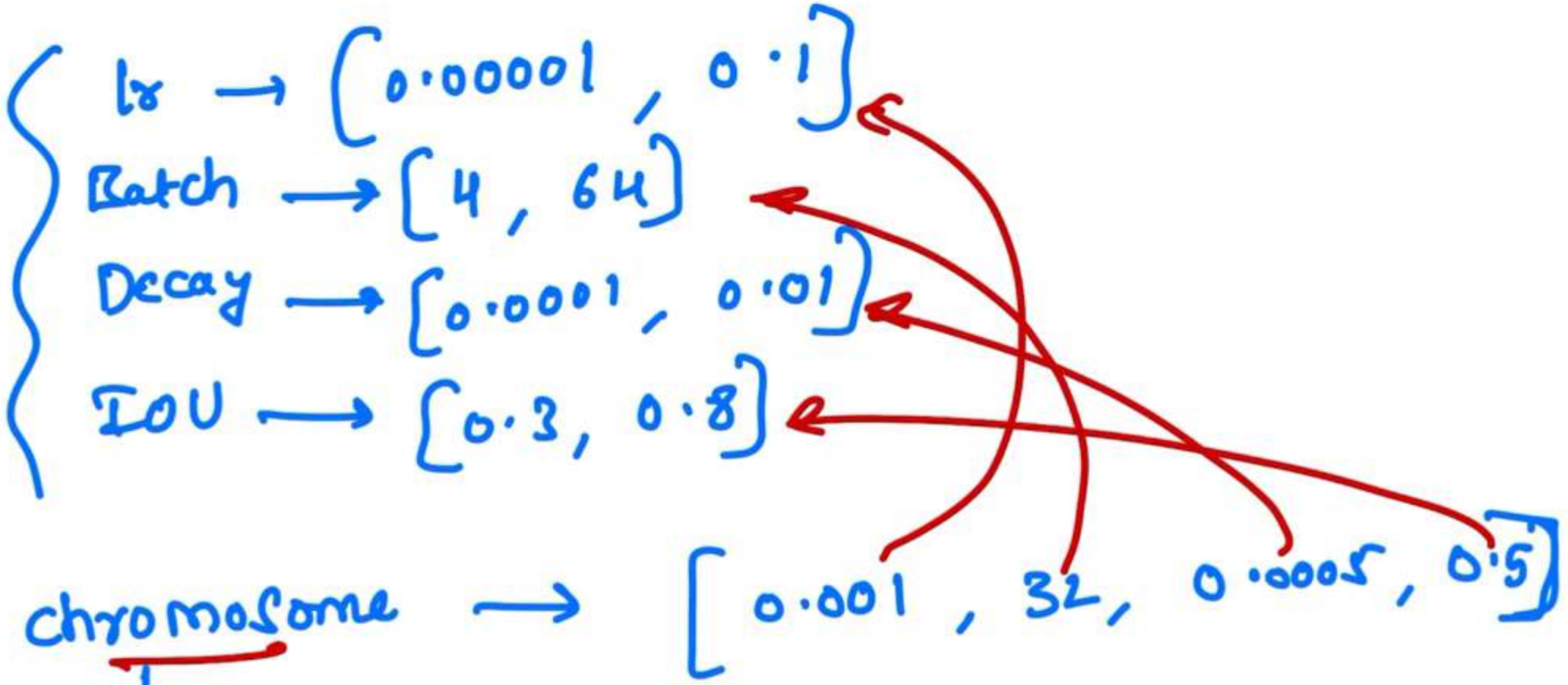
Decay → [0.0001 , 0.01]

IOU → [0.3 , 0.8]

chromosome → [0.001 , 32 , 0.0005 , 0.5]


lr → [0.00001 , 0.1]
Batch → [4 , 64]
Decay → [0.0001 , 0.01]
IOU → [0.3 , 0.8]
chromosome → [0.001 , 32 , 0.0005 , 0.5]

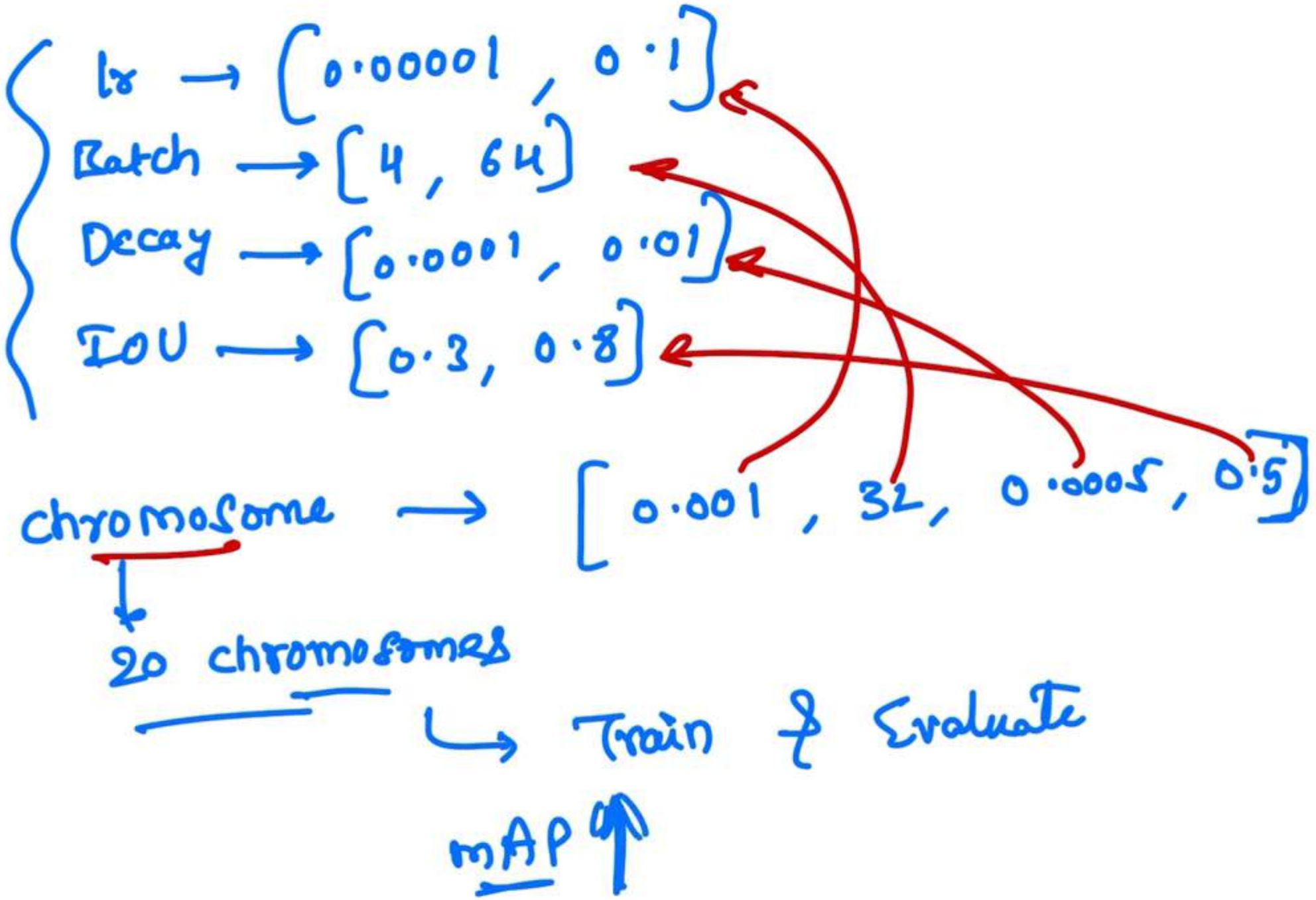

lr → [0.00001 , 0.1]
Batch → [4 , 64]
Decay → [0.0001 , 0.01]
IOU → [0.3 , 0.8]
chromosome → [0.001 , 32 , 0.0005 , 0.5]
20 chromosomes

