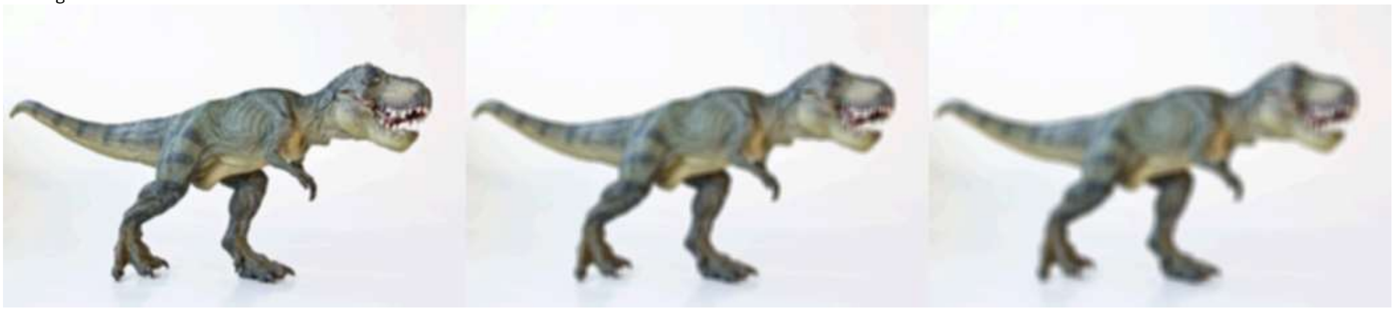
```
# USAGE
# python blurring.py --image ../images/trex.png
# python blurring.py --image ../images/beach.png

# Import the necessary packages
import numpy as np
import cv2
from google.colab.patches import cv2_imshow


# Load the image and show it
image = cv2.imread("trex.png")
print("Original")
cv2_imshow(image)
```

```
Original
```

```
# Let's apply standard "averaging" blurring first. Average
# blurring (as the name suggests), takes the average of all
# pixels in the surrounding area and replaces the centeral
# element of the output image with the average. Thus, in
# order to have a central element, the area surrounding the
# central must be odd. Here are a few examples with varying
# kernel sizes. Notice how the larger the kernel gets, the
# more blurred the image becomes
blurred = np.hstack([
  cv2.blur(image, (3, 3)),
  cv2.blur(image, (5, 5)),
  cv2.blur(image, (7, 7))])
print("Averaged")
cv2_imshow(blurred)
cv2.waitKey(0)
```

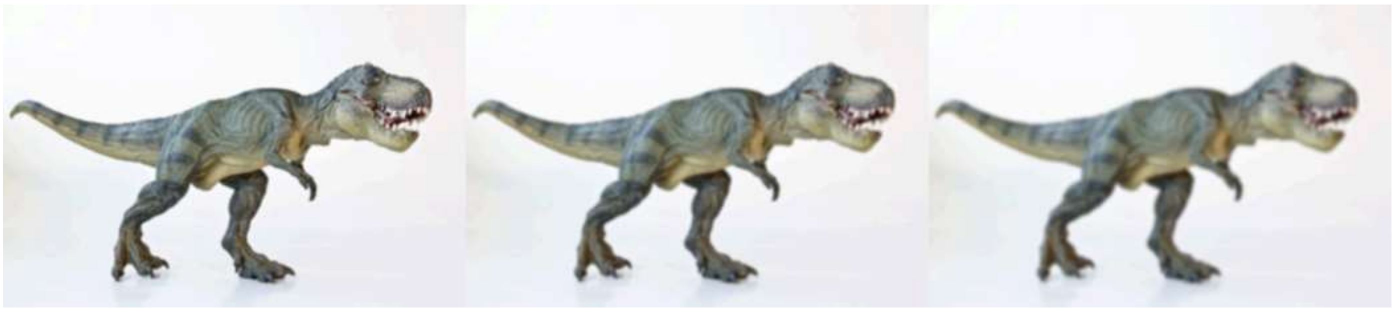
```
Averaged
```



```
-1
```

```
# We can also apply Gaussian blurring, where the relevant
# parameters are the image we want to blur and the standard
# deviation in the X and Y direction. Again, as the standard
# deviation size increases, the image becomes progressively
# more blurred
blurred = np.hstack([
  cv2.GaussianBlur(image, (3, 3), 0),
  cv2.GaussianBlur(image, (5, 5), 0),
  cv2.GaussianBlur(image, (7, 7), 0)])
print("Gaussian")
cv2_imshow(blurred)
cv2.waitKey(0)
```

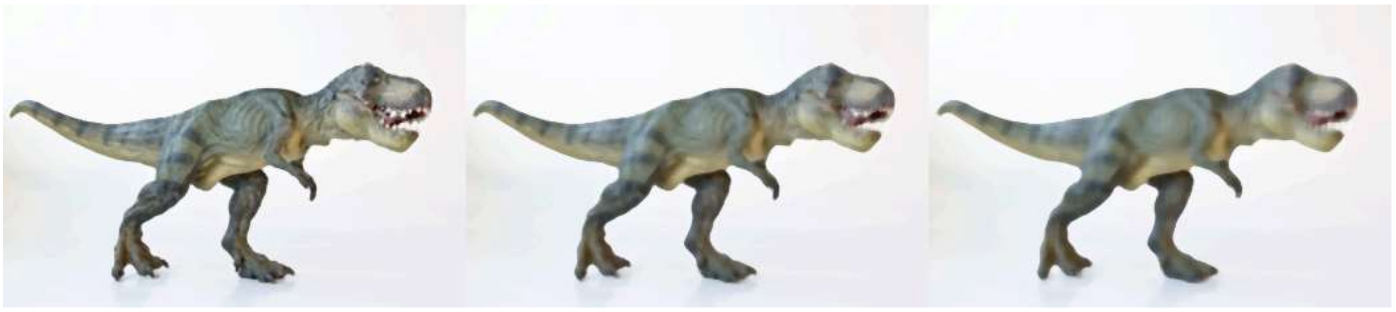
```
Gaussian
```



```
-1
```

```
# The cv2.medianBlur function is mainly used for removing
# what is called "salt-and-pepper" noise. Unlike the Average
# method mentioned above, the median method (as the name
# suggests), calculates the median pixel value amongst the
# surrounding area.
blurred = np.hstack([
    cv2.medianBlur(image, 3),
    cv2.medianBlur(image, 5),
    cv2.medianBlur(image, 7)])
print("Median")
cv2_imshow(blurred)
cv2.waitKey(0)
```

```
Median
```



```
-1
```

```
# You may have noticed that blurring can help remove noise,
# but also makes edge less sharp. In order to keep edges
# sharp, we can use bilateral filtering. We need to specify
# the diameter of the neighborhood (as in examples above),
# along with sigma values for color and coordinate space.
# The larger these sigma values, the more pixels will be
# considered within the neighborhood.
# Diameter: 5, 7, 9
# Sigma values for color and coordinate space: (21, 21), (31, 31), and (41, 41)
# cv2.bilateralFilter()
blurred = np.hstack([
  cv2.bilateralFilter(image, 5, 21, 21),
  cv2.bilateralFilter(image, 7, 31, 31),
  cv2.bilateralFilter(image, 9, 41, 41)])
print("Bilateral")
cv2_imshow(blurred)
cv2.waitKey(0)
```

```
Bilateral
```

```
-1
```