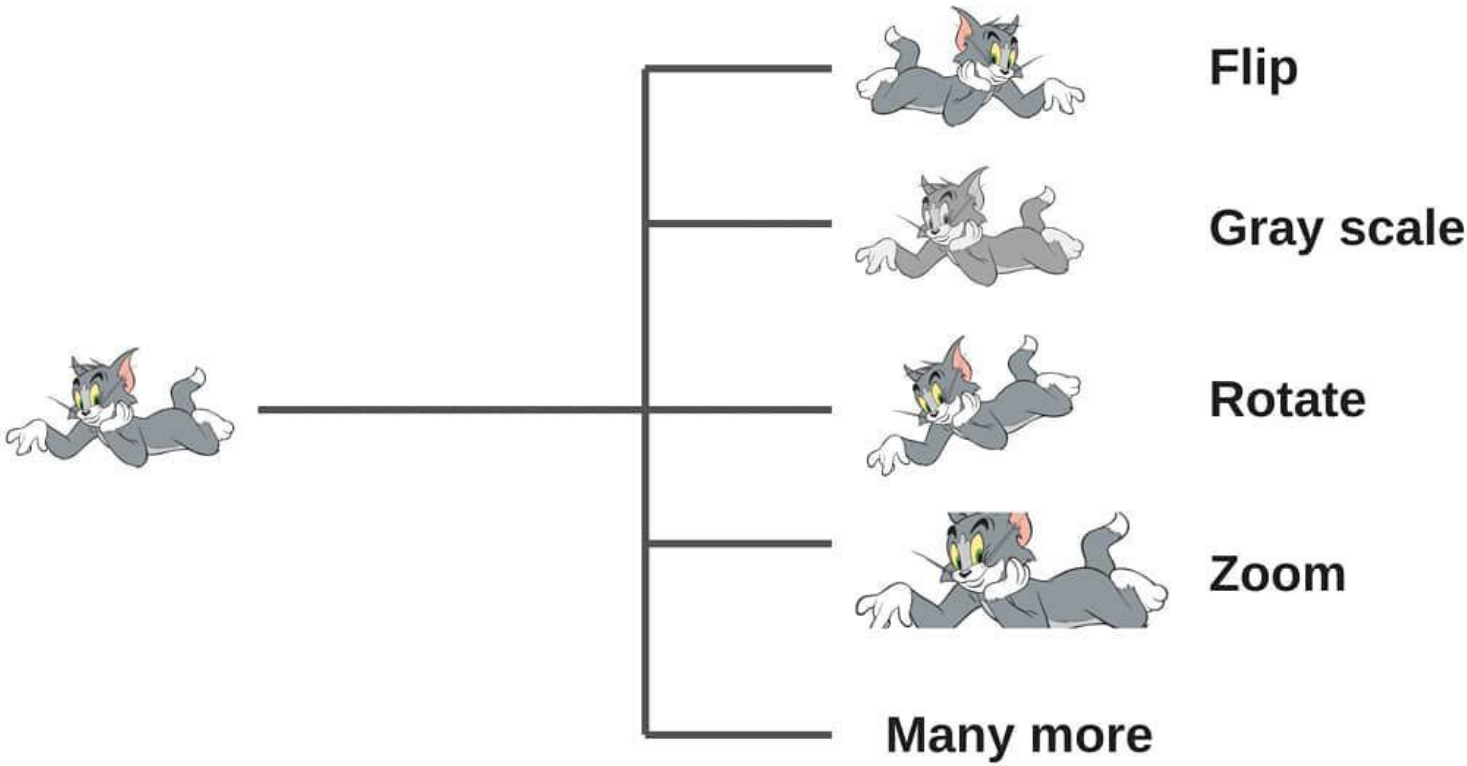
INCREASE YOUR DATASET SIZE USING DIFFERENT DATA AUGMENTATION TECHNIQUES USING

What is Data Augmentation



- It is a technique of artificially increasing the size of data used for training a model, that represents a comprehensive set of possible images.
- Deep learning models often require a lot of training data, which is not always available.
- This is done by applying different transformation techniques like above.

Why Data Augmentation

- This essentially is the premise of data augmentation. In the real world scenario, we may have a dataset of images taken in a limited set of conditions.
- But, our target application may exist in a variety of conditions, such as different orientation, location, scale, brightness etc.
- We account for these situations by training our neural network with additional synthetically modified data.
- Modern deep learning algorithms, such as the convolutional neural network, or CNN, can learn features that are invariant to their location in the image.
- Nevertheless, augmentation can further aid in this transform invariant approach to learning and can aid the model in learning features that are also invariant to transforms such as left-to-right to top-to-bottom ordering, light levels in photographs, and more.

1. FLip

- **We will be using tensorlfow.image module which contains various functions for image processing and decoding-encoding Ops.**

Input

Output

```
# More function https://www.tensorflow.org/api_docs/python/tf/image

# import libraries
import tensorflow as tf
import matplotlib.pyplot as plt

# image = image in a tensor format
flipped = tf.image.flip_left_right(image)
up_down = tf.image.flip_up_down(image)

plt.imshow(flipped)
plt.imshow(up_down)
```

2.brightness

Input 

Output 

```python
# More function https://www.tensorflow.org/api_docs/python/tf/image

# import libraries
import tensorflow as tf
import matplotlib.pyplot as plt

# image = image in a tensor format
# 0.4 = Brightness factor (it can be any)
bright = tf.image.adjust_brightness(image, 0.4)

plt.imshow(bright)
```

3.Rotate

Input

Output

```python
# More function https://www.tensorflow.org/api_docs/python/tf/image

# import libraries
import tensorflow as tf
import matplotlib.pyplot as plt

# image = image in a tensor format
rotated = tf.image.rot90(image)

plt.imshow(rotated)
```

4.Center crop

Input

Output

```python
# More function https://www.tensorflow.org/api_docs/python/tf/image

# import libraries
import tensorflow as tf
import matplotlib.pyplot as plt

# image = image in a tensor format
# central_fraction = (Crop the image from center upto the
#                                    image part you desire)
cropped = tf.image.central_crop(image, central_fraction=0.5)

plt.imshow(cropped)
```

- It is neither practical nor efficient to store the augmented data in memory
- We will be using ImageDataGenerator (TensorFlow's high level api: tensorflow.keras) generates batches of tensor image data with real-time data augmentation.
- We can also use all these functions at once

```python
# import libraries
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# x_train = train dataset
# y_train = labels of train dataset
dataget = ImageDataGenerator(rotation_range=20, brightness_range=None,
          shear_range=0.0, zoom_range=0.5, fill_mode='nearest',
          horizontal_flip=True, vertical_flip=True)

datagen.fit(x_train)

# model = Your model object
# fits the model on batches with real-time data augmentation:
model.fit_generator(datagen.flow(x_train, y_train, batch_size=32),
                    steps_per_epoch=len(x_train) / 32, epochs=epochs)
```