Open in Colab

T81-558: Applications of Deep Neural Networks

Module 6: Convolutional Neural Networks (CNN) for Computer Vision

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 6 Material

- **Part 6.1: Image Processing in Python** [Video] [Notebook]
- Part 6.2: Using Convolutional Neural Networks [Video] [Notebook]
- Part 6.3: Using Pretrained Neural Networks with Keras [Video] [Notebook]
- Part 6.4: Looking at Keras Generators and Image Augmentation [Video] [Notebook]
- Part 6.5: Recognizing Multiple Images with YOLOv5 [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [ ]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False

# Nicely formatted time string
def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return f"{h}:{m:>02}:{s:>05.2f}"
```

```
Note: using Google CoLab
```

Part 6.1: Image Processing in Python

Computer vision requires processing images. These images might come from a

video stream, a camera, or files on a storage drive. We begin this chapter by looking at how to process images with Python. To use images in Python, we will make use of the Pillow package. The following program uses Pillow to load and display an image.

In [ ]:

```
from PIL import Image, ImageFile
from matplotlib.pyplot import imshow
import requests
from io import BytesIO
import numpy as np

%matplotlib inline

url = "https://data.heatonresearch.com/images/jupyter/brookings.jpeg"

response = requests.get(url,headers={'User-Agent': 'Mozilla/5.0'})
img = Image.open(BytesIO(response.content))
img.load()

print(np.asarray(img))

img
```

```
[[[199 213 240]
  [200 214 240]
  [200 214 240]
  ...
  [ 86  34  96]
  [ 48   4  57]
  [ 57  21  65]]

 [[199 213 239]
  [200 214 240]
  [200 214 240]
  ...
  [215 215 251]
  [252 242 255]
  [237 218 250]]

 [[200 214 240]
  [200 214 240]
  [201 215 241]
  ...
  [227 238 255]
  [167 180 197]
  [ 61  79  91]]

 ...

 [[136 112 108]
  [137 113 109]
  [140 116 112]
  ...
  [ 85  84  63]
  [ 91  90  69]
  [ 93  92  72]]

 [[119  90  84]
  [118  89  83]
  [119  90  84]
  ...
  [ 86  84  61]
  [ 89  87  64]
  [ 90  88  65]]

 [[129  96  89]
  [129  96  89]
  [131  98  91]
  ...
  [ 86  82  57]
  [ 89  85  60]
  [ 89  85  60]]]
```

Out[ ]:

Creating Images from Pixels in Python

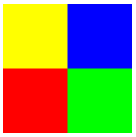
You can use Pillow to create an image from a 3D NumPy cube-shaped array. The rows and columns specify the pixels. The third dimension (size 3) defines red, green, and blue color values. The following code demonstrates creating a simple image from a NumPy array.

In [ ]:

```
from PIL import Image
import numpy as np

w, h = 64, 64
data = np.zeros((h, w, 3), dtype=np.uint8)

# Yellow
for row in range(32):
    for col in range(32):
        data[row,col] = [255,255,0]

# Red
for row in range(32):
    for col in range(32):
        data[row+32,col] = [255,0,0]

# Green
for row in range(32):
    for col in range(32):
        data[row+32,col+32] = [0,255,0]

# Blue
for row in range(32):
    for col in range(32):
```

```
        data[row,col+32] = [0,0,255]


img = Image.fromarray(data, 'RGB')
img
```

Out[ ]:



Transform Images in Python (at the pixel level)

We can combine the last two programs and modify images. Here we take the mean color of each pixel and form a grayscale image.

In [ ]:

```
from PIL import Image, ImageFile
from matplotlib.pyplot import imshow
import requests
from io import BytesIO

%matplotlib inline

url = "https://data.heatonresearch.com/images/jupyter/brookings.jpeg"
response = requests.get(url,headers={'User-Agent': 'Mozilla/5.0'})

img = Image.open(BytesIO(response.content))
img.load()

img_array = np.asarray(img)
rows = img_array.shape[0]
cols = img_array.shape[1]

print("Rows: {}, Cols: {}".format(rows,cols))

# Create new image
img2_array = np.zeros((rows, cols, 3), dtype=np.uint8)
for row in range(rows):
    for col in range(cols):
        t = np.mean(img_array[row,col])
        img2_array[row,col] = [t,t,t]

img2 = Image.fromarray(img2_array, 'RGB')
img2
```

```
Rows: 768, Cols: 1024
```

Out[ ]:

Standardize Images

When processing several images together, it is sometimes essential to standardize them. The following code reads a sequence of images and causes them to all be of the same size and perfectly square. If the input images are not square, cropping will occur.

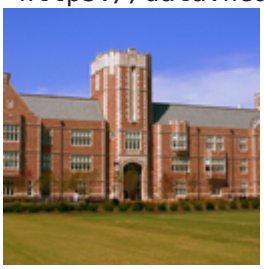
In [ ]:

```
%matplotlib inline
from PIL import Image, ImageFile
from matplotlib.pyplot import imshow
import requests
import numpy as np
from io import BytesIO
from IPython.display import display, HTML

images = [
  "https://data.heatonresearch.com/images/jupyter/brookings.jpeg",
  "https://data.heatonresearch.com/images/jupyter/SeigleHall.jpeg",
  "https://data.heatonresearch.com/images/jupyter/WUSTLKnight.jpeg"
]

def crop_square(image):
    width, height = image.size

    # Crop the image, centered
    new_width = min(width,height)
    new_height = new_width
    left = (width - new_width)/2
    top = (height - new_height)/2
    right = (width + new_width)/2
    bottom = (height + new_height)/2
```

```
    return image.crop((left, top, right, bottom))

x = []

for url in images:
    ImageFile.LOAD_TRUNCATED_IMAGES = False
    response = requests.get(url,headers={'User-Agent': 'Mozilla/5.0'})
    img = Image.open(BytesIO(response.content))
    img.load()
    img = crop_square(img)
    img = img.resize((128,128), Image.ANTIALIAS)
    print(url)
    display(img)
    img_array = np.asarray(img)
    img_array = img_array.flatten()
    img_array = img_array.astype(np.float32)
    img_array = (img_array-128)/128
    x.append(img_array)


x = np.array(x)

print(x.shape)
```

```
https://data.heatonresearch.com/images/jupyter/brookings.jpeg
```

```
https://data.heatonresearch.com/images/jupyter/SeigleHall.jpeg
```



```
https://data.heatonresearch.com/images/jupyter/WUSTLKnight.jpeg
```

```
(3, 49152)
```

Adding Noise to an Image

Sometimes it is beneficial to add noise to images. We might use noise to augment images to generate more training data or modify images to test the recognition capabilities of neural networks. It is essential to see how to add noise to an image. There are many ways to add such noise. The following code adds random black

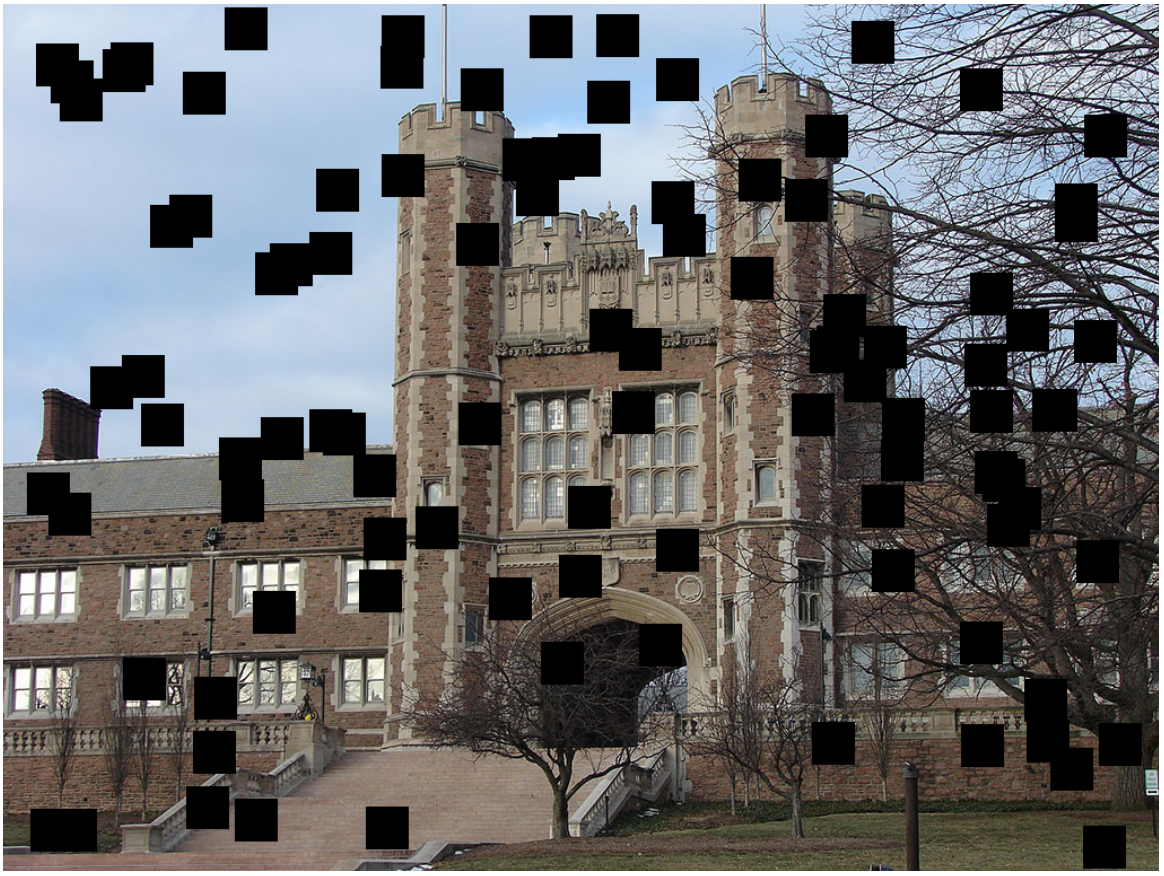
squares to the image to produce noise.

In [ ]:

```
from PIL import Image, ImageFile
from matplotlib.pyplot import imshow
import requests
from io import BytesIO

%matplotlib inline


def add_noise(a):
    a2 = a.copy()
    rows = a2.shape[0]
    cols = a2.shape[1]
    s = int(min(rows,cols)/20) # size of spot is 1/20 of smallest dimens

    for i in range(100):
        x = np.random.randint(cols-s)
        y = np.random.randint(rows-s)
        a2[y:(y+s),x:(x+s)] = 0

    return a2

url = "https://data.heatonresearch.com/images/jupyter/brookings.jpeg"

response = requests.get(url,headers={'User-Agent': 'Mozilla/5.0'})
img = Image.open(BytesIO(response.content))
img.load()

img_array = np.asarray(img)
rows = img_array.shape[0]
cols = img_array.shape[1]

print("Rows: {}, Cols: {}".format(rows,cols))

# Create new image
img2_array = img_array.astype(np.uint8)
print(img2_array.shape)
img2_array = add_noise(img2_array)
img2 = Image.fromarray(img2_array, 'RGB')
img2
```

```
Rows: 768, Cols: 1024
(768, 1024, 3)
```

Out[ ]:



Preprocessing Many Images

To download images, we define several paths. We will download sample images of paperclips from the URL specified by **DOWNLOAD_SOURCE**. Once downloaded, we will unzip and perform the preprocessing on these paper clips. I mean for this code as a starting point for other image preprocessing.

In [ ]:

```
import os

URL = "https://github.com/jeffheaton/data-mirror/releases/"
#DOWNLOAD_SOURCE = URL+"download/v1/iris-image.zip"
DOWNLOAD_SOURCE = URL+"download/v1/paperclips.zip"
DOWNLOAD_NAME = DOWNLOAD_SOURCE[DOWNLOAD_SOURCE.rfind('/')+1:]

if COLAB:
  PATH = "/content"
  EXTRACT_TARGET = os.path.join(PATH,"clips")
  SOURCE = os.path.join(PATH, "/content/clips/paperclips")
  TARGET = os.path.join(PATH,"/content/clips-processed")
else:
  # I used this locally on my machine, you may need different
  PATH = "/Users/jeff/temp"
  EXTRACT_TARGET = os.path.join(PATH,"clips")
  SOURCE = os.path.join(PATH, "clips/paperclips")
  TARGET = os.path.join(PATH,"clips-processed")
```

Next, we download the images. This part depends on the origin of your images. The following code downloads images from a URL, where a ZIP file contains the images. The code unzips the ZIP file.

In [ ]:
```
# HIDE OUTPUT
!wget -O {os.path.join(PATH,DOWNLOAD_NAME)} {DOWNLOAD_SOURCE}
!mkdir -p {SOURCE}
!mkdir -p {TARGET}
!mkdir -p {EXTRACT_TARGET}
!unzip -o -j -d {SOURCE} {os.path.join(PATH, DOWNLOAD_NAME)} >/dev/null
```

```
--2021-11-26 19:11:35--  https://github.com/jeffheaton/data-mirror/release
s/download/v1/paperclips.zip
Resolving github.com (github.com)... 140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-
asset-2e65be/408419764/25830812-b9e6-4ddf-93b6-7932d9ef5982?X-Amz-Algorith
m=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20211126%2Fus-e
ast-1%2Fs3%2Faws4_request&X-Amz-Date=20211126T191135Z&X-Amz-Expires=300&X-
Amz-Signature=37ac15e40f8fcdc15ad13d36ef58562e1fdab5e74d71e8e6adedd5cdf580
8c60&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=408419764&respon
se-content-disposition=attachment%3B%20filename%3Dpaperclips.zip&response-
content-type=application%2Foctet-stream [following]
--2021-11-26 19:11:35--  https://objects.githubusercontent.com/github-prod
uction-release-asset-2e65be/408419764/25830812-b9e6-4ddf-93b6-7932d9ef598
2?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2
F20211126%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20211126T191135Z&X-Am
z-Expires=300&X-Amz-Signature=37ac15e40f8fcdc15ad13d36ef58562e1fdab5e74d71
e8e6adedd5cdf5808c60&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=
408419764&response-content-disposition=attachment%3B%20filename%3Dpapercli
ps.zip&response-content-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.co
m)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 163590691 (156M) [application/octet-stream]
Saving to: '/content/paperclips.zip'

/content/paperclips 100%[===================>] 156.01M  29.9MB/s    in 5.4
s

2021-11-26 19:11:41 (29.1 MB/s) - '/content/paperclips.zip' saved [1635906
91/163590691]
```

The following code contains functions that we use to preprocess the images. The **crop_square** function converts images to a square by cropping extra data. The **scale** function increases or decreases the size of an image. The **standardize** function ensures an image is full color; a mix of color and grayscale images can be problematic.

In [ ]:
```
import imageio
import glob
from tqdm import tqdm
from PIL import Image
import os

def scale(img, scale_width, scale_height):
    # Scale the image
    img = img.resize((
        scale_width,
```

```
        scale_height),
        Image.ANTIALIAS)

    return img

def standardize(image):
    rgbimg = Image.new("RGB", image.size)
    rgbimg.paste(image)
    return rgbimg

def fail_below(image, check_width, check_height):
    width, height = image.size
    assert width == check_width
    assert height == check_height
```

Next, we loop through each image. The images are loaded, and you can apply any desired transformations. Ultimately, the script saves the images as JPG.

In [ ]:

```
files = glob.glob(os.path.join(SOURCE,"*.jpg"))

for file in tqdm(files):
    try:
        target = ""
        name = os.path.basename(file)
        filename, _ = os.path.splitext(name)
        img = Image.open(file)
        img = standardize(img)
        img = crop_square(img)
        img = scale(img, 128, 128)
        #fail_below(img, 128, 128)

        target = os.path.join(TARGET,filename+".jpg")
        img.save(target, quality=25)
    except KeyboardInterrupt:
        print("Keyboard interrupt")
        break
    except AssertionError:
        print("Assertion")
        break
    except:
        print("Unexpected exception while processing image source: " \
              f"{file}, target: {target}" , exc_info=True)
```

```
100%|██████████| 25000/25000 [01:32<00:00, 268.82it/s]
```

Now we can zip the preprocessed files and store them somewhere.

Module 6 Assignment

You can find the first assignment here: assignment 6

In [ ]: