# Data_Augmentation_new

February 4, 2025

## 0.1 Setting Up

[50]:
```
%%capture
%pip install tensorflow_datasets -q
```

[51]:
```
%%capture
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds

from keras import layers
import keras
```

## 0.2 Loading Cat Vs Dog dataset

[26]:
```
%%capture
from tqdm import tqdm as tqdm
(train_ds, val_ds, test_ds), metadata = tfds.load(
    'cats_vs_dogs',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)
```

## 0.3 Data Analysis

[27]:
```
num_classes = metadata.features['label'].num_classes
print(num_classes)
```

```
2
```

[28]:
```
get_label_name = metadata.features['label'].int2str
train_iter = iter(train_ds)
fig = plt.figure(figsize=(7, 8))
for x in range(4):
        image, label = next(train_iter)
```

```
        fig.add_subplot(1, 4, x+1)
        plt.imshow(image)
        plt.axis('off')
        plt.title(get_label_name(label));
```


## 0.4 Data Augmentation with Keras

### 0.4.1 Resize and Rescale

[29]:
```
IMG_SIZE = 180

resize_and_rescale = keras.Sequential([
  layers.Resizing(IMG_SIZE, IMG_SIZE),
  layers.Rescaling(1./255)
])
```

[7]:
```
result = resize_and_rescale(image)
plt.axis('off')
plt.imshow(result);
```

### 0.4.2 Random Rotate and Flip

[30]:
```
data_augmentation = keras.Sequential([
        layers.RandomFlip("horizontal_and_vertical"),
        layers.RandomRotation(0.4),
])
```

[ ]:
```
```

[9]:
```
plt.figure(figsize=(8, 7))
for i in range(6):
  augmented_image = data_augmentation(image)
  ax = plt.subplot(2, 3, i + 1)
  plt.imshow(augmented_image.numpy()/255)
  plt.axis("off")
```

### 0.4.3 Directily adding to Model Layer

### 0.4.4 Applying Augmentation using .map

[23]:
```
aug_ds = train_ds.map(lambda x, y: (data_augmentation(x, training=True), y))
```

[31]:
```
batch_size = 32
AUTOTUNE = tf.data.AUTOTUNE

def prepare(ds, shuffle=False, augment=False):
  # Resize and rescale all datasets.
  ds = ds.map(lambda x, y: (resize_and_rescale(x), y),
              num_parallel_calls=AUTOTUNE)

  if shuffle:
    ds = ds.shuffle(1000)

  # Batch all datasets.
  ds = ds.batch(batch_size)

  # Use data augmentation only on the training set.
  if augment:
```

```
    ds = ds.map(lambda x, y: (data_augmentation(x, training=True), y),
                num_parallel_calls=AUTOTUNE)

  # Use buffered prefetching on all datasets.
  return ds.prefetch(buffer_size=AUTOTUNE)
```

[32]:
```
train_ds = prepare(train_ds, shuffle=True, augment=True)
val_ds = prepare(val_ds)
test_ds = prepare(test_ds)
```

### 0.4.5 Data Processing

### 0.4.6 Model Building

[38]:
```
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), input_shape=(180,180,3), padding='same',␣
 ↪activation='relu'),
                layers.MaxPooling2D(pool_size=(2, 2)),

                layers.Flatten(),
                layers.Dense(32, activation='relu'),
                layers.Dense(1,activation='sigmoid')
])
```

Compiling the model

[39]:
```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

### 0.4.7 Model Training and Evaluation

[40]:
```
epochs=1
history = model.fit(train_ds,
  validation_data=val_ds,
  epochs=epochs )
```