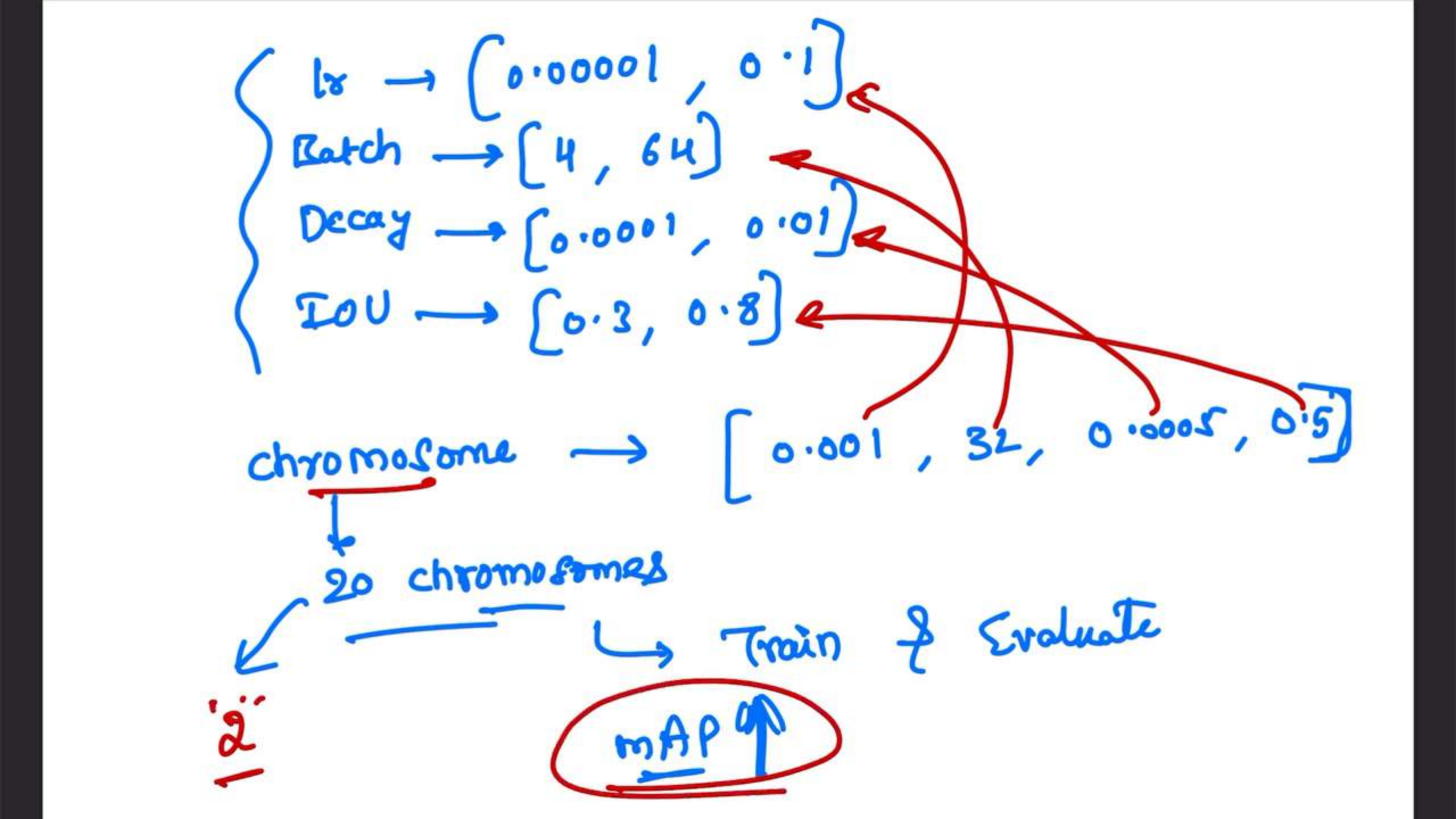


20 chromosomes

↳ Train & Evaluate

mAP


lr → [0.00001 , 0.1]
Batch → [4 , 64]
Decay → [0.0001 , 0.01]
IOU → [0.3 , 0.8]
chromosome → [0.001 , 32 , 0.0005 , 0.5]
20 chromosomes
Train & Evaluate
mAP


lr → [0.00001 , 0.1]
Batch → [4 , 64]
Decay → [0.0001 , 0.01]
IOU → [0.3 , 0.8]
chromosome → [0.001 , 32, 0.0005 , 0.5]
20 chromosomes
Train & Evaluate
mAP
2

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2 { C1 = [0.001, 32, 0.0005, 0.7]
C2 = [0.005, 16, 0.0008, 0.6] }

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2 {
C1 = [0.001, 32, 0.0005, 0.7]
C2 = [0.005, 16, 0.0008, 0.6]
}

||
child

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2

→ C1 = [0.001, 32, 0.0005, 0.7]

→ C2 = [0.005, 16, 0.0008, 0.6]

||

child

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2

→ C1 = [0.001, 32, 0.0005, 0.7]

→ C2 = [0.005, 16, 0.0008, 0.6]

||

child

[0.001, 16, 0.

lr → [0.00001, 0.1]

Batch → [4, 64]

Decay → [0.0001, 0.01]

IOU → [0.3, 0.8]

→ 2

C1 = [0.001, 32, 0.0005, 0.7]

C2 = [0.005, 16, 0.0008, 0.6]

child

→ [0.001, 16, 0.0005, 0.6]

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2 {
p1 = [0.001, 32, 0.0005, 0.7]
p2 = [0.005, 16, 0.0008, 0.6]
}

||

child

c1 → [0.001, 16, 0.0005, 0.6]

|| mute

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2 { p1 = [0.001, 32, 0.0005, 0.7]
p2 = [0.005, 16, 0.0008, 0.6]

child

C1 → [0.001, 16, 0.0005, 0.6]

mutation

o

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2 { p1 = [0.001, 32, 0.0005, 0.7]
→ p2 = [0.005, 16, 0.0008, 0.6]

child

c1 → [0.001, 16, 0.0005, 0.6]

mutation

→ [0.002, 16, 0.0005, 0.6]

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2 { p1 = [0.001, 32, 0.0005, 0.7]
p2 = [0.005, 16, 0.0008, 0.6]

child

c1 → [0.001, 16, 0.0005, 0.6]

mutation

[0.002, 16, 0.0005, 0.6]

10

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2 { p1 = [0.001, 32, 0.0005, 0.7]
p2 = [0.005, 16, 0.0008, 0.6] }

child

c1 → [0.001, 16, 0.0005, 0.6]

mutation

[0.002, 16, 0.0005, 0.6]

map

'10'

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2 { p1 = [0.001, 32, 0.0005, 0.7]
p2 = [0.005, 16, 0.0008, 0.6]

child

100 20

m

c1 → [0.001, 16, 0.0005, 0.6]

mutation

0.002, 16, 0.0005, 0.6

map

'10'

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

→ 2 { p1 = [0.001, 32, 0.0005, 0.7]
p2 = [0.005, 16, 0.0008, 0.6] }

child

100 20

mAP 0.9

C1 → [0.001, 16, 0.0005, 0.6]

mutation

[0.002, 16, 0.0005, 0.6]

map

'10'

lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]

2

p1 = [0.001, 32, 0.0005, 0.7]
p2 = [0.005, 16, 0.0008, 0.6]

child

100 20

mAP 0.9

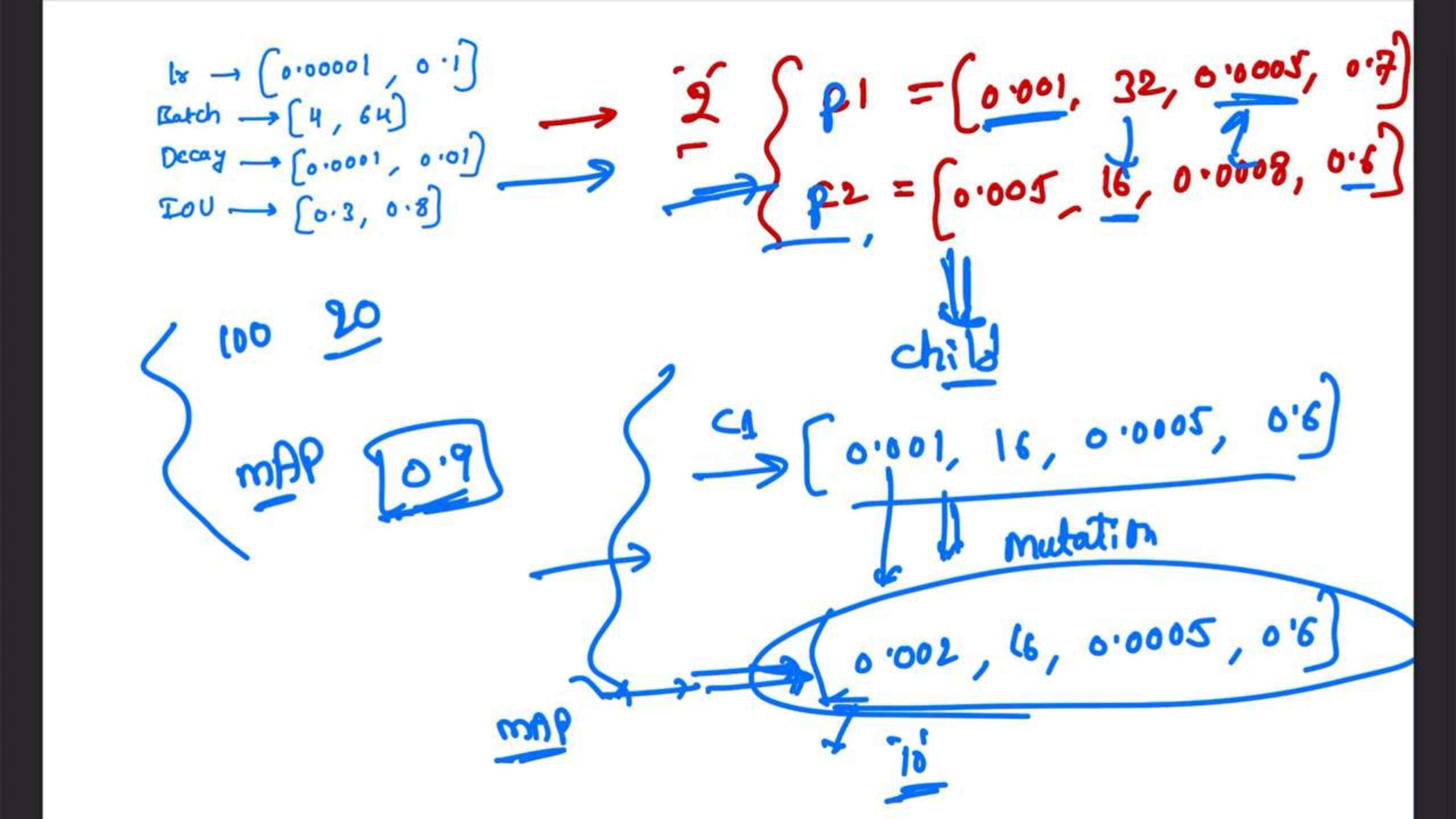
c1 → [0.001, 16, 0.0005, 0.6]

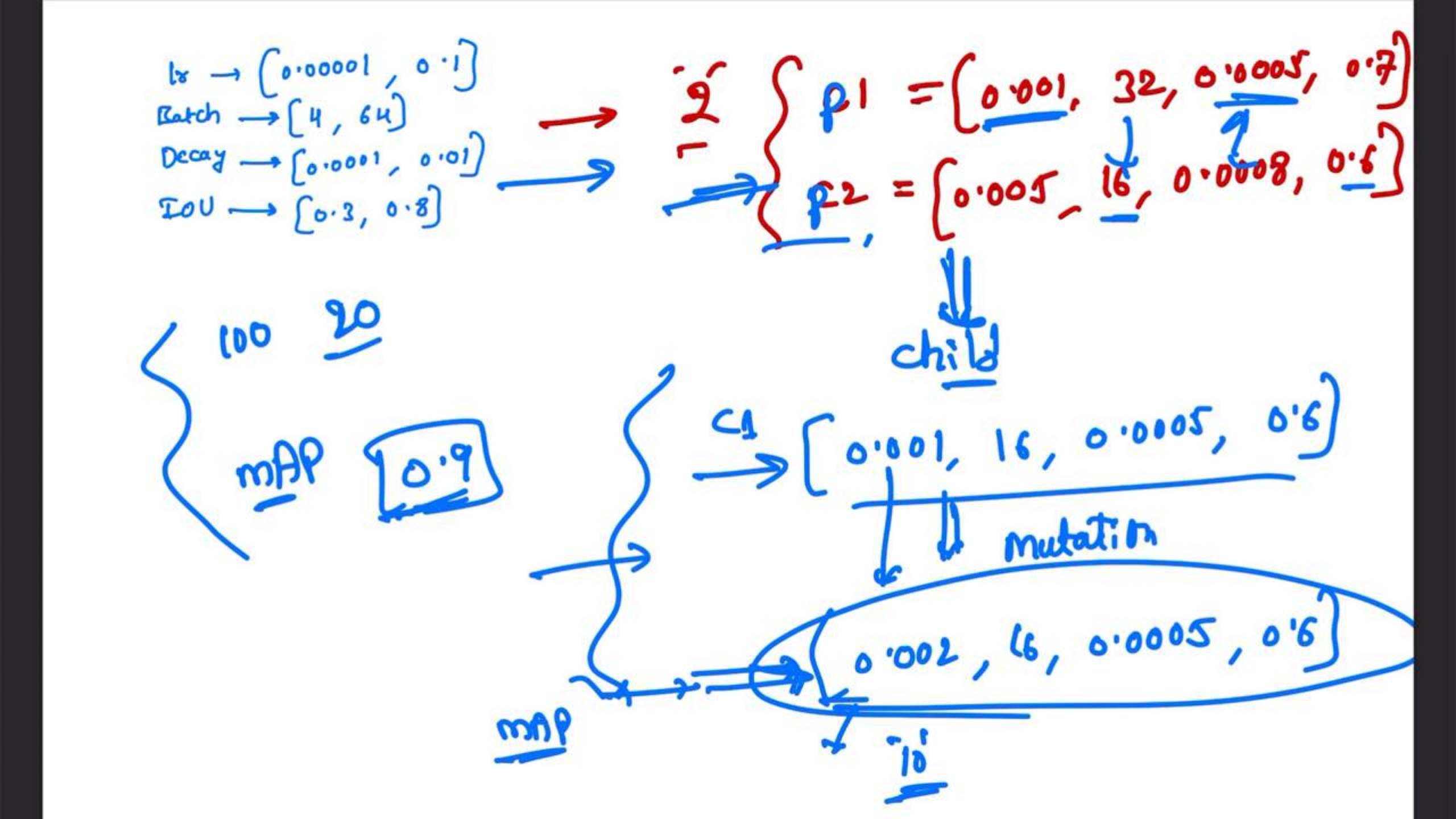
mutation

0.002, 16, 0.0005, 0.6

map

10


lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]
2
p1 = [0.001, 32, 0.0005, 0.7]
p2 = [0.005, 16, 0.0008, 0.6]
child
100
20
mAP
0.9
c1
[0.001, 16, 0.0005, 0.6]
mutation
0.002, 16, 0.0005, 0.6
map
10


lr → [0.00001, 0.1]
Batch → [4, 64]
Decay → [0.0001, 0.01]
IOU → [0.3, 0.8]
2
p1 = [0.001, 32, 0.0005, 0.7]
p2 = [0.005, 16, 0.0008, 0.6]
child
100
20
mAP
0.9
c1
[0.001, 16, 0.0005, 0.6]
mutation
0.002, 16, 0.0005, 0.6
map
10

Genetic Algorithms

- **Define Hyperparameters**: Learning rate, Batch size, Momentum, Weight decay etc.,
- **Encode Hyperparameters**: Chromosome - [learning_rate, batch_size, momentum].
- **Initial Population**: Generate an initial population of random hyperparameter sets.
- **Fitness Function**: Train the YOLOv4 model with each set of hyperparameters and evaluate its performance using a fitness function. This could be based on metrics like mAP on a validation set.
- **Selection**: Select the top-performing hyperparameter sets based on their fitness scores.
- **Crossover**: Combine pairs of selected hyperparameter sets to create new sets.
- **Mutation**: Introduce random changes to some hyperparameter sets to maintain diversity in the population and explore new areas of the hyperparameter space.
- **New Generation**: Create a new generation of hyperparameter sets using the selected, crossover, and mutated sets.
- **Repeat**: Repeat the process for a number of generations or until convergence, i.e., when the improvement in the fitness score plateaus.
- **Best Hyperparameters**: The best-performing hyperparameter set from the final generation is chosen as the optimal set.

Genetic Algorithms

- **Define Hyperparameters**: Learning rate, Batch size, Momentum, Weight decay etc.,
- **Encode Hyperparameters**: Chromosome - [learning_rate, batch_size, momentum].
- **Initial Population**: Generate an initial population of random hyperparameter sets.
- **Fitness Function**: Train the YOLOv4 model with each set of hyperparameters and evaluate its performance using a fitness function. This could be based on metrics like mAP on a validation set.
- **Selection**: Select the top-performing hyperparameter sets based on their fitness scores.
- **Crossover**: Combine pairs of selected hyperparameter sets to create new sets.
- **Mutation**: Introduce random changes to some hyperparameter sets to maintain diversity in the population and explore new areas of the hyperparameter space.
- **New Generation**: Create a new generation of hyperparameter sets using the selected, crossover, and mutated sets.
- **Repeat**: Repeat the process for a number of generations or until convergence, i.e., when the improvement in the fitness score plateaus.
- **Best Hyperparameters**: The best-performing hyperparameter set from the final generation is chosen as the optimal set.

Genetic Algorithms

- **Define Hyperparameters**: Learning rate, Batch size, Momentum, Weight decay etc.,
- **Encode Hyperparameters**: Chromosome - [learning_rate, batch_size, momentum].
- **Initial Population**: Generate an initial population of random hyperparameter sets.
- **Fitness Function**: Train the YOLOv4 model with each set of hyperparameters and evaluate its performance using a fitness function. This could be based on metrics like mAP on a validation set.
- **Selection**: Select the top-performing hyperparameter sets based on their fitness scores.
- **Crossover**: Combine pairs of selected hyperparameter sets to create new sets.
- **Mutation**: Introduce random changes to some hyperparameter sets to maintain diversity in the population and explore new areas of the hyperparameter space.
- **New Generation**: Create a new generation of hyperparameter sets using the selected, crossover, and mutated sets.
- **Repeat**: Repeat the process for a number of generations or until convergence, i.e., when the improvement in the fitness score plateaus.
- **Best Hyperparameters**: The best-performing hyperparameter set from the final generation is chosen as the optimal set.

Genetic Algorithms

- **Define Hyperparameters**: Learning rate, Batch size, Momentum, Weight decay etc.,
- **Encode Hyperparameters**: Chromosome - [learning_rate, batch_size, momentum].
- **Initial Population**: Generate an initial population of random hyperparameter sets.
- **Fitness Function**: Train the YOLOv4 model with each set of hyperparameters and evaluate its performance using a fitness function. This could be based on metrics like mAP on a validation set.
- **Selection**: Select the top-performing hyperparameter sets based on their fitness scores.
- **Crossover**: Combine pairs of selected hyperparameter sets to create new sets.
- **Mutation**: Introduce random changes to some hyperparameter sets to maintain diversity in the population and explore new areas of the hyperparameter space.
- **New Generation**: Create a new generation of hyperparameter sets using the selected, crossover, and mutated sets.
- **Repeat**: Repeat the process for a number of generations or until convergence, i.e., when the improvement in the fitness score plateaus.
- **Best Hyperparameters**: The best-performing hyperparameter set from the final generation is chosen as the optimal set.

Genetic Algorithms

- **Define Hyperparameters**: Learning rate, Batch size, Momentum, Weight decay etc.,
- **Encode Hyperparameters**: Chromosome - [learning_rate, batch_size, momentum].
- **Initial Population**: Generate an initial population of random hyperparameter sets.
- **Fitness Function**: Train the YOLOv4 model with each set of hyperparameters and evaluate its performance using a fitness function. This could be based on metrics like mAP on a validation set.
- **Selection**: Select the top-performing hyperparameter sets based on their fitness scores.
- **Crossover**: Combine pairs of selected hyperparameter sets to create new sets.
- **Mutation**: Introduce random changes to some hyperparameter sets to maintain diversity in the population and explore new areas of the hyperparameter space.
- **New Generation**: Create a new generation of hyperparameter sets using the selected, crossover, and mutated sets.
- **Repeat**: Repeat the process for a number of generations or until convergence, i.e., when the improvement in the fitness score plateaus.
- **Best Hyperparameters**: The best-performing hyperparameter set from the final generation is chosen as the optimal set.

Genetic Algorithms

- **Define Hyperparameters**: Learning rate, Batch size, Momentum, Weight decay etc.,
- **Encode Hyperparameters**: Chromosome - [learning_rate, batch_size, momentum].
- **Initial Population**: Generate an initial population of random hyperparameter sets.
- **Fitness Function**: Train the YOLOv4 model with each set of hyperparameters and evaluate its performance using a fitness function. This could be based on metrics like mAP on a validation set.
- **Selection**: Select the top-performing hyperparameter sets based on their fitness scores.
- **Crossover**: Combine pairs of selected hyperparameter sets to create new sets.
- **Mutation**: Introduce random changes to some hyperparameter sets to maintain diversity in the population and explore new areas of the hyperparameter space.
- **New Generation**: Create a new generation of hyperparameter sets using the selected, crossover, and mutated sets.
- **Repeat**: Repeat the process for a number of generations or until convergence, i.e., when the improvement in the fitness score plateaus.
- **Best Hyperparameters**: The best-performing hyperparameter set from the final generation is chosen as the optimal set.

Genetic Algorithms

- **Define Hyperparameters**: Learning rate, Batch size, Momentum, Weight decay etc.,
- **Encode Hyperparameters**: Chromosome - [learning_rate, batch_size, momentum].
- **Initial Population**: Generate an initial population of random hyperparameter sets.
- **Fitness Function**: Train the YOLOv4 model with each set of hyperparameters and evaluate its performance using a fitness function. This could be based on metrics like mAP on a validation set.
- **Selection**: Select the top-performing hyperparameter sets based on their fitness scores.
- **Crossover**: Combine pairs of selected hyperparameter sets to create new sets.
- **Mutation**: Introduce random changes to some hyperparameter sets to maintain diversity in the population and explore new areas of the hyperparameter space.
- **New Generation**: Create a new generation of hyperparameter sets using the selected, crossover, and mutated sets.
- **Repeat**: Repeat the process for a number of generations or until convergence, i.e., when the improvement in the fitness score plateaus.
- **Best Hyperparameters**: The best-performing hyperparameter set from the final generation is chosen as the optimal set.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

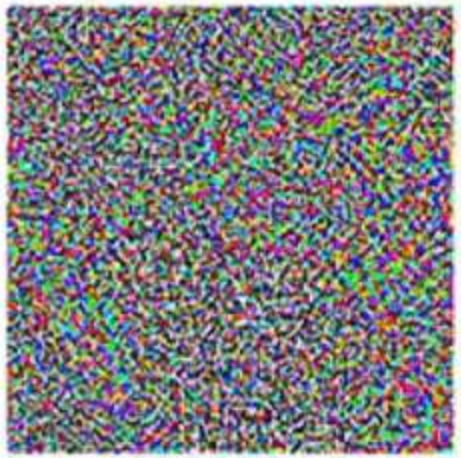
Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Adversarial Attacks

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

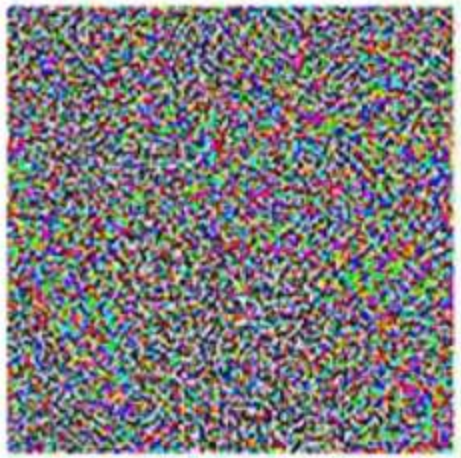
"gibbon"
99.3 % confidence

Adversarial Attacks

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

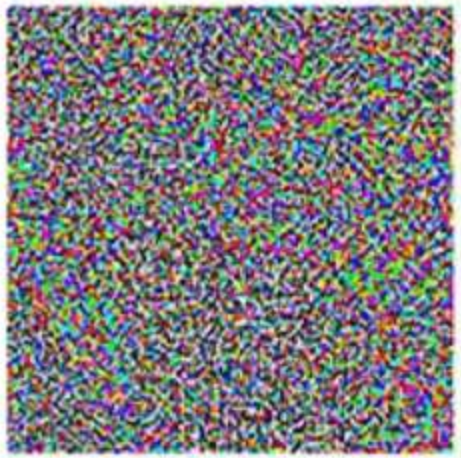
"gibbon"
99.3 % confidence

Adversarial Attacks

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

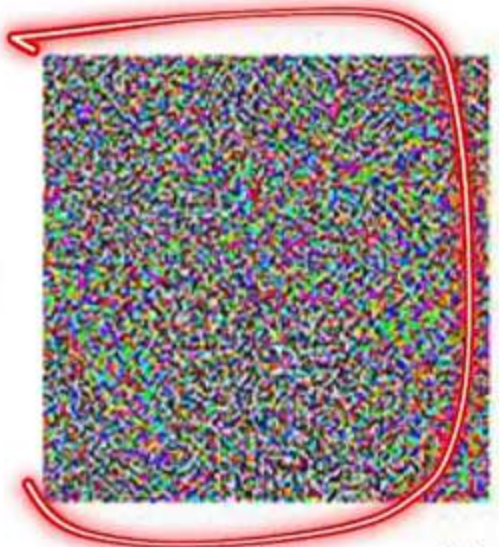
"gibbon"
99.3 % confidence

Adversarial Attacks

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

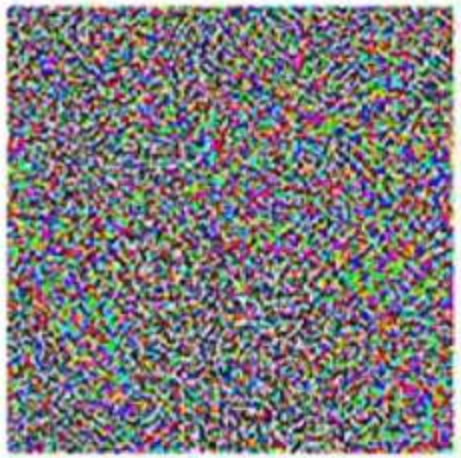
"gibbon"
99.3 % confidence

Adversarial Attacks

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

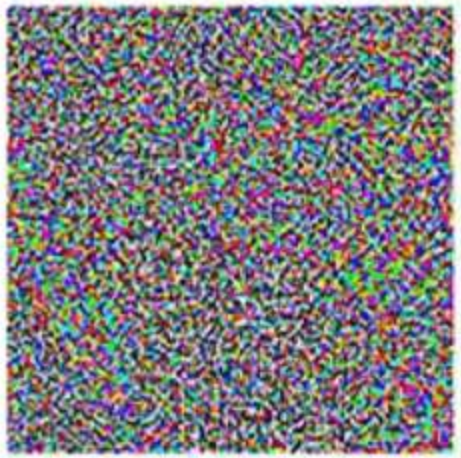
"gibbon"
99.3 % confidence

Adversarial Attacks

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

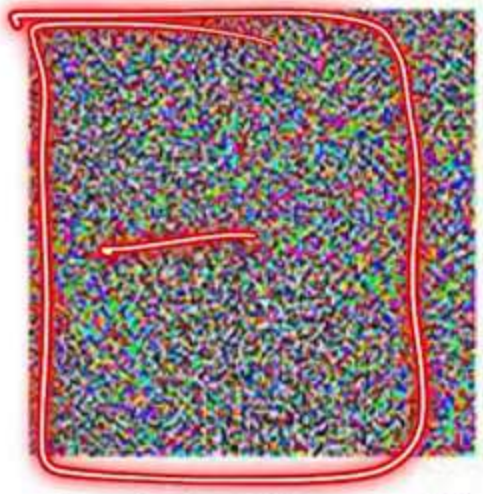
"gibbon"
99.3 % confidence

Adversarial Attacks

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

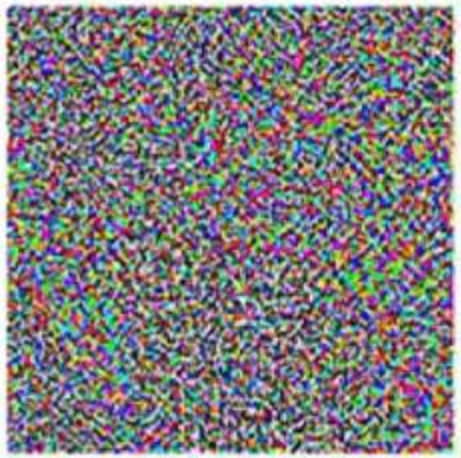
"gibbon"
99.3 % confidence

Adversarial Attacks

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

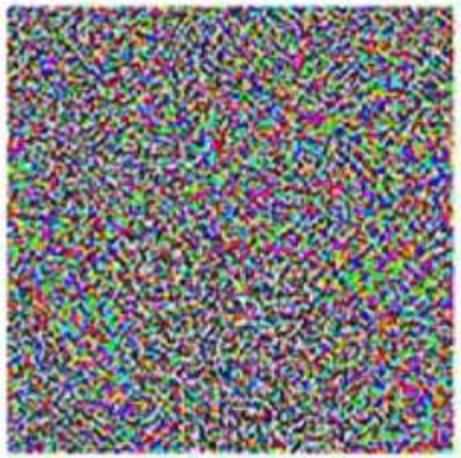
"gibbon"
99.3 % confidence

Adversarial Attacks

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



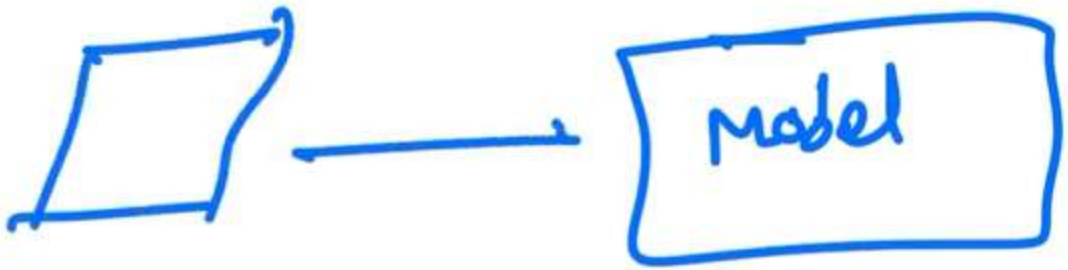
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

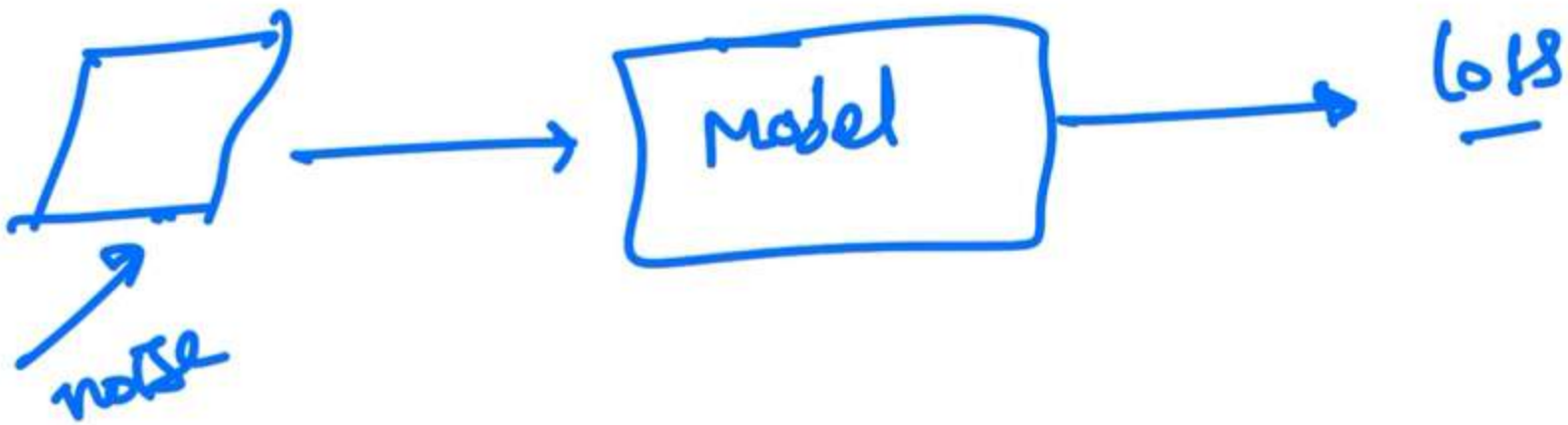
"nematode"
8.2% confidence

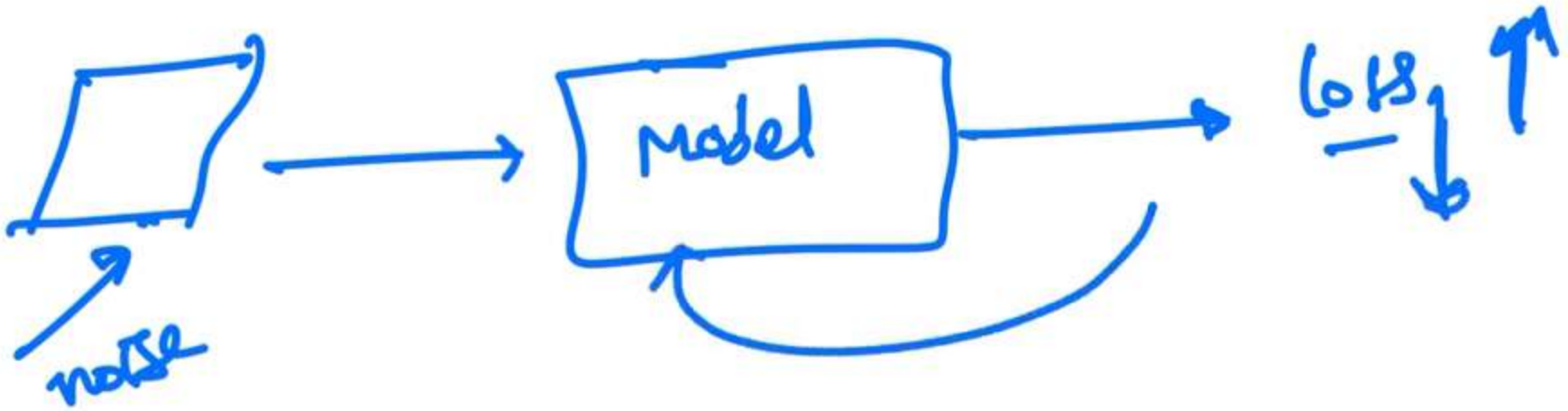
$=$

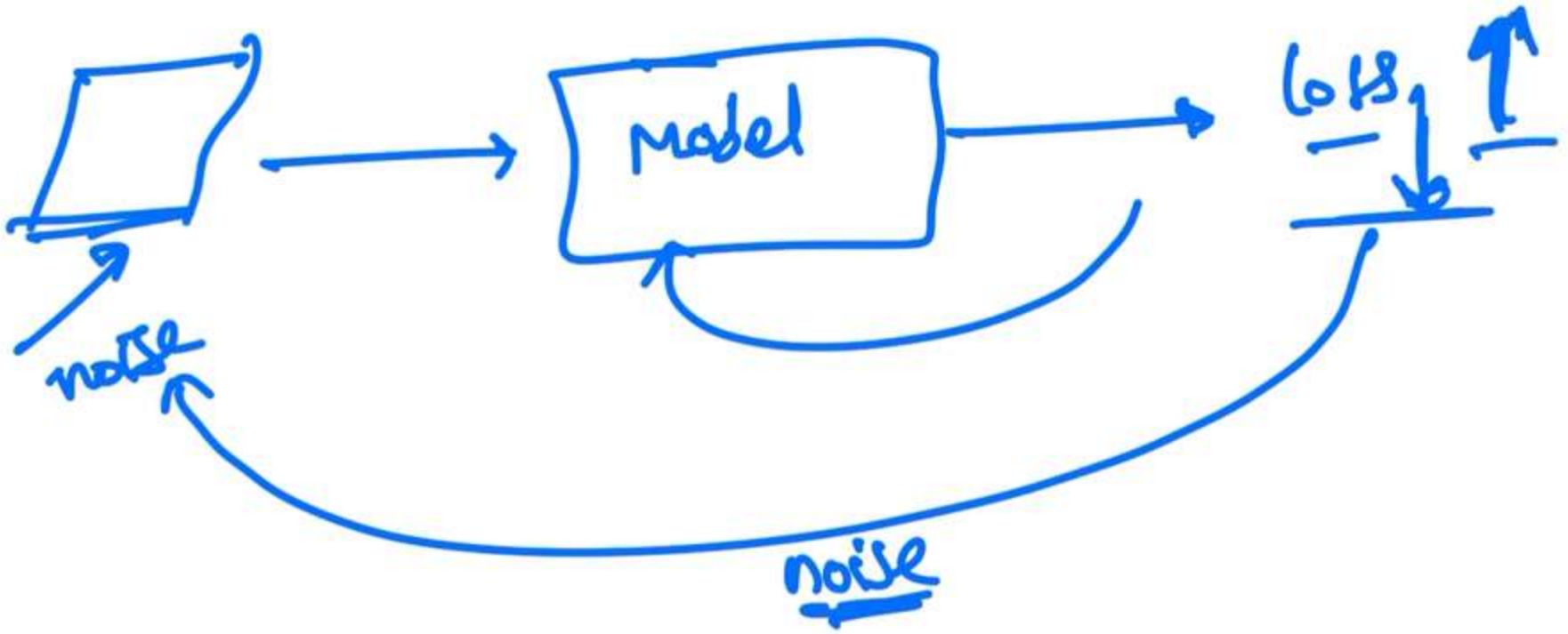
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

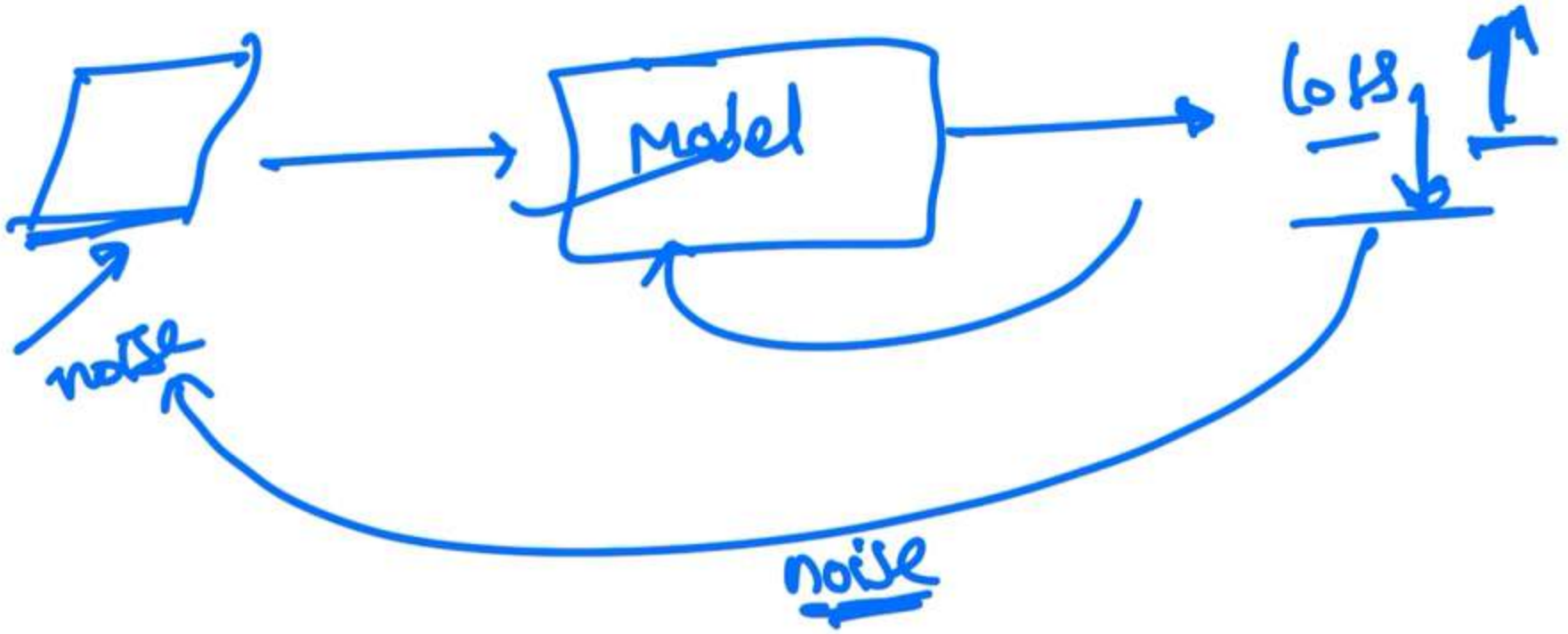
"gibbon"
99.3 % confidence


Model


model
noise


model
loss
noise

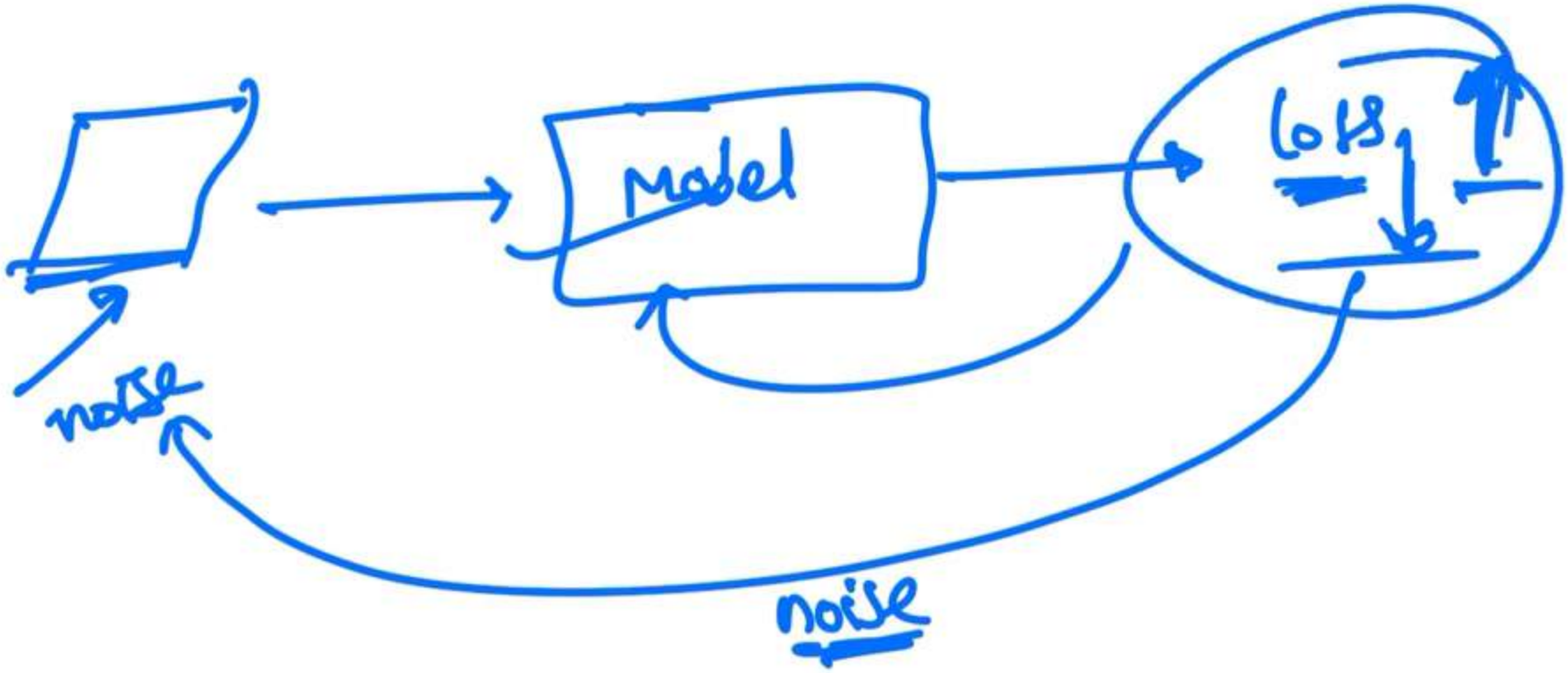

model
loss
noise
noise

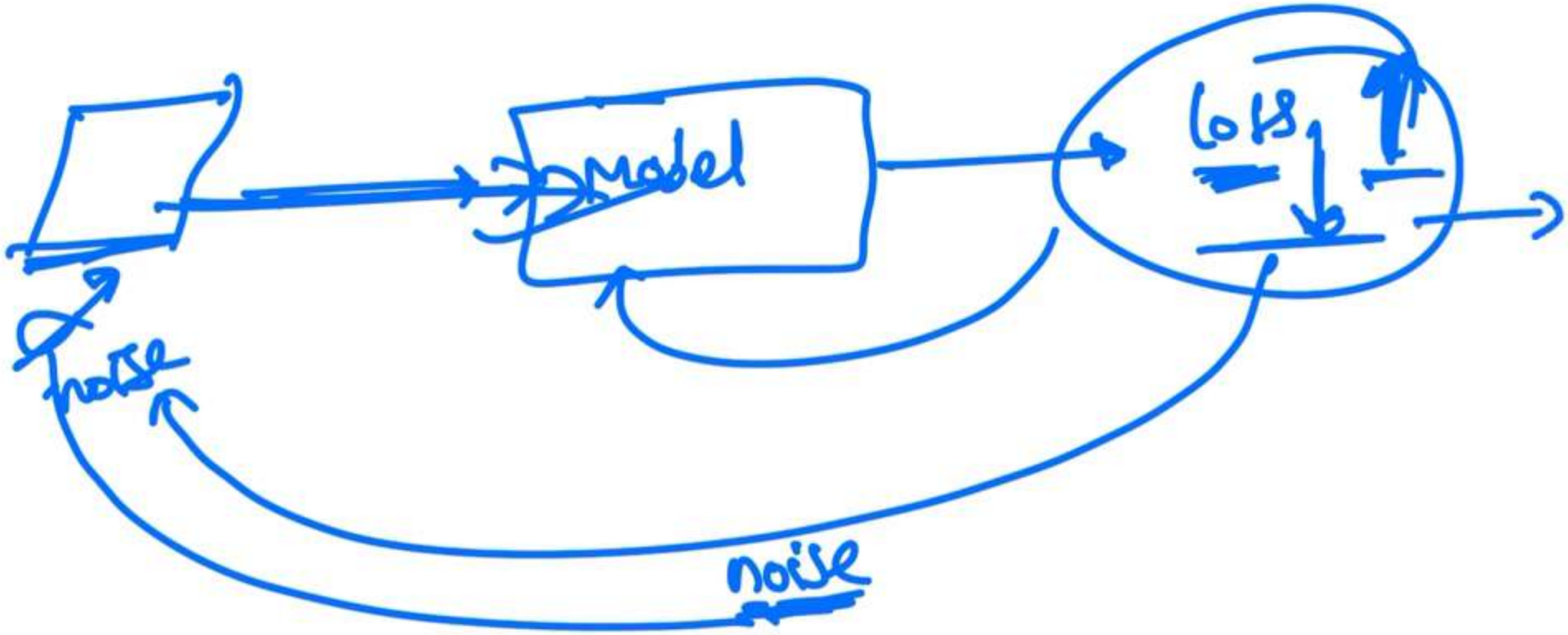

Model
noise
noise

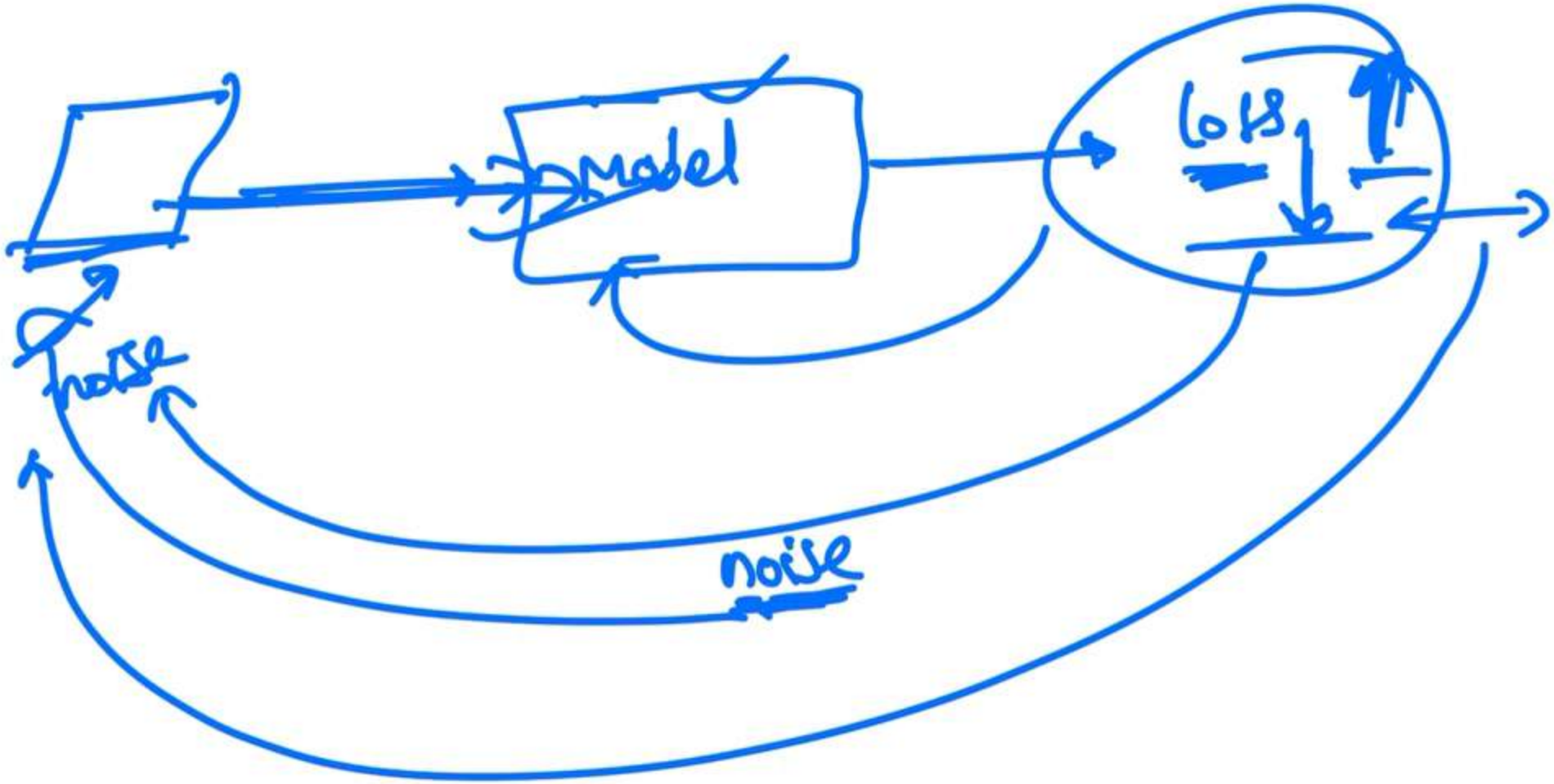
Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.


Model
noise
noise


model
loss
noise
noise


model
loss
noise
noise

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.

Self Adversarial Training

Step 1: The model is initially trained on clean, unmodified images. During this phase, the model's predictions are used to generate adversarial examples by slightly perturbing the input images. This is done by maximizing the model's prediction error while keeping the perturbations within a certain limit to ensure they are subtle and realistic.

Step 2: In the next phase, these adversarial examples are used as inputs for further training. The model is then trained to correctly predict the objects in these perturbed images, effectively learning to be more robust against such adversarial attacks.