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

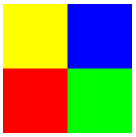
You can use Pillow to create an image from a 3D NumPy cube-shaped array. The rows and columns specify the pixels. The third dimension (size 3) defines red, green, and blue color values. The following code demonstrates creating a simple image from a NumPy array.

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.

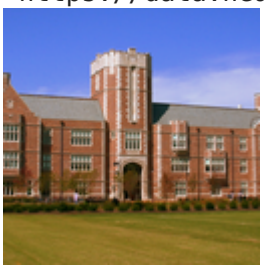
In [ ]:

```
%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

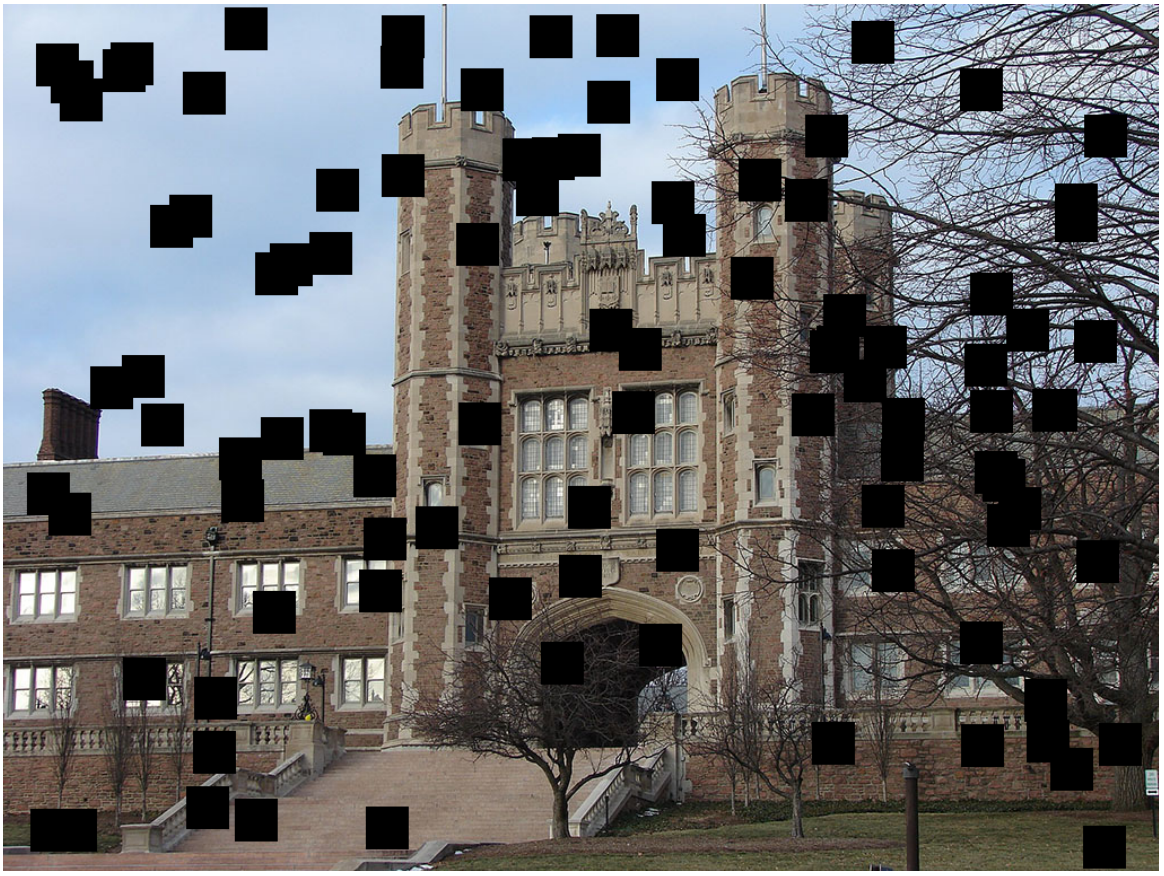
squares to the image to produce noise.

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 [ ]:

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 [1]:
```
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 6.2: Keras Neural Networks for Digits and Fashion MNIST

This module will focus on computer vision. There are some important differences and similarities with previous neural networks.

- We will usually use classification, though regression is still an option.
- The input to the neural network is now 3D (height, width, color)
- Data are not transformed; no z-scores or dummy variables.
- Processing time is much longer.
- We now have different layer times: dense layers (just like before), convolution layers, and max-pooling layers.
- Data will no longer arrive as CSV files. TensorFlow provides some utilities for going directly from the image to the input for a neural network.

Common Computer Vision Data Sets

There are many data sets for computer vision. Two of the most popular classic datasets are the MNIST digits data set and the CIFAR image data sets. We will not use either of these datasets in this course, but it is important to be familiar with them since neural network texts often refer to them.

The MNIST Digits Data Set is very popular in the neural network research community. You can see a sample of it in Figure 6.MNIST.

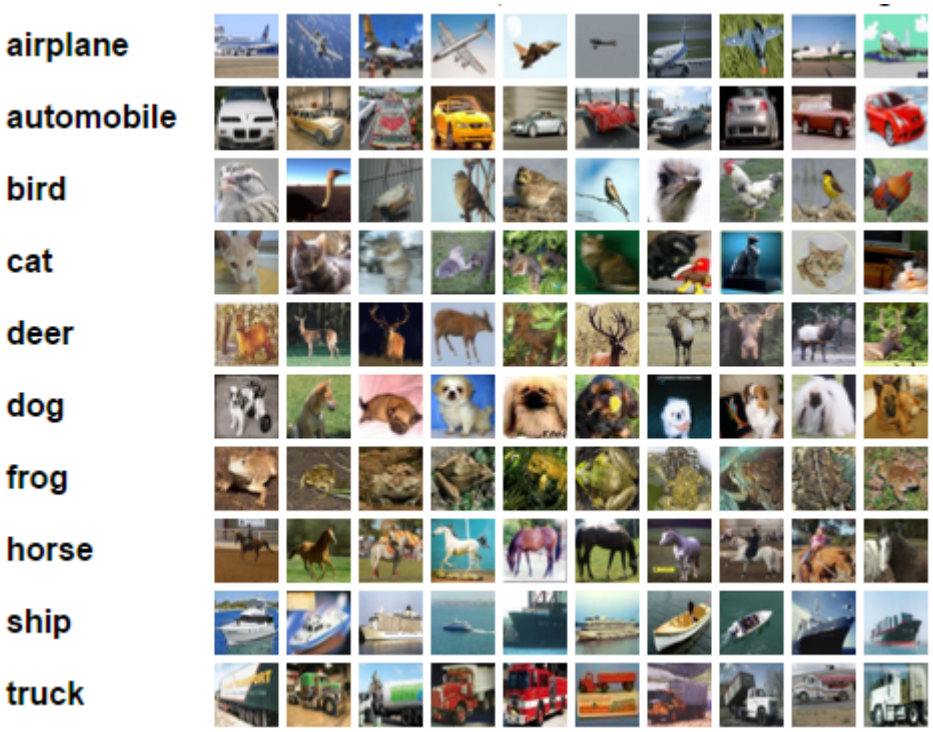
Figure 6.MNIST: MNIST Data Set

Fashion-MNIST is a dataset of Zalando 's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image associated with a label from 10 classes. Fashion-MNIST is a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits. You can see this data in Figure 6.MNIST-FASHION.

Figure 6.MNIST-FASHION: MNIST Fashon Data Set

The CIFAR-10 and CIFAR-100 datasets are also frequently used by the neural network research community.

Figure 6.CIFAR: CIFAR Data Set



The CIFAR-10 data set contains low-rez images that are divided into 10 classes. The CIFAR-100 data set contains 100 classes in a hierarchy.

Convolutional Neural Networks (CNNs)

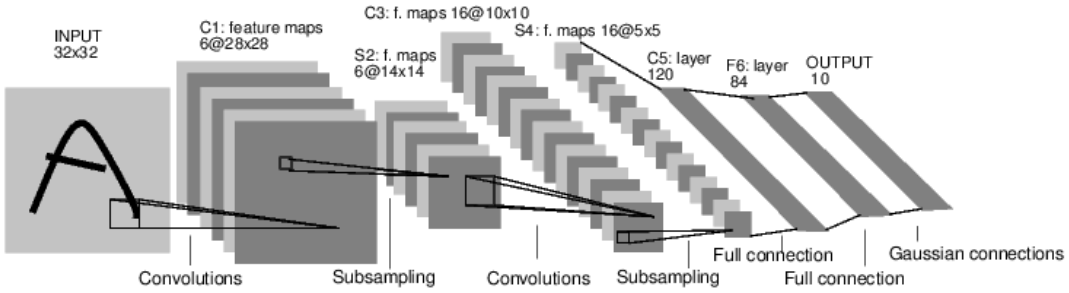
The convolutional neural network (CNN) is a neural network technology that has profoundly impacted the area of computer vision (CV). Fukushima (1980) [Cite:fukushima1980neocognitron] introduced the original concept of a convolutional neural network, and LeCun, Bottou, Bengio & Haffner (1998) [Cite:lecun1995convolutional] greatly improved this work. From this research, Yan LeCun introduced the famous LeNet-5 neural network architecture. This chapter follows the LeNet-5 style of convolutional neural network.
Although computer vision primarily uses CNNs, this technology has some applications outside of the field. You need to realize that if you want to utilize CNNs on non-visual data, you must find a way to encode your data to mimic the properties of visual data.

The order of the input array elements is crucial to the training. In contrast, most neural networks that are not CNNs treat their input data as a long vector of values, and the order in which you arrange the incoming features in this vector is irrelevant. You cannot change the order for these types of neural networks after you have trained the network.

The CNN network arranges the inputs into a grid. This arrangement worked well with images because the pixels in closer proximity to each other are important to each other. The order of pixels in an image is significant. The human body is a relevant example of this type of order. For the design of the face, we are accustomed to eyes being near to each other.

This advance in CNNs is due to years of research on biological eyes. In other words, CNNs utilize overlapping fields of input to simulate features of biological eyes. Until this breakthrough, AI had been unable to reproduce the capabilities of biological vision. Scale, rotation, and noise have presented challenges for AI computer vision research. You can observe the complexity of biological eyes in the example that follows. A friend raises a sheet of paper with a large number written on it. As your friend moves nearer to you, the number is still identifiable. In the same way, you can still identify the number when your friend rotates the paper. Lastly, your friend creates noise by drawing lines on the page, but you can still identify the number. As you can see, these examples demonstrate the high function of the biological eye and allow you to understand better the research breakthrough of CNNs. That is, this neural network can process scale, rotation, and noise in the field of computer vision. You can see this network structure in Figure 6.LENET.

Figure 6.LENET: A LeNET-5 Network (LeCun, 1998)



So far, we have only seen one layer type (dense layers). By the end of this book we will have seen:

- **Dense Layers** - Fully connected layers.
- **Convolution Layers** - Used to scan across images.
- **Max Pooling Layers** - Used to downsample images.
- **Dropout Layers** - Used to add regularization.
- **LSTM and Transformer Layers** - Used for time series data.

Convolution Layers

The first layer that we will examine is the convolutional layer. We will begin by looking at the hyper-parameters that you must specify for a convolutional layer in most neural network frameworks that support the CNN:

- Number of filters
- Filter Size
- Stride
- Padding
- Activation Function/Non-Linearity

The primary purpose of a convolutional layer is to detect features such as edges, lines, blobs of color, and other visual elements. The filters can detect these features. The more filters we give to a convolutional layer, the more features it can see.

A filter is a square-shaped object that scans over the image. A grid can represent the individual pixels of a grid. You can think of the convolutional layer as a smaller grid that sweeps left to right over each image row. There is also a hyperparameter that specifies both the width and height of the square-shaped filter. The following figure shows this configuration in which you see the six convolutional filters sweeping over the image grid:

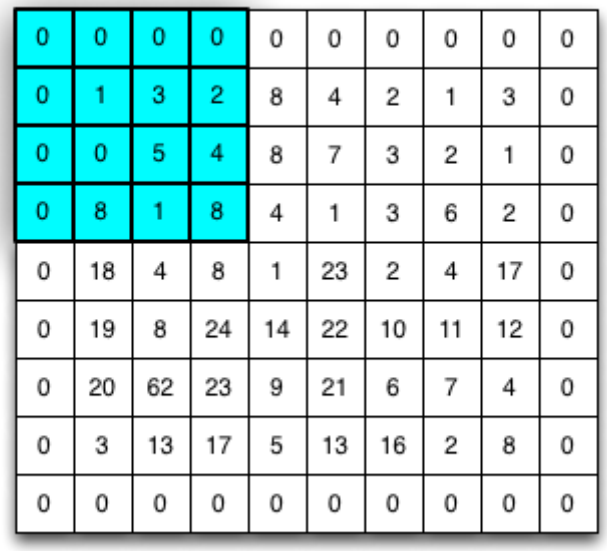
A convolutional layer has weights between it and the previous layer or image grid. Each pixel on each convolutional layer is a weight. Therefore, the number of weights between a convolutional layer and its predecessor layer or image field is the following:

```
[FilterSize] * [FilterSize] * [# of Filters]
```

For example, if the filter size were 5 (5x5) for 10 filters, there would be 250 weights.

You need to understand how the convolutional filters sweep across the previous layer's output or image grid. Figure 6.CNN illustrates the sweep:

Figure 6.CNN: Convolutional Neural Network



The above figure shows a convolutional filter with 4 and a padding size of 1. The padding size is responsible for the border of zeros in the area that the filter sweeps. Even though the image is 8x7, the extra padding provides a virtual image size of 9x8 for the filter to sweep across. The stride specifies the number of positions the convolutional filters will stop. The convolutional filters move to the right, advancing by the number of cells specified in the stride. Once you reach the far right, the convolutional filter moves back to the far left; then, it moves down by the stride amount and continues to the right again.

Some constraints exist concerning the size of the stride. The stride cannot be 0. The convolutional filter would never move if you set the stride. Furthermore, neither the stride nor the convolutional filter size can be larger than the previous grid. There

are additional constraints on the stride (*s*), padding (*p*), and the filter width (*f*) for an image of width (*w*). Specifically, the convolutional filter must be able to start at the far left or top border, move a certain number of strides, and land on the far right or bottom border. The following equation shows the number of steps a convolutional operator must take to cross the image:

$$steps = \frac{w - f + 2p}{s} + 1$$

The number of steps must be an integer. In other words, it cannot have decimal places. The purpose of the padding (*p*) is to be adjusted to make this equation become an integer value.

Max Pooling Layers

Max-pool layers downsample a 3D box to a new one with smaller dimensions. Typically, you can always place a max-pool layer immediately following the convolutional layer. The LENET shows the max-pool layer immediately after layers C1 and C3. These max-pool layers progressively decrease the size of the dimensions of the 3D boxes passing through them. This technique can avoid overfitting (Krizhevsky, Sutskever & Hinton, 2012).

A pooling layer has the following hyper-parameters:

- Spatial Extent (*f*)
- Stride (*s*)

Unlike convolutional layers, max-pool layers do not use padding. Additionally, max-pool layers have no weights, so training does not affect them. These layers downsample their 3D box input. The 3D box output by a max-pool layer will have a width equal to this equation:

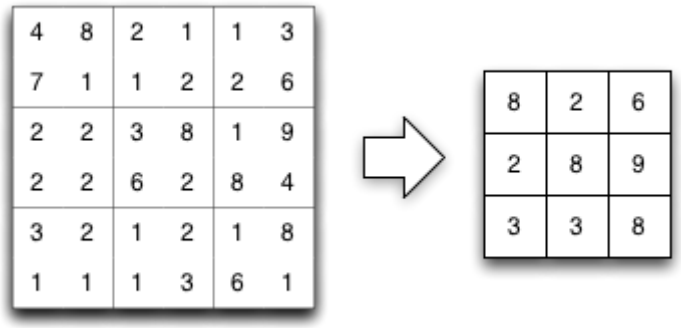
$$w_2 = \frac{w_1 - f}{s} + 1$$

The height of the 3D box produced by the max-pool layer is calculated similarly with this equation:

$$h_2 = \frac{h_1 - f}{s} + 1$$

The depth of the 3D box produced by the max-pool layer is equal to the depth the 3D box received as input. The most common setting for the hyper-parameters of a max-pool layer is f=2 and s=2. The spatial extent (f) specifies that boxes of 2x2 will be scaled down to single pixels. Of these four pixels, the pixel with the maximum value will represent the 2x2 pixel in the new grid. Because squares of size 4 are replaced with size 1, 75% of the pixel information is lost. The following figure shows this transformation as a 6x6 grid becomes a 3x3:

Figure 6.MAXPOOL: Max Pooling Layer



Of course, the above diagram shows each pixel as a single number. A grayscale image would have this characteristic. We usually take the average of the three numbers for an RGB image to determine which pixel has the maximum value.

Regression Convolutional Neural Networks

We will now look at two examples, one for regression and another for classification. For supervised computer vision, your dataset will need some labels. For classification, this label usually specifies what the image is a picture of. For regression, this "label" is some numeric quantity the image should produce, such as a count. We will look at two different means of providing this label.

The first example will show how to handle regression with convolution neural networks. We will provide an image and expect the neural network to count items in that image. We will use a dataset that I created that contains a random number of paperclips. The following code will download this dataset for you.

In [2]:
```
import os

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

if COLAB:
  PATH = "/content"
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(EXTRACT_TARGET, "paperclips")
```

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 [3]:
```
# 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
```

```
--2022-03-01 22:45:29--  https://github.com/jeffheaton/data-mirror/release
s/download/v1/paperclips.zip
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|: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%2F20220301%2Fus-e
ast-1%2Fs3%2Faws4_request&X-Amz-Date=20220301T224529Z&X-Amz-Expires=300&X-
Amz-Signature=92d63a15fadf8b244e13610e65457b6628f86d8465cb450a763a00f4e70f
de8f&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]
--2022-03-01 22:45:29--  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
F20220301%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20220301T224529Z&X-Am
z-Expires=300&X-Amz-Signature=92d63a15fadf8b244e13610e65457b6628f86d8465cb
450a763a00f4e70fde8f&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  22.9MB/s    in 6.0
s

2022-03-01 22:45:35 (26.1 MB/s) - '/content/paperclips.zip' saved [1635906
91/163590691]
```

The labels are contained in a CSV file named **train.csv**for regression. This file has just two labels, **id** and **clip_count**. The ID specifies the filename; for example, row id 1 corresponds to the file **clips-1.jpg**. The following code loads the labels for the training set and creates a new column, named **filename**, that contains the filename of each image, based on the **id** column.

In [4]:
```
import pandas as pd

df = pd.read_csv(
    os.path.join(SOURCE,"train.csv"),
    na_values=['NA', '?'])

df['filename']="clips-"+df["id"].astype(str)+".jpg"
```

This results in the following dataframe.

In [5]:
```
df
```

Out[5]:

	id	clip_count	filename
0	30001	11	clips-30001.jpg
1	30002	2	clips-30002.jpg
2	30003	26	clips-30003.jpg
3	30004	41	clips-30004.jpg
4	30005	49	clips-30005.jpg
...	...	...	...
19995	49996	35	clips-49996.jpg
19996	49997	54	clips-49997.jpg
19997	49998	72	clips-49998.jpg
19998	49999	24	clips-49999.jpg
19999	50000	35	clips-50000.jpg

20000 rows × 3 columns

Separate into a training and validation (for early stopping)

In [6]:

```
TRAIN_PCT = 0.9
TRAIN_CUT = int(len(df) * TRAIN_PCT)

df_train = df[0:TRAIN_CUT]
df_validate = df[TRAIN_CUT:]

print(f"Training size: {len(df_train)}")
print(f"Validate size: {len(df_validate)}")
```

```
Training size: 18000
Validate size: 2000
```

We are now ready to create two ImageDataGenerator objects. We currently use a generator, which creates additional training data by manipulating the source material. This technique can produce considerably stronger neural networks. The generator below flips the images both vertically and horizontally. Keras will train the neuron network both on the original images and the flipped images. This augmentation increases the size of the training data considerably. Module 6.4 goes deeper into the transformations you can perform. You can also specify a target size to resize the images automatically.

The function **flow_from_dataframe** loads the labels from a Pandas dataframe connected to our **train.csv** file. When we demonstrate classification, we will use the **flow_from_directory**; which loads the labels from the directory structure rather than a CSV.

In [7]:

```
import tensorflow as tf
import keras_preprocessing
```

```
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator

training_datagen = ImageDataGenerator(
  rescale = 1./255,
  horizontal_flip=True,
  vertical_flip=True,
  fill_mode='nearest')

train_generator = training_datagen.flow_from_dataframe(
        dataframe=df_train,
        directory=SOURCE,
        x_col="filename",
        y_col="clip_count",
        target_size=(256, 256),
        batch_size=32,
        class_mode='other')

validation_datagen = ImageDataGenerator(rescale = 1./255)

val_generator = validation_datagen.flow_from_dataframe(
        dataframe=df_validate,
        directory=SOURCE,
        x_col="filename",
        y_col="clip_count",
        target_size=(256, 256),
        class_mode='other')
```

```
Found 18000 validated image filenames.
Found 2000 validated image filenames.
```

We can now train the neural network. The code to build and train the neural network is not that different than in the previous modules. We will use the Keras Sequential class to provide layers to the neural network. We now have several new layer types that we did not previously see.

- **Conv2D** - The convolution layers.
- **MaxPooling2D** - The max-pooling layers.
- **Flatten** - Flatten the 2D (and higher) tensors to allow a Dense layer to process.
- **Dense** - Dense layers, the same as demonstrated previously. Dense layers often form the final output layers of the neural network.

The training code is very similar to previously. This code is for regression, so a final linear activation is used, along with mean_squared_error for the loss function. The generator provides both the *x* and *y* matrixes we previously supplied.

In [8]:

```
from tensorflow.keras.callbacks import EarlyStopping
import time

model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 150x150
    # with 3 bytes color.
    # This is the first convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu',
        input_shape=(256, 256, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # The second convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
```

```
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='linear')
])


model.summary()
epoch_steps = 250 # needed for 2.2
validation_steps = len(df_validate)
model.compile(loss = 'mean_squared_error', optimizer='adam')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
        patience=5, verbose=1, mode='auto',
        restore_best_weights=True)

start_time = time.time()
history = model.fit(train_generator,
  verbose = 1,
  validation_data=val_generator, callbacks=[monitor], epochs=25)

elapsed_time = time.time() - start_time
print("Elapsed time: {}".format(hms_string(elapsed_time)))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 64)      1792

 max_pooling2d (MaxPooling2D  (None, 127, 127, 64)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      36928

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 flatten (Flatten)           (None, 246016)            0

 dense (Dense)               (None, 512)               125960704

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 125,999,937
Trainable params: 125,999,937
Non-trainable params: 0
_________________________________________________________________
Epoch 1/25
563/563 [==============================] - 64s 96ms/step - loss: 193.3214
- val_loss: 25.4486
Epoch 2/25
563/563 [==============================] - 52s 92ms/step - loss: 25.5836 -
val_loss: 13.8235
Epoch 3/25
563/563 [==============================] - 53s 93ms/step - loss: 18.8956 -
val_loss: 12.4469
Epoch 4/25
563/563 [==============================] - 52s 92ms/step - loss: 18.8489 -
val_loss: 20.0634
Epoch 5/25
563/563 [==============================] - 52s 92ms/step - loss: 16.5164 -
val_loss: 17.8989
Epoch 6/25
563/563 [==============================] - 52s 91ms/step - loss: 15.5483 -
val_loss: 16.3132
Epoch 7/25
563/563 [==============================] - 52s 92ms/step - loss: 15.6795 -
val_loss: 16.9717
Epoch 8/25
563/563 [==============================] - 52s 92ms/step - loss: 11.5606 -
val_loss: 12.1518
Epoch 9/25
563/563 [==============================] - 52s 92ms/step - loss: 26.0139 -
val_loss: 34.7480
Epoch 10/25
563/563 [==============================] - 52s 93ms/step - loss: 13.2884 -
val_loss: 12.1712
Epoch 11/25
563/563 [==============================] - 52s 93ms/step - loss: 10.7682 -
val_loss: 12.7467
Epoch 12/25
563/563 [==============================] - 53s 94ms/step - loss: 9.8051 -
val_loss: 9.8815
Epoch 13/25
563/563 [==============================] - 62s 109ms/step - loss: 8.2021 -
```

```
val_loss: 8.1277
Epoch 14/25
563/563 [==============================] - 52s 93ms/step - loss: 7.0807 -
val_loss: 6.8239
Epoch 15/25
563/563 [==============================] - 52s 93ms/step - loss: 6.6521 -
val_loss: 7.0292
Epoch 16/25
563/563 [==============================] - 52s 92ms/step - loss: 5.2696 -
val_loss: 6.5994
Epoch 17/25
563/563 [==============================] - 52s 93ms/step - loss: 5.1749 -
val_loss: 12.0623
Epoch 18/25
563/563 [==============================] - 52s 92ms/step - loss: 27.0990 -
val_loss: 15.1000
Epoch 19/