Introduction to Deep Learning (CS474)

Lecture 3

Outline

- Representing data through Pytorch **Tensor**
    - Introduction
    - Working with image

Introduction

- We learned that tensors are the building blocks for data in PyTorch.

- Neural networks take tensors as input and produce tensors as outputs.

- In fact, all operations within a neural network and during optimization are operations between tensors, and all parameters (for example, weights and biases) in a neural network are tensors.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Introduction

- Having a good sense of how to perform operations on tensors and index them effectively is central to using tools like PyTorch successfully.

- How do we take a piece of data, a video, or a line of text, and represent it with a tensor in a way that is appropriate for training a deep learning model?

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Working with images

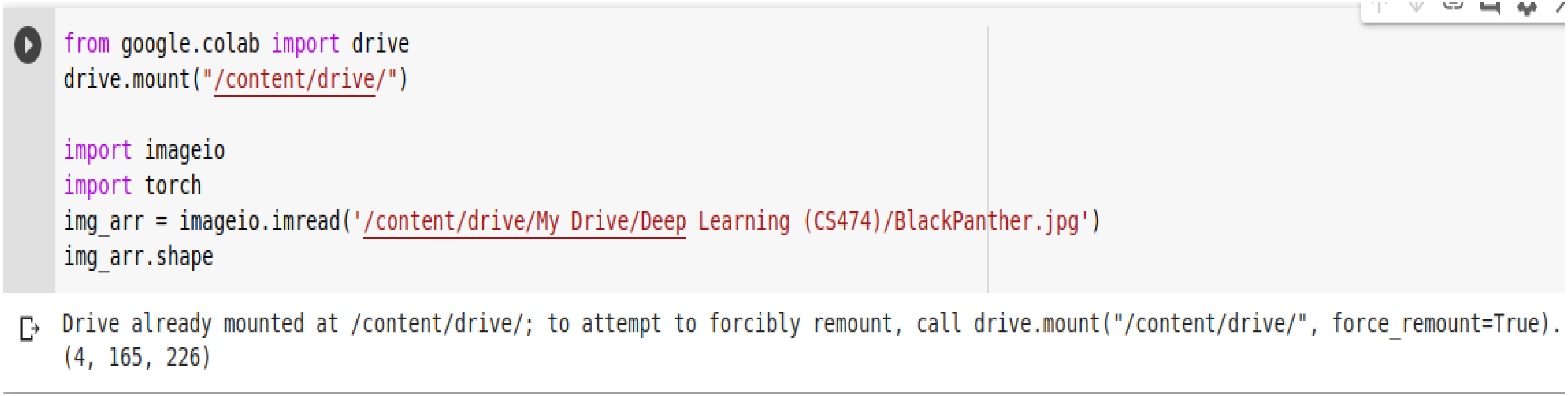
- The introduction of convolutional neural networks revolutionized computer vision and image-based systems have since acquired a whole new set of capabilities.

- In order to participate in this revolution, we need to be able to load an image from common image formats and then transform the data into a tensor representation that has the various parts of the image arranged in the way PyTorch expects.

- Images come in several different file formats, but luckily there are plenty of ways to load images in Python. Let’s start by loading an image using the **imageio** module.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Working with images (contd.)

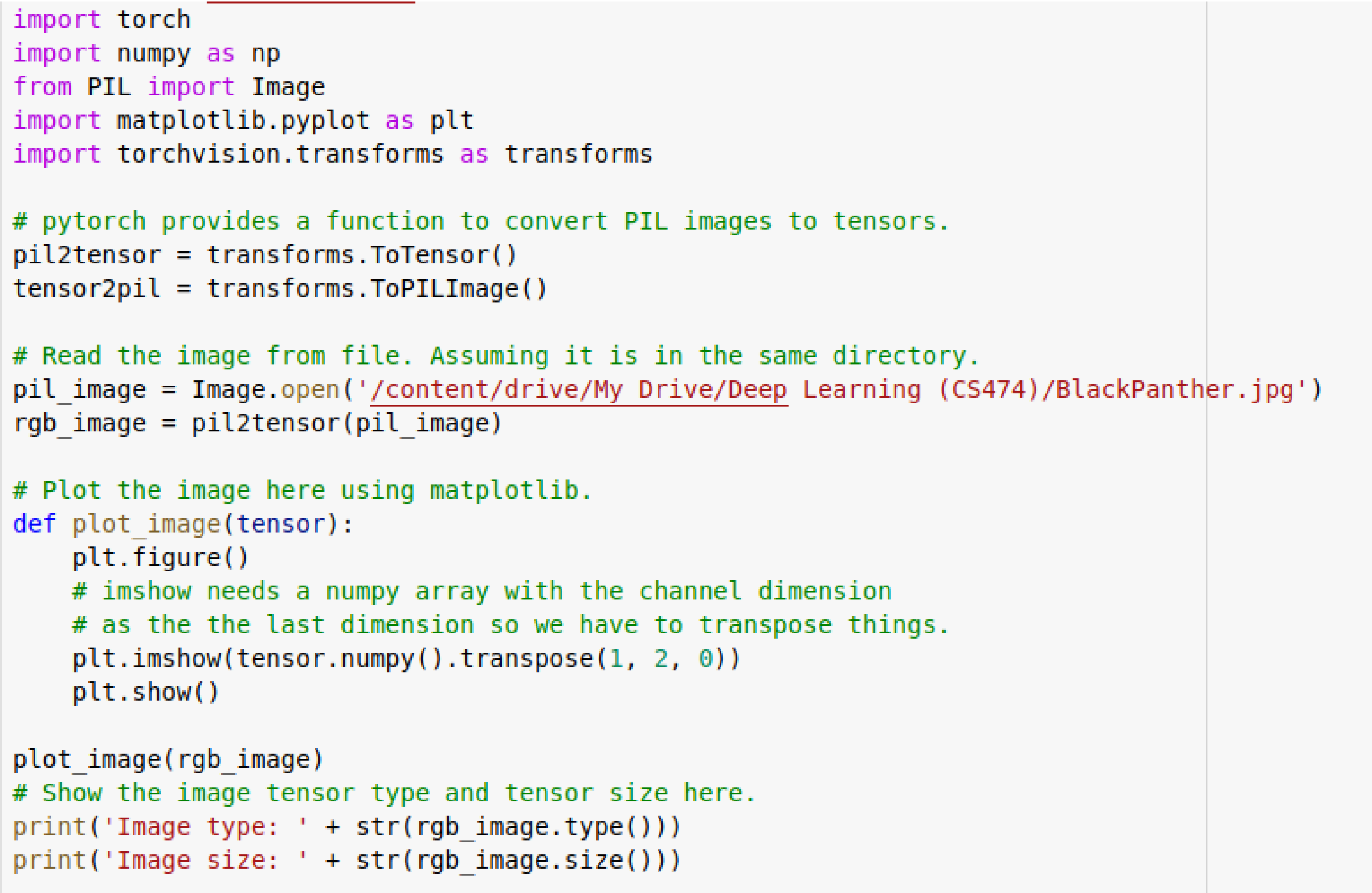


```
from google.colab import drive
drive.mount("/content/drive/")

import imageio
import torch
img_arr = imageio.imread('/content/drive/My Drive/Deep Learning (CS474)/BlackPanther.jpg')
img_arr.shape
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).
(4, 165, 226)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

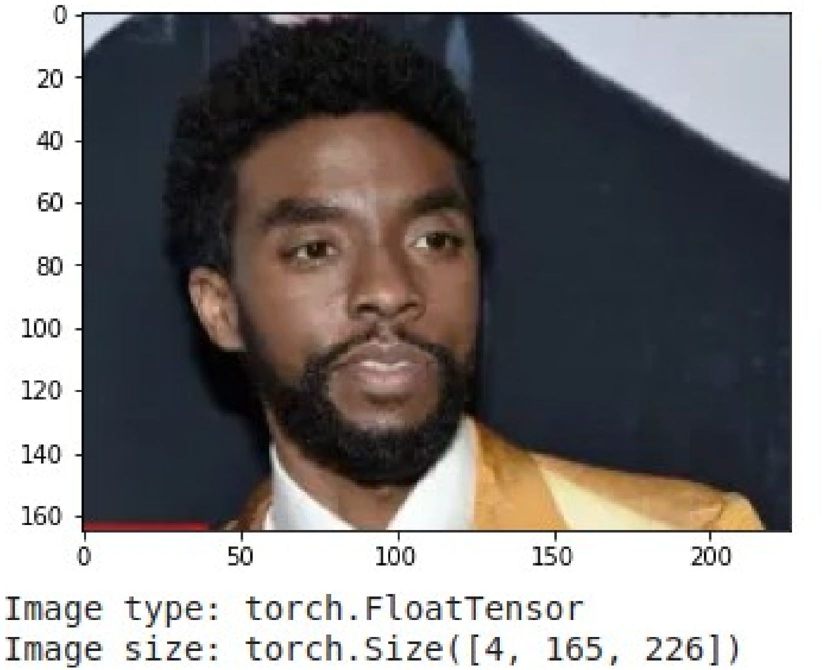
Working with images (contd.)



```python
import torch
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import torchvision.transforms as transforms

# pytorch provides a function to convert PIL images to tensors.
pil2tensor = transforms.ToTensor()
tensor2pil = transforms.ToPILImage()

# Read the image from file. Assuming it is in the same directory.
pil_image = Image.open('/content/drive/My Drive/Deep Learning (CS474)/BlackPanther.jpg')
rgb_image = pil2tensor(pil_image)

# Plot the image here using matplotlib.
def plot_image(tensor):
    plt.figure()
    # imshow needs a numpy array with the channel dimension
    # as the the last dimension so we have to transpose things.
    plt.imshow(tensor.numpy().transpose(1, 2, 0))
    plt.show()

plot_image(rgb_image)
# Show the image tensor type and tensor size here.
print('Image type: ' + str(rgb_image.type()))
print('Image size: ' + str(rgb_image.size()))
```

Slide credit: Vicente Ordóñez R

Working with images (contd.)



```
Image type: torch.FloatTensor
Image size: torch.Size([4, 165, 226])
```

- The **rgb_image** variable contains a **torch.FloatTensor** of size channels x height x width corresponding to the dimensions of the image. Each entry is a floating-point number between 0 and 1.

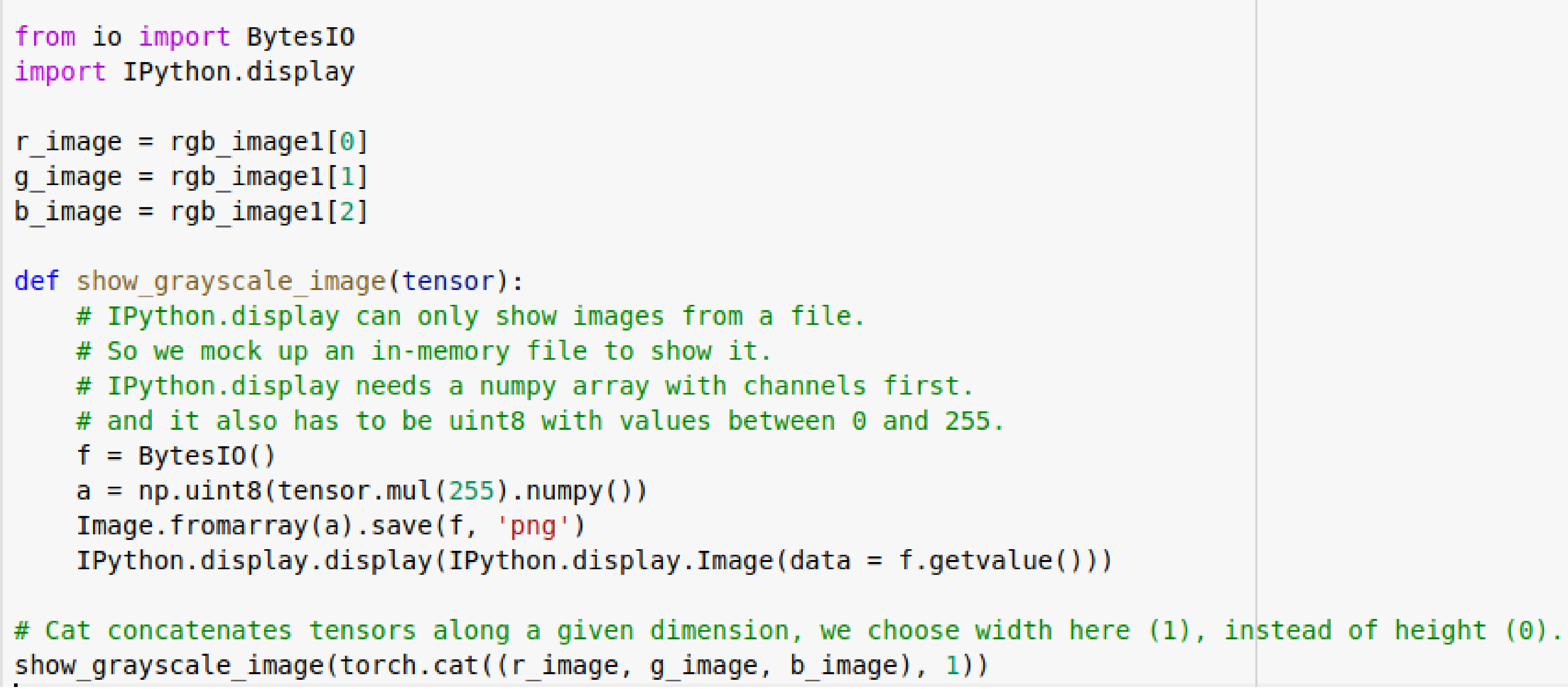
Slide credit: Vicente Ordóñez R

Working with images (contd.)

- Now, we will load the above image to know about ***image channels***. The **rgb_image1** variable contains a **torch.FloatTensor** of size channels x height x width corresponding to the dimensions of the image.

Slide credit: Vicente Ordóñez R

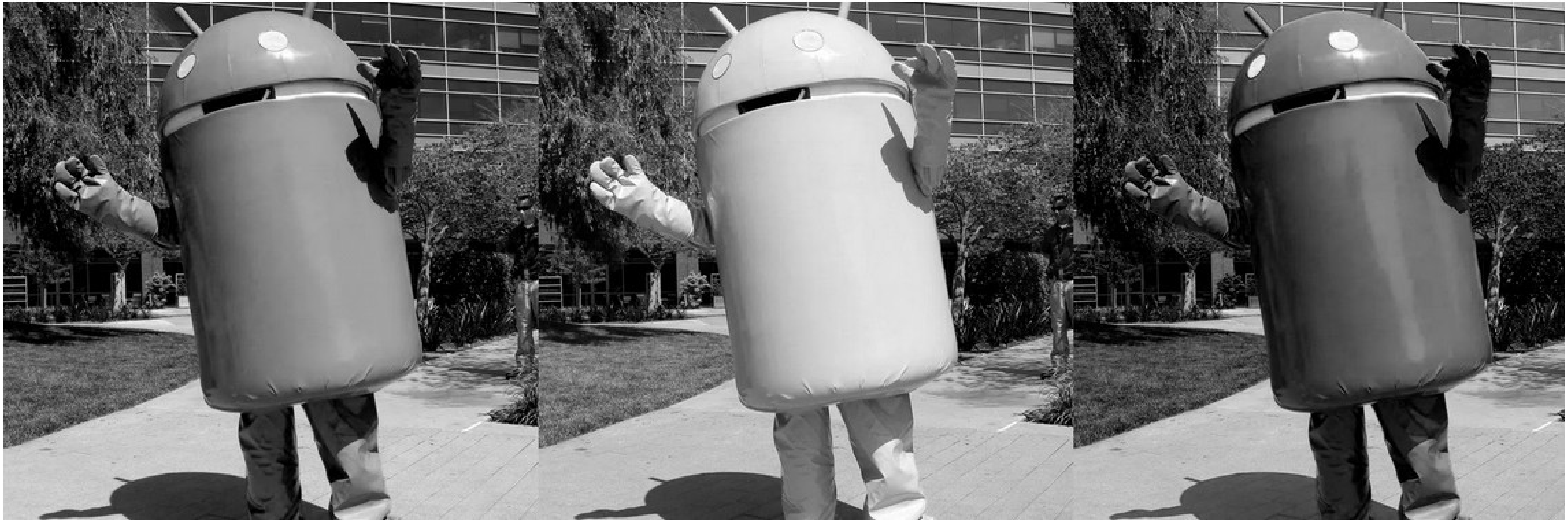
Working with images (contd.)



```
from io import BytesIO
import IPython.display

r_image = rgb_image1[0]
g_image = rgb_image1[1]
b_image = rgb_image1[2]

def show_grayscale_image(tensor):
    # IPython.display can only show images from a file.
    # So we mock up an in-memory file to show it.
    # IPython.display needs a numpy array with channels first.
    # and it also has to be uint8 with values between 0 and 255.
    f = BytesIO()
    a = np.uint8(tensor.mul(255).numpy())
    Image.fromarray(a).save(f, 'png')
    IPython.display.display(IPython.display.Image(data = f.getvalue()))

# Cat concatenates tensors along a given dimension, we choose width here (1), instead of height (0).
show_grayscale_image(torch.cat((r_image, g_image, b_image), 1))
```

Slide credit: Vicente Ordóñez R

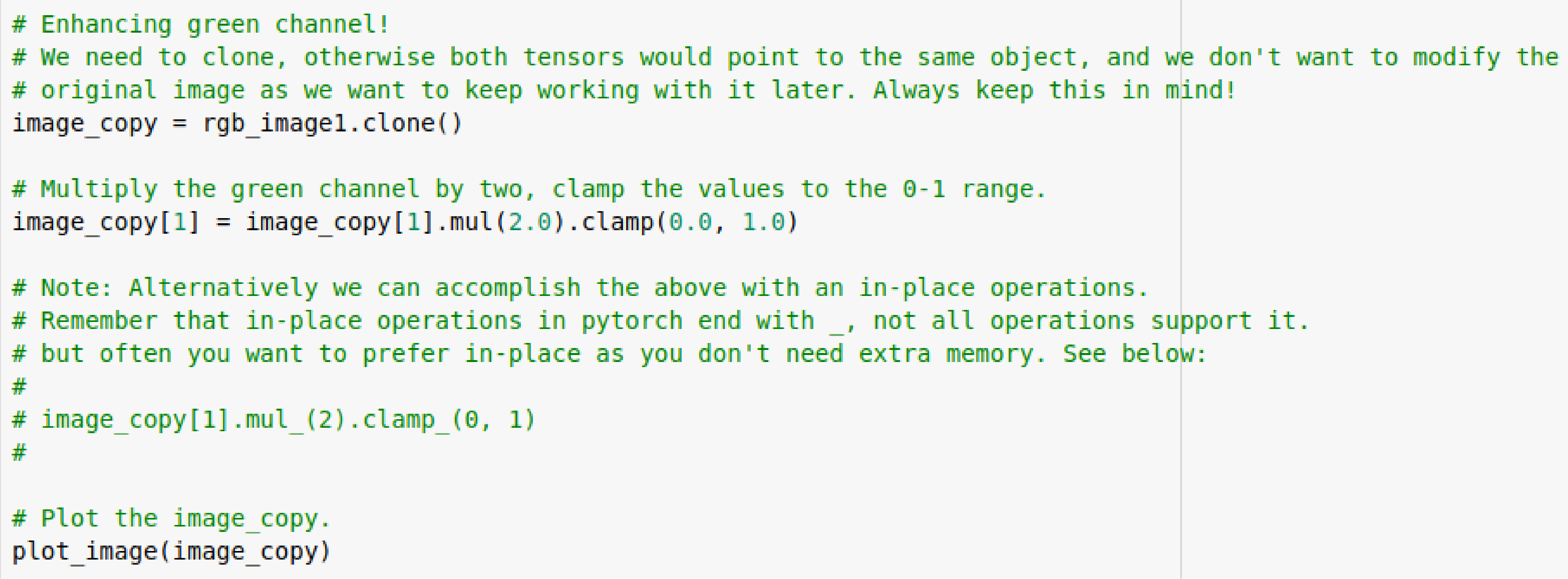
Working with images (contd.)



- Each image in the above code is a one-channel image (e.g. grayscale image) corresponding to each RGB channel. You can clearly notice the Android figure looks brighter in the Green channel (the one in the middle).

Slide credit: Vicente Ordóñez R

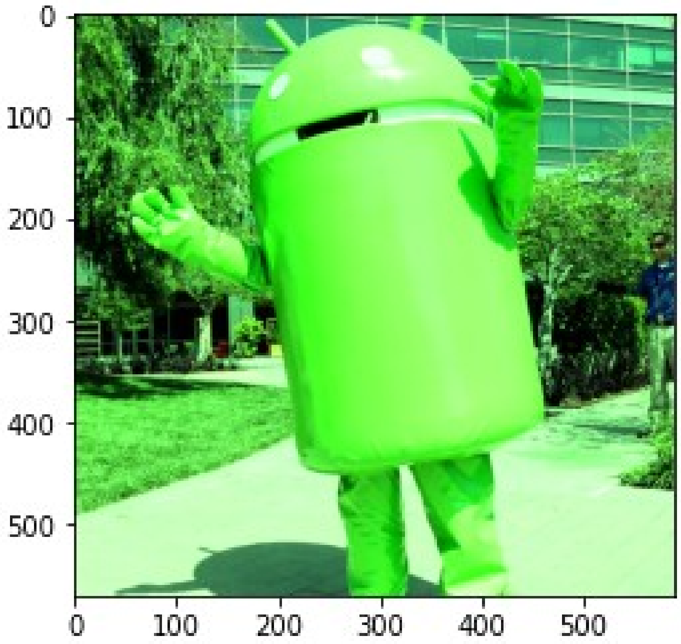
Working with images (contd.)



```
# Enhancing green channel!
# We need to clone, otherwise both tensors would point to the same object, and we don't want to modify the
# original image as we want to keep working with it later. Always keep this in mind!
image_copy = rgb_image1.clone()

# Multiply the green channel by two, clamp the values to the 0-1 range.
image_copy[1] = image_copy[1].mul(2.0).clamp(0.0, 1.0)

# Note: Alternatively we can accomplish the above with an in-place operations.
# Remember that in-place operations in pytorch end with _, not all operations support it.
# but often you want to prefer in-place as you don't need extra memory. See below:
#
# image_copy[1].mul_(2).clamp_(0, 1)
#

# Plot the image_copy.
plot_image(image_copy)
```

Slide credit: Vicente Ordóñez R

Working with images (contd.)



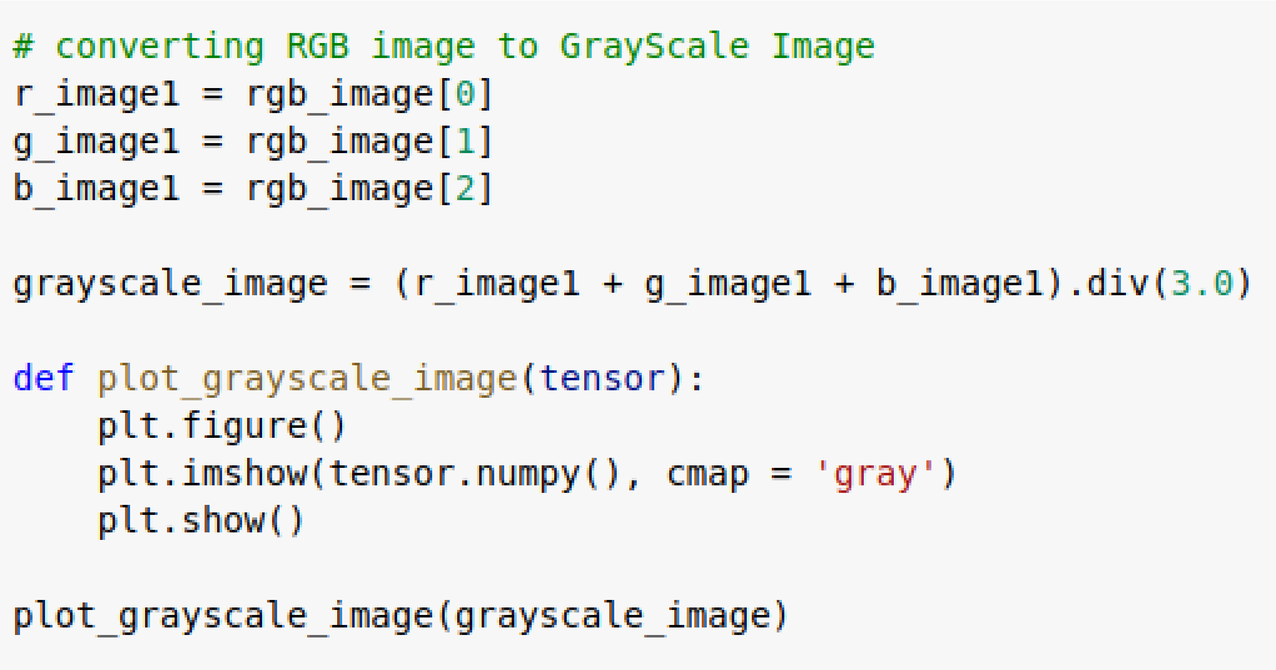
- We have enhanced the green channel in the original image by multiplying this channel by a constant.

Slide credit: Vicente Ordóñez R

Working with images (contd.)

- How do we convert an RGB image to grayscale?

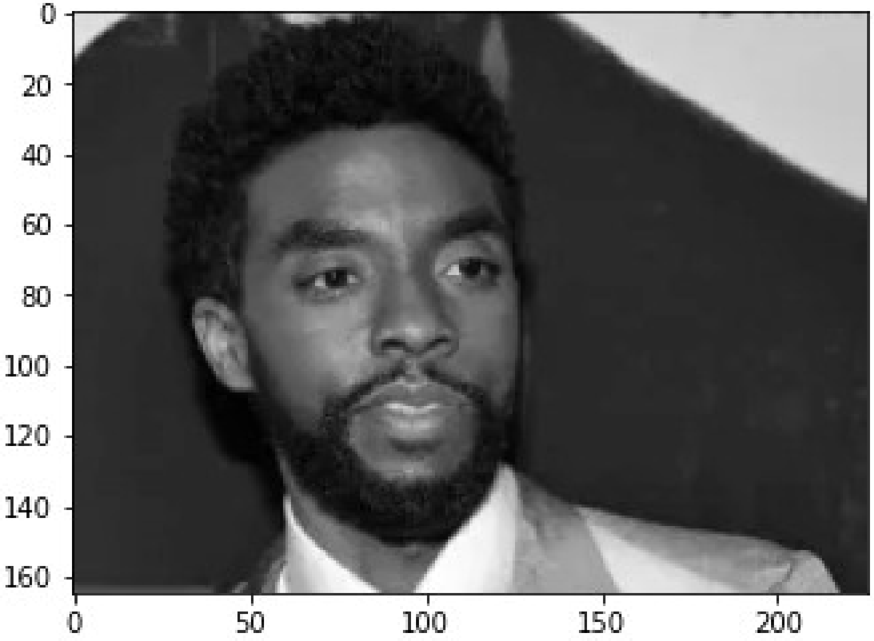
- A simple way would be to average all three RGB channels. Note the division by 3, since each channel has values between 0 and 1, we want to make sure the resulting grayscale image also has values between 0 and 1.



```
# converting RGB image to GrayScale Image
r_image1 = rgb_image[0]
g_image1 = rgb_image[1]
b_image1 = rgb_image[2]

grayscale_image = (r_image1 + g_image1 + b_image1).div(3.0)

def plot_grayscale_image(tensor):
    plt.figure()
    plt.imshow(tensor.numpy(), cmap = 'gray')
    plt.show()

plot_grayscale_image(grayscale_image)
```

Slide credit: Vicente Ordóñez R

Working with images (contd.)

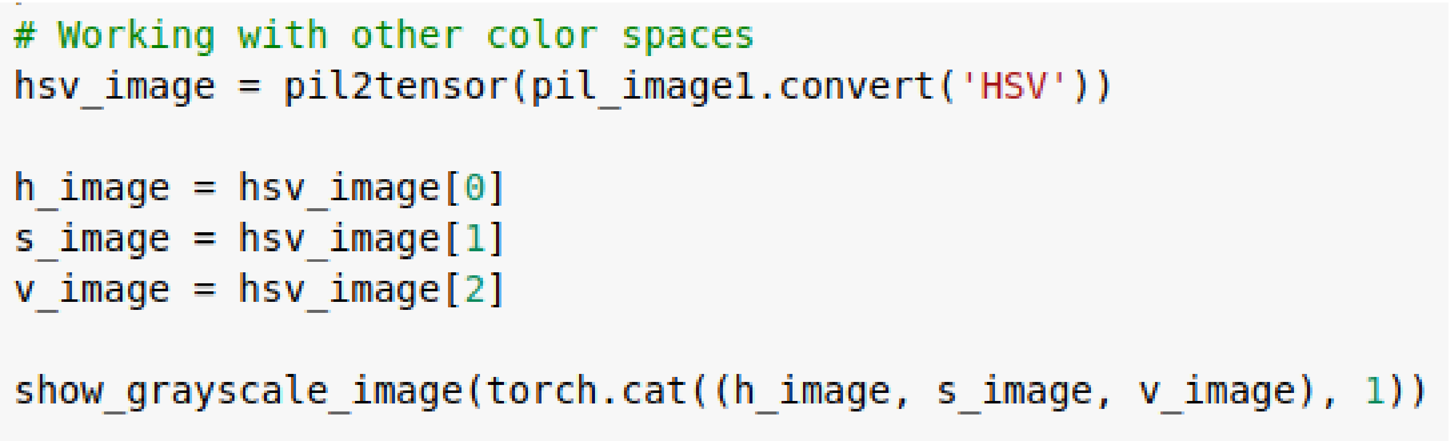


Slide credit: Vicente Ordóñez R

Working with images (contd.)

- Other Colour Spaces!

- In addition to RGB images, we can represent images as HSV, where each channel corresponds to Hue, Saturation and Value (~lightness) instead. Other color spaces include: HSV, Lab, YUV, etc.



```
# Working with other color spaces
hsv_image = pil2tensor(pil_image1.convert('HSV'))

h_image = hsv_image[0]
s_image = hsv_image[1]
v_image = hsv_image[2]

show_grayscale_image(torch.cat((h_image, s_image, v_image), 1))
```

Slide credit: Vicente Ordóñez R

Working with images (contd.)

References

- All the contents present in the slides are taken from various online resources. Due credit is given in the respective slides.These slides are used for *academic* purposes only.