```
582/582             652s 1s/step -
accuracy: 0.5104 - loss: 1.2294 - val_accuracy: 0.5185 - val_loss: 0.6931
```

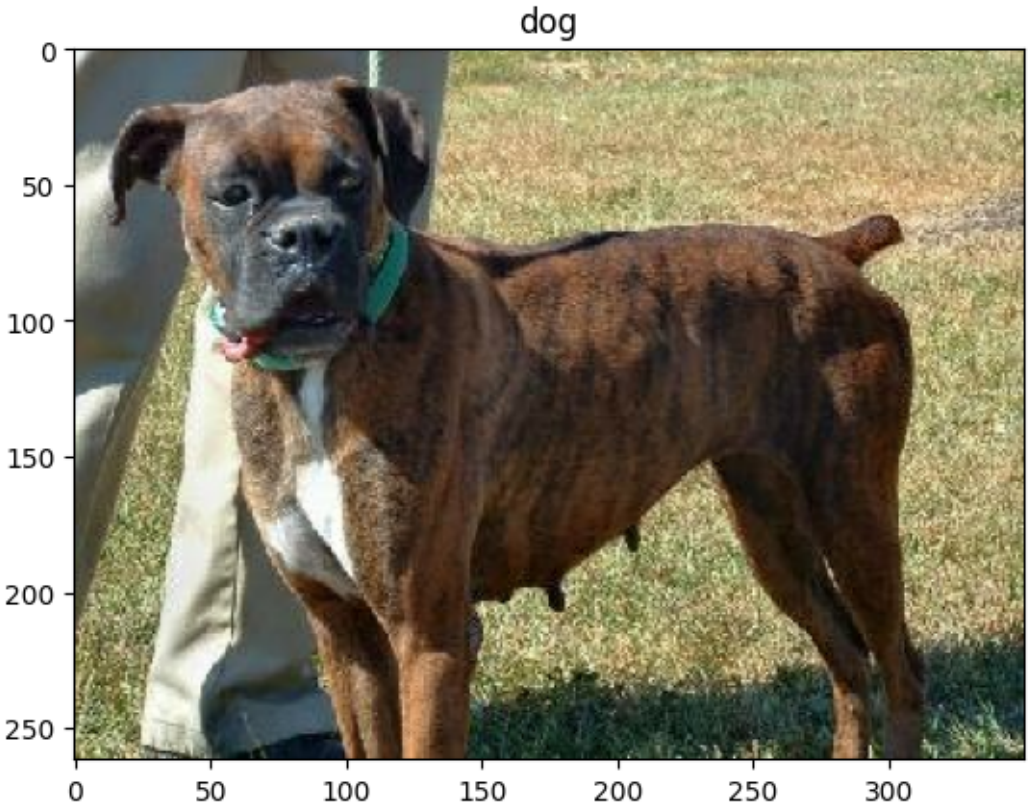
[41]:
```
 loss, acc = model.evaluate(test_ds)
 print("Accuracy", acc)
```

```
73/73             27s 359ms/step -
accuracy: 0.4831 - loss: 0.6932
Accuracy 0.5012897849082947
```

### 0.5 Data Augmentation using tf.image

[52]:
```
%%capture
(train_ds, val_ds, test_ds), metadata = tfds.load(
    'cats_vs_dogs',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)
```

[53]:
```
image, label = next(iter(train_ds))
plt.imshow(image)
plt.title(get_label_name(label));
```

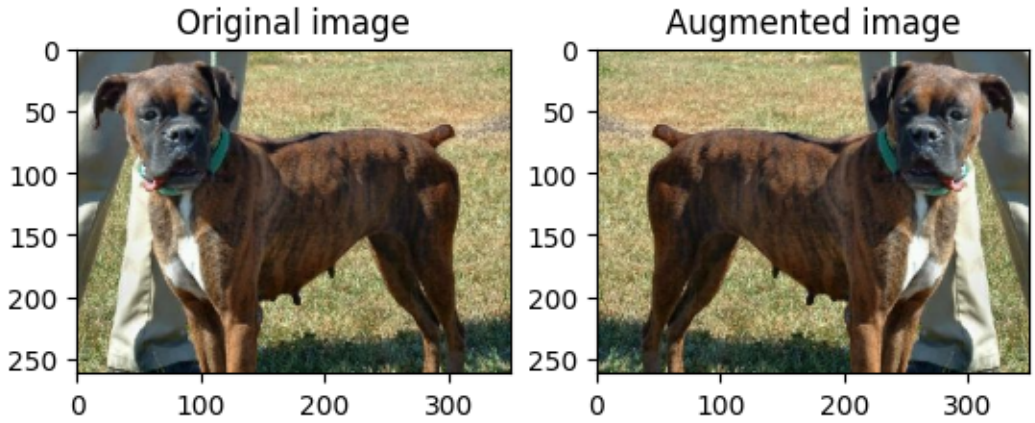



[44]:
```
def visualize(original, augmented):
  fig = plt.figure()
  plt.subplot(1,2,1)
  plt.title('Original image')
  plt.imshow(original)
  #plt.axis("off")
```

```
  plt.subplot(1,2,2)
  plt.title('Augmented image')
  plt.imshow(augmented)
  #plt.axis("off")
```

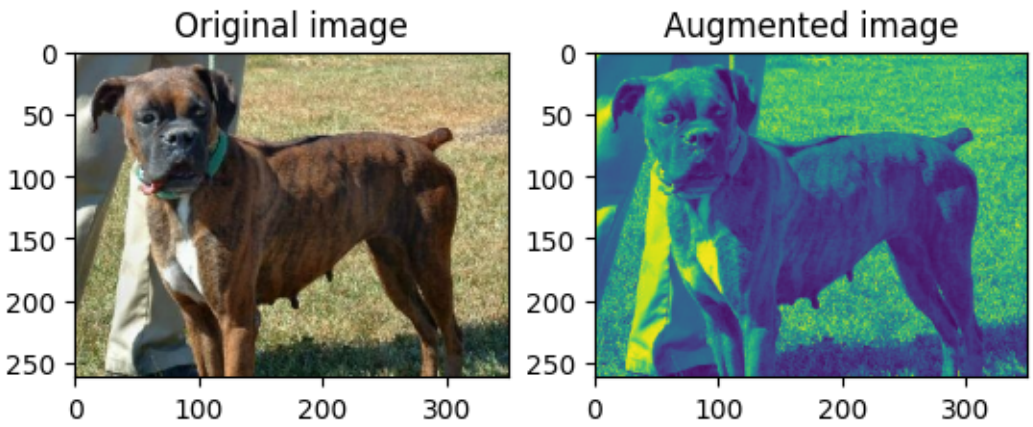
### 0.5.1 Flip left to right

[54]:
```
flipped = tf.image.flip_left_right(image)
visualize(image, flipped)
```




### 0.5.2 Rgb to Grayscale

[58]:
```
grayscaled = tf.image.rgb_to_grayscale(image)



visualize(image,  tf.squeeze(grayscaled))

#visualize(image,  grayscaled)
```




[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

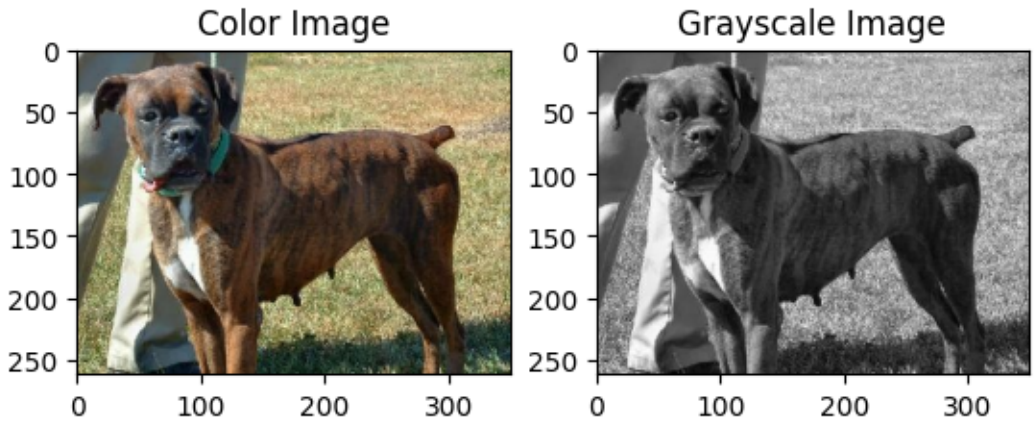
```
Mounted at /content/drive
```

[ ]:
```
%cd drive/My Drive/Deep_Learning_With_Tensorflow
```

```
/content/drive/My Drive/Deep_Learning_With_Tensorflow
```

[48]:
```
import cv2

def convert_to_grayscale(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    return gray_image

# Example usage:
# Assuming 'color_image' is your input color image as a NumPy array
gray_image = convert_to_grayscale(image.numpy())
#visualize(image,  gray_image)


plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Color Image')

# Display the grayscale image using the 'gray' colormap
plt.subplot(1, 2, 2)
plt.imshow(gray_image, cmap='gray')
plt.title('Grayscale Image')

# Show the plots
plt.show()
```

```
cv2.imwrite('gray_image.png', gray_image)
```




[48]: True

### 0.5.3 Adjust Saturation

Saturation refers to the intensity or purity of a color. A fully saturated color is vivid and rich, while a desaturated color is closer to grayscale.

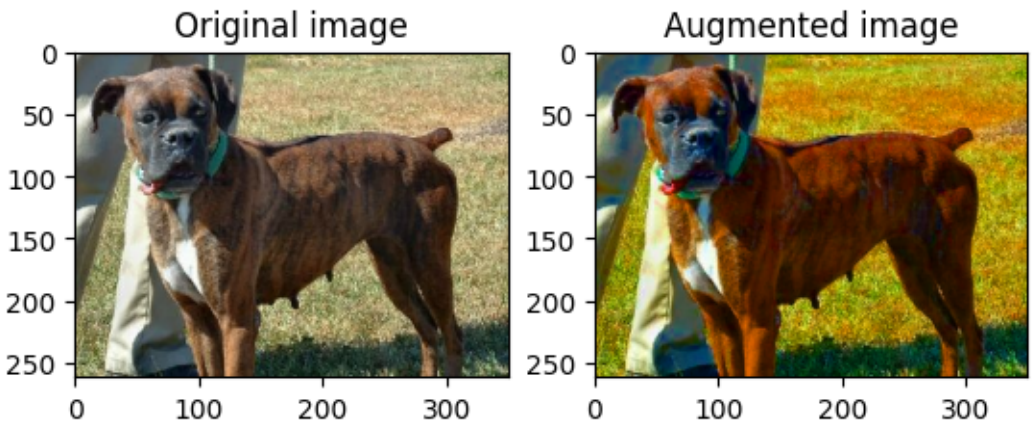
Saturation primarily deals with the vividness or intensity of the color. A high saturation means the color is intense and rich, while a low saturation means the color is more muted and closer to grayscale.

tf.image.adjust_saturation(image, 3) is adjusting the saturation of the input image by a factor of 3.

Here's what it means:

image: This is the input image that you want to adjust. 3: This is the saturation factor. A factor of 1.0 means no change, values less than 1.0 will desaturate the image (move towards grayscale), and values greater than 1.0 will increase the saturation. So, in the case of tf.image.adjust_saturation(image, 3), the image's saturation is increased by a factor of 3. This means the colors in the image will become more vibrant and saturated.

[49]:
```
saturated = tf.image.adjust_saturation(image, 3)
visualize(image, saturated)
```



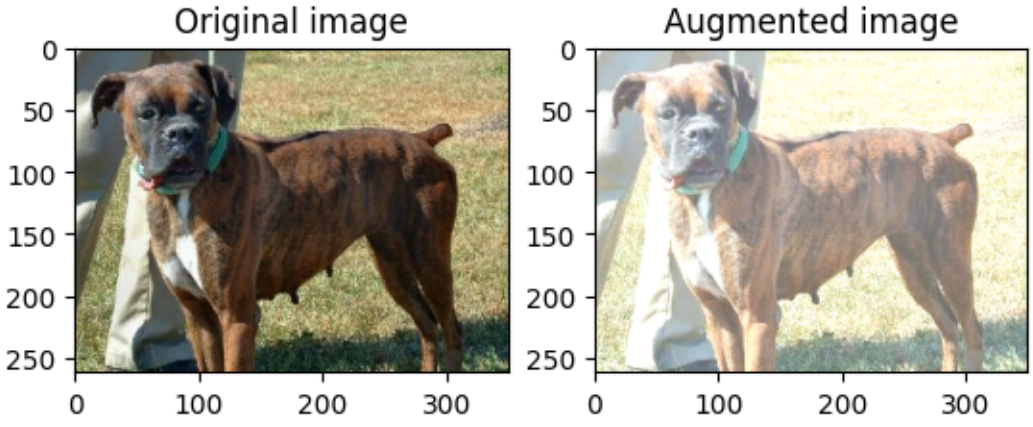

### 0.5.4 Adjust Brightness

Brightness, represents the lightness or darkness of a color. , brightness is typically represented as a percentage, ranging from 0% (black) to 100% (white).

Brightness deals with the overall lightness or darkness of the image. Higher brightness values result in a lighter image, while lower values result in a darker image.

If adjust_brightness(image, 0.4) is applied: 0.4 means that the brightness of the image will be scaled by a factor of 0.4. If brightness is the original brightness of a pixel (ranging from 0 to 1), then the adjusted brightness would be brightness * 0.4.

[59]:
```
bright = tf.image.adjust_brightness(image, 0.4)
visualize(image, bright)
```

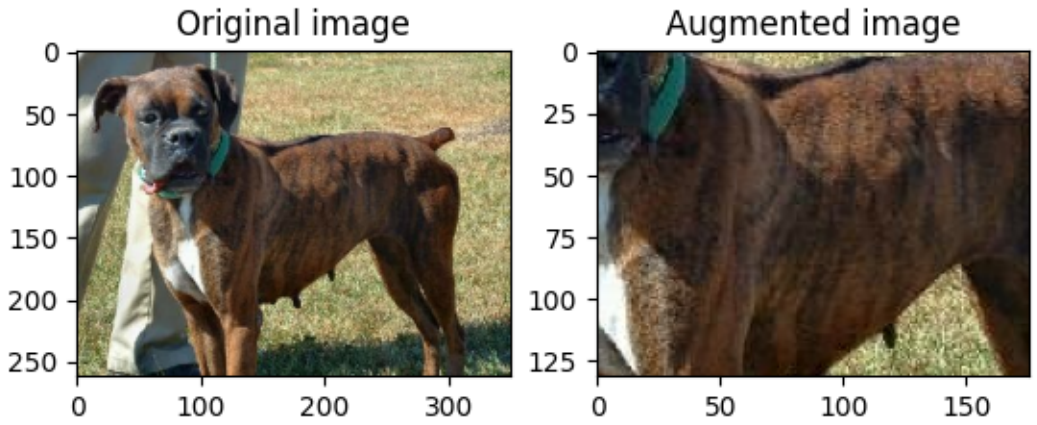


### 0.5.5 Croping

In the case of tf.image.central_crop(image, central_fraction=0.5), 0.5 represents the fraction of the original image's size that will be retained. Specifically:

If central_fraction is set to 0.5, it means that the central portion of the image, corresponding to 50% of both the width and height, will be retained.

[60]:
```
cropped = tf.image.central_crop(image, central_fraction=0.5)
visualize(image, cropped)
```

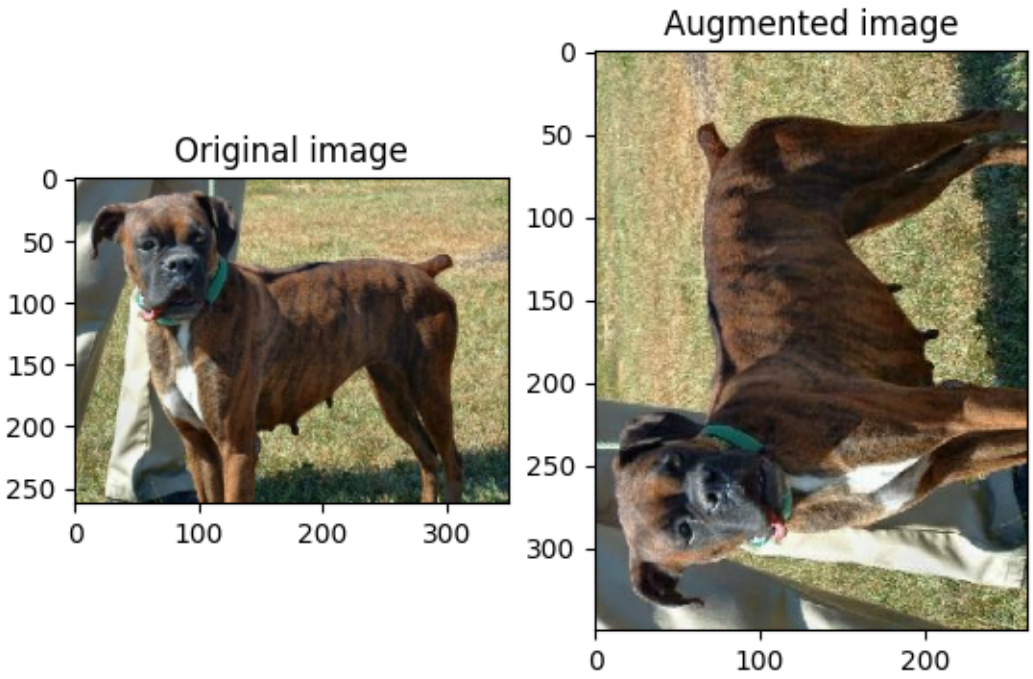



### 0.5.6 Rotating

tf.image.rot90( image, k=1 )

k: An integer. It determines the number of times the image is rotated by 90 degrees. The value of k can be: k=1: Rotate 90 degrees counterclockwise. k=2: Rotate 180 degrees. k=3: Rotate 270 degrees counterclockwise (or 90 degrees clockwise). k=0: No rotation (same as original image).

[61]:
```
rotated = tf.image.rot90(image)    # Rotate anticlockwise by 90 degrees
visualize(image, rotated)
```




### 0.5.7 Apply Augmentation

[62]:
```
def augment(image, label):
  image = tf.cast(image, tf.float32)
  image = tf.image.resize(image, [IMG_SIZE, IMG_SIZE])
  image = (image / 255.0)
  image = tf.image.random_crop(image, size=[IMG_SIZE, IMG_SIZE, 3])
  image = tf.image.random_brightness(image, max_delta=0.5)
  return image, label
```

[63]:
```
train_ds = (
    train_ds
    .shuffle(1000)
    .map(augment, num_parallel_calls=AUTOTUNE)
    .batch(batch_size)
    .prefetch(AUTOTUNE)
)
```

## 0.6 Image Augmentation using ImageDataGenerator

CIFAR-10:

Size: CIFAR-10 consists of 60,000 32x32 color images in 10 different classes, with 6,000 images per class. Classes: The 10 classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and

truck. Purpose: It serves as a benchmark for small-scale image classification tasks. Due to its small size, CIFAR-10 is often used for quick experimentation and prototyping.

[64]:
```
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

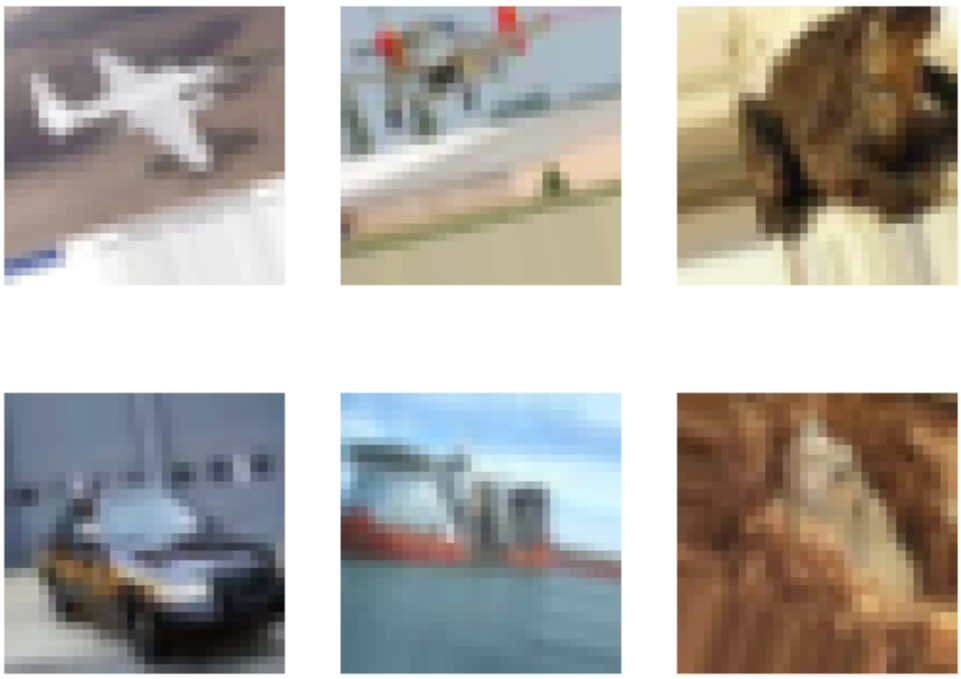
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071             13s
0us/step
```

Rotation_range: Randomly rotates the image by a specified maximum degree. In this case, it rotates the image by a random angle between -20 and 20 degrees.

width_shift_range: Shifts the image horizontally by a fraction of its width. A value of 0.2 means that the image can be shifted horizontally by up to 20% of its width, either to the left or right.

height_shift_range: Shifts the image vertically by a fraction of its height. Similar to width_shift_range, a value of 0.2 means that the image can be shifted vertically by up to 20% of its height, either up or down.

[67]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)

datagen.fit(x_train)

for X_batch, y_batch in datagen.flow(x_train,y_train, batch_size=6):
    for i in range(0, 6):
        plt.subplot(2,3,i+1)
        plt.imshow(X_batch[i]/255)
        plt.axis('off')
    break
```



[ ]:
```
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
# CNN Model definition
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes in CIFAR-10
])

# Compile the model
model.compile(optimizer=Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Train the model using augmented data
history = model.fit(datagen.flow(x_train, y_train, batch_size=64,␣
 ↪subset='training'),
                    validation_data=datagen.flow(x_train, y_train,␣
 ↪batch_size=64, subset='validation'),
                    epochs=20,
                    callbacks=[EarlyStopping(monitor='val_loss', patience=3,␣
 ↪restore_best_weights=True)])

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_acc}")

# Plotting training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```

[ ]: