```
!pip install ultralytics
```

```
Collecting ultralytics
  Downloading ultralytics-8.3.59-py3-none-any.whl.metadata (35 kB)
Requirement already satisfied: numpy>=1.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.26.4)
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (3.10.0)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.10.0.84)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (11.1.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (6.0.2)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.32.3)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.13.1)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.5.1+cu121)
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.20.1+cu121)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.67.1)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.0.0)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.2.2)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.13.2)
Collecting ultralytics-thop>=2.0.0 (from ultralytics)
  Downloading ultralytics_thop-2.0.13-py3-none-any.whl.metadata (9.4 kB)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.4->ultralytics) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.4->ultralytics) (2024.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (2024.12.14)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.1.5)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch>=1.8.0->ultralytics) (1.3.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=3.3.0->ultralytics) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.8.0->ultralytics) (3.0.2)
Downloading ultralytics-8.3.59-py3-none-any.whl (906 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 906.8/906.8 kB 13.8 MB/s eta 0:00:00
Downloading ultralytics_thop-2.0.13-py3-none-any.whl (26 kB)
Installing collected packages: ultralytics-thop, ultralytics
Successfully installed ultralytics-8.3.59 ultralytics-thop-2.0.13
```

```
from ultralytics import YOLO
```

```
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see https://docs.ultralytics.com/quickstart/#ultralytics-settings.
```

```
# Load a pretrained YOLOv3 model (highly recommended for better results)
model = YOLO('yolov3.pt')
```

```
PRO TIP 💡 Replace 'model=yolov3.pt' with new 'model=yolov3u.pt'.
YOLOv5 'u' models are trained with https://github.com/ultralytics/ultralytics and feature improved performance vs standard YOLOv5 models trained with https://github.com/ultra

Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov3u.pt to 'yolov3u.pt'...
100%|██████████| 198M/198M [00:07<00:00, 29.7MB/s]
```

```python
def boxes(results, image):
    """
    Draws bounding boxes and labels on an image based on YOLO predictions.

    Args:
        results: YOLO model predictions containing bounding box coordinates, classes, and confidence scores.
        image: The image (NumPy array) on which to draw the bounding boxes.

    Returns:
        The image with bounding boxes and labels drawn on it.
    """
    # Create a copy of the image to avoid modifying the original
    annotated_image = image.copy()

    # Extract predictions
    for result in results:
        for box in result.boxes:
            # Get bounding box coordinates
            x1, y1, x2, y2 = box.xyxy[0].int().numpy()  # Convert tensor to integers
            label = result.names[int(box.cls[0])]      # Get class label
            confidence = box.conf[0].item()           # Get confidence score

            # Draw bounding box and label on the image
            cv2.rectangle(annotated_image, (x1, y1), (x2, y2), (255, 0, 0), 2)  # Bounding box in blue
            cv2.putText(annotated_image, f"{label} {confidence:.2f}", (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)  # Label in blue
    return annotated_image



# Function to apply Wiener deblurring (or any other deblurring technique)
def wiener_deblur(image):
    deblurred_image = cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)
    return deblurred_image

# Function to apply Unsharp Masking (in RGB space)
def unsharp_mask(image, sigma=1.0, strength=1.5):
    blurred = cv2.GaussianBlur(image, (5, 5), sigma)
    sharpened = cv2.addWeighted(image, 1.0 + strength, blurred, -strength, 0)
    return sharpened

# Function to apply CLAHE (Contrast Limited Adaptive Histogram Equalization) in RGB space
def clahe(image):
    lab = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
    cl = clahe.apply(l)
    limg = cv2.merge((cl, a, b))
```

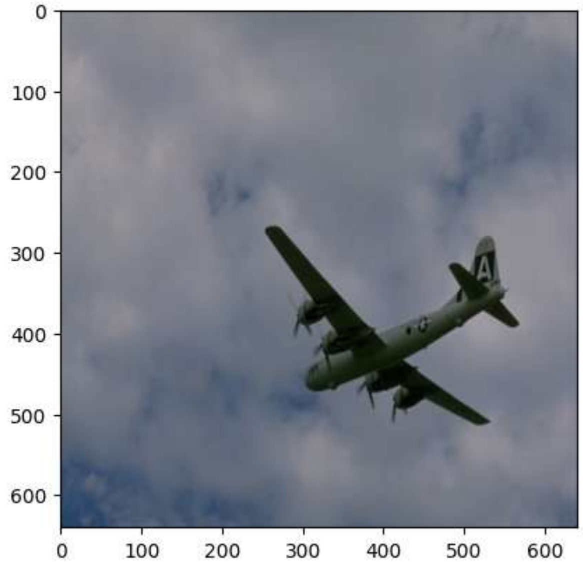
```python
    return cv2.cvtColor(limg, cv2.COLOR_LAB2RGB)

# Function to apply edge enhancement filter in RGB space
def edge_enhance(image):
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    magnitude = cv2.magnitude(sobel_x, sobel_y)
    magnitude = np.uint8(np.clip(magnitude, 0, 255))
    return cv2.cvtColor(magnitude, cv2.COLOR_GRAY2RGB)

# Function to denoise the image
def denoise(image):
    return cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)

# Guided filter function for deblurring (in RGB space)
def guided_filter(image, radius=5, epsilon=0.2):
    image = np.float32(image)
    guide = image
    result = cv2.ximgproc.guidedFilter(guide, image, radius, epsilon)
    return np.uint8(np.clip(result, 0, 255))

# Bilateral sharpening function (in RGB space)
def bilateral_sharpening(image, d=9, sigma_color=75, sigma_space=75):
    smoothed = cv2.bilateralFilter(image, d, sigma_color, sigma_space)
    sharpened = cv2.addWeighted(image, 1.5, smoothed, -0.5, 0)
    return sharpened

def laplacian_sharpen(image):
    # Convert the image to grayscale to get the Laplacian
    gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    # Apply the Laplacian operator to detect edges
    laplacian = cv2.Laplacian(gray_image, cv2.CV_64F)

    # Convert Laplacian to uint8 (8-bit unsigned integer) for proper display
    laplacian = cv2.convertScaleAbs(laplacian)

    # Enhance the image by adding the Laplacian to the original image
    sharpened = cv2.addWeighted(image, 1.2, cv2.cvtColor(laplacian, cv2.COLOR_GRAY2RGB), 1.2, -20)

    return sharpened

def sobel_edge_enhance(image):
    # Convert the image to grayscale for edge detection
    gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    # Apply Sobel filter in both x and y directions to detect edges
    sobel_x = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(gray_image, cv2.CV_64F, 0, 1, ksize=3)

    # Calculate the gradient magnitude
    magnitude = cv2.magnitude(sobel_x, sobel_y)

    # Normalize the magnitude to 8-bit
    magnitude = np.uint8(np.clip(magnitude, 0, 255))
```

```python
    # Convert the magnitude back to RGB
    edge_image = cv2.cvtColor(magnitude, cv2.COLOR_GRAY2RGB)

    # Enhance edges: We can add the edge map back to the original image
    enhanced_image = cv2.addWeighted(image, 1, edge_image, 0.3, 0)

    return enhanced_image

def laplacian_of_gaussian(image, kernel_size=1, sigma=1.0, k=1.0):
    """
    Applies Laplacian of Gaussian (LoG) filtering to enhance edges in an RGB image.

    Args:
        image: The input RGB image (numpy array).
        kernel_size: The size of the LoG kernel (default: 5).
        sigma: Standard deviation for the Gaussian kernel (default: 1.0).
        k: Scaling factor for the Laplacian (default: 1.0).

    Returns:
        An enhanced RGB image (numpy array).
    """

    # Apply Gaussian blur to the image
    blurred_image = cv2.GaussianBlur(image, (kernel_size, kernel_size), sigma)

    # Calculate the Laplacian (second derivative)
    laplacian = cv2.Laplacian(blurred_image, cv2.CV_64F, ksize=kernel_size)

    # Convert Laplacian to uint8 for display and blending
    laplacian = cv2.convertScaleAbs(laplacian)

    # Enhance the image by adding the Laplacian to the original image
    enhanced_image = cv2.addWeighted(image, 1.0, laplacian, 0.5, 0)

    return enhanced_image

def gaussian_filter(image, kernel_size, sigma):
  """
  Applies a Gaussian filter to an image.

  Args:
      image: The input image (numpy array).
      kernel_size: The size of the Gaussian kernel (must be odd).
      sigma: Standard deviation of the Gaussian distribution.

  Returns:
      The filtered image (numpy array).
  """

  # Apply Gaussian blur using OpenCV
  filtered_image = cv2.GaussianBlur(image, (kernel_size, kernel_size), sigma)
  return filtered_image

# Function to apply all enhancements to the image (for blurred images)
```

```
def enhance_image(image):
    # Clip the values and convert to uint8
    image = np.clip(image, 0, 255)
    image = np.uint8(image)
    image = wiener_deblur(image)
    image = unsharp_mask(image)
    image = bilateral_sharpening(image)
    #image = bilateral_sharpening(image)
    image = guided_filter(image)
    #image = bilateral_sharpening(image)
    image = laplacian_of_gaussian(image, 3, 1.5)
    image = unsharp_mask(image)
    #image = gaussian_filter(image, 3, 1)

    return image



import cv2
import os
import matplotlib.pyplot as plt

images = os.listdir('/content/Learning-to-See-in-the-Dark/CBSD68-dataset/CBSD68/noisy50')
img = []
for i, image in enumerate(images):
  img.append(cv2.resize(cv2.imread('/content/Learning-to-See-in-the-Dark/CBSD68-dataset/CBSD68/noisy50/' + image), (640, 640)))


images1 = os.listdir('/content/Learning-to-See-in-the-Dark/CBSD68-dataset/CBSD68/original')
img1 = []
for i, image in enumerate(images1):
  img1.append(cv2.resize(cv2.imread('/content/Learning-to-See-in-the-Dark/CBSD68-dataset/CBSD68/original/' + image), (640, 640)))


def addNoiseandBlur(image):
  noise = np.random.normal(0, 20, image.shape).astype(np.uint8)
  noisy_image = cv2.add(image, noise)
  return noisy_image

def addSaltPepperNoiseAndBrighten(image, salt_prob=0.02, pepper_prob=0.03, brightness_factor=1.2):
    # Get the image dimensions
    row, col, ch = image.shape

    # Add salt and pepper noise
    noisy_image = image.copy()

    # Salt noise (set some pixels to white)
    salt = np.random.rand(row, col, ch) < salt_prob
    noisy_image[salt] = 255

    # Pepper noise (set some pixels to black)
    pepper = np.random.rand(row, col, ch) < pepper_prob
    noisy_image[pepper] = 0

    # Brighten the image by scaling the pixel values
    brightened_image = np.clip(noisy_image * brightness_factor, 0, 255).astype(np.uint8)
```

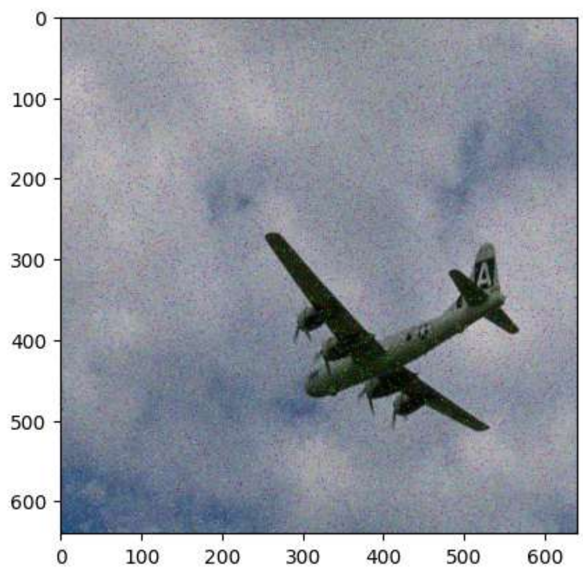
```python
    return brightened_image


def reverseSaltPepperNoiseAndBrightness(noisy_brightened_image, salt_prob=0.05, pepper_prob=0.05, brightness_factor=1.2):
    # Step 1: Denoise the image using Median Filtering to remove salt and pepper noise
    denoised_image = cv2.medianBlur(noisy_brightened_image, 5)  # 3x3 kernel

    # Step 2: Compensate for brightness adjustment by darkening the image (inverse of brightness factor)
    # We divide by the brightness factor and clip values to stay within the valid range [0, 255]
    restored_image = np.clip(denoised_image / brightness_factor, 0, 255).astype(np.uint8)

    return restored_image


def contrastEnhance(image):
  (b, g, r) = cv2.split(image)

  # Apply histogram equalization on each channel
  r = cv2.equalizeHist(r)
  g = cv2.equalizeHist(g)
  b = cv2.equalizeHist(b)

  # Merge the channels back together
  image = cv2.merge([b, g, r])

  return image


import os
import cv2
import matplotlib.pyplot as plt
import numpy as np



plane = cv2.imread('/content/plane.jpg')
plane = cv2.resize(plane, (640, 640))
plane = cv2.cvtColor(plane, cv2.COLOR_BGR2RGB)
plt.imshow(plane)
```

```
<matplotlib.image.AxesImage at 0x797b4ae28400>
```




```
plane = addSaltPepperNoiseAndBrighten(plane)
plt.imshow(plane)
```

```
<matplotlib.image.AxesImage at 0x797b483feec0>
```

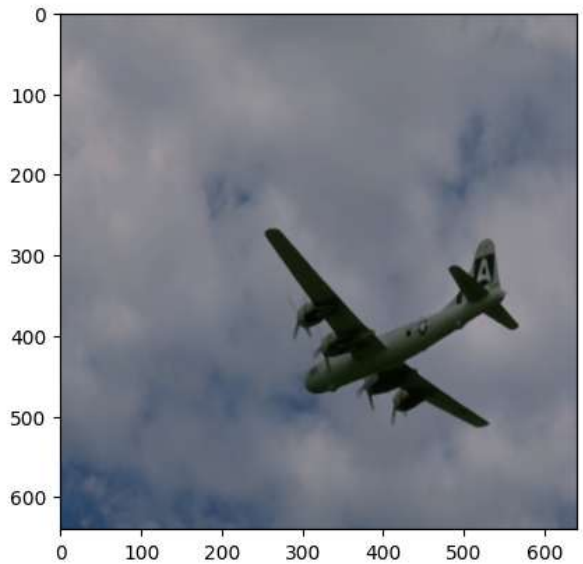



```
p = model.predict(source=plane)
```

```
0: 640x640 (no detections), 5785.4ms
Speed: 4.8ms preprocess, 5785.4ms inference, 1.6ms postprocess per image at shape (1, 3, 640, 640)
```

```python
plane = reverseSaltPepperNoiseAndBrightness(plane)
plt.imshow(plane)
```

```
<matplotlib.image.AxesImage at 0x797b4b1cfd60>
```



```python
p = model.predict(source=plane)
```

```
0: 640x640 1 airplane, 6473.1ms
Speed: 6.7ms preprocess, 6473.1ms inference, 1.7ms postprocess per image at shape (1, 3, 640, 640)
```

```python
# Create the 'noisy' folder if it doesn't exist
if not os.path.exists('/content/noisy'):
    os.makedirs('/content/noisy')
```

```python
files = os.listdir('/content/')
jpegs = [file for file in files if file.endswith('.jpg')]
noisy = []
for file in jpegs:
  noisy.append(addSaltPepperNoiseAndBrighten(cv2.cvtColor(cv2.resize(cv2.imread('/content/' + file), (640, 640)), cv2.COLOR_BGR2RGB)))
```

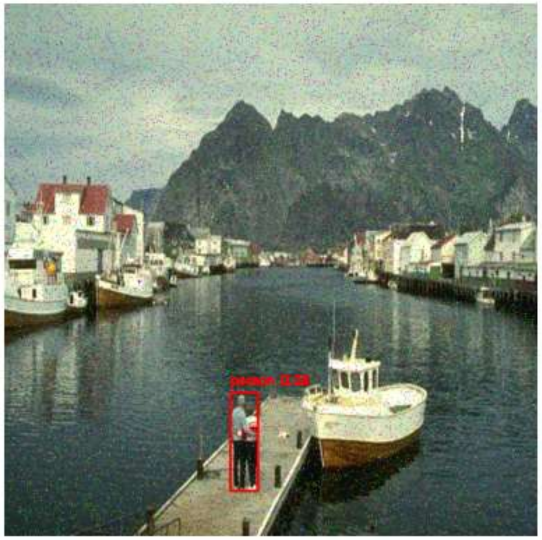
```python
orig = []
for img in noisy:
  orig.append(model.predict(source=img))
```

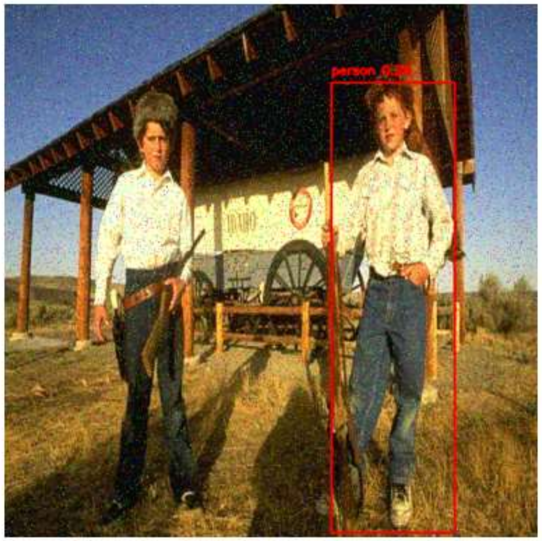
```
0: 640x640 (no detections), 5588.2ms
```

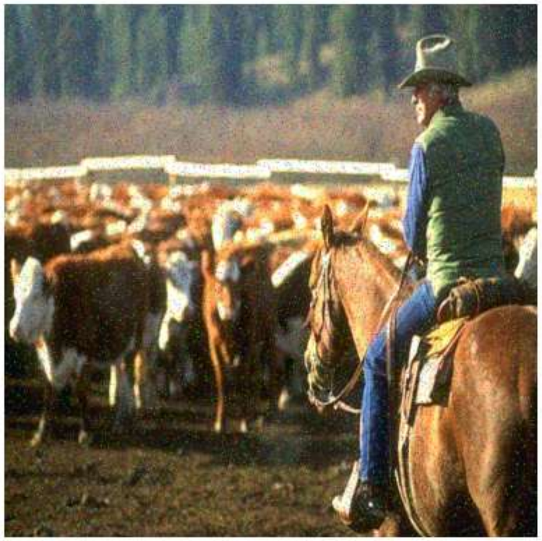
```
Speed: 4.2ms preprocess, 5588.2ms inference, 17.8ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 1 person, 6286.4ms
Speed: 3.3ms preprocess, 6286.4ms inference, 20.7ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 1 person, 4983.9ms
Speed: 3.4ms preprocess, 4983.9ms inference, 1.5ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 5890.0ms
Speed: 3.4ms preprocess, 5890.0ms inference, 1.7ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 5289.6ms
Speed: 3.8ms preprocess, 5289.6ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 4947.3ms
Speed: 3.3ms preprocess, 4947.3ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 6306.9ms
Speed: 4.4ms preprocess, 6306.9ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 4936.5ms
Speed: 3.3ms preprocess, 4936.5ms inference, 1.1ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 7978.3ms
Speed: 4.3ms preprocess, 7978.3ms inference, 1.4ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 4961.9ms
Speed: 3.1ms preprocess, 4961.9ms inference, 1.1ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 5997.9ms
Speed: 2.9ms preprocess, 5997.9ms inference, 1.6ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 6064.1ms
Speed: 5.3ms preprocess, 6064.1ms inference, 1.2ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 5177.5ms
Speed: 4.3ms preprocess, 5177.5ms inference, 1.3ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 6336.2ms
Speed: 3.6ms preprocess, 6336.2ms inference, 1.1ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 4982.8ms
Speed: 3.7ms preprocess, 4982.8ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 6806.8ms
Speed: 4.5ms preprocess, 6806.8ms inference, 2.2ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 7097.7ms
Speed: 13.3ms preprocess, 7097.7ms inference, 1.7ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 6522.2ms
Speed: 4.7ms preprocess, 6522.2ms inference, 1.1ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 5097.3ms
Speed: 3.3ms preprocess, 5097.3ms inference, 1.2ms postprocess per image at shape (1, 3, 640, 640)
```

```
plt.figure(figsize=(12, 20))
for result, i in zip(orig, range(10)):
  plt.subplot(5, 2, i+1)
  plt.imshow(boxes(result, noisy[i]))
```

```
  plt.imshow(boxes(result, noisy[i]))
  plt.axis("off")
plt.tight_layout()
plt.figure(figsize=(12, 20))
l = len(orig[11:])
for result, i in zip(orig[11:], range(l)):
  plt.subplot(l//2+1, 2, i+1)
  plt.imshow(boxes(result, noisy[i+11]))
  plt.axis("off")
plt.tight_layout()
```








CASTROL
Castrol
96







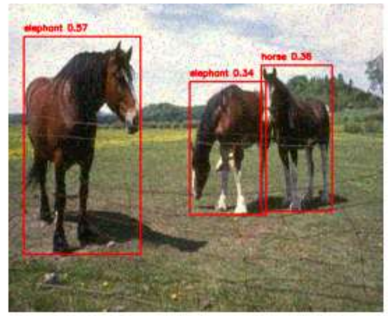











elephant 0.57
elephant 0.34
horse 0.36


clock 0.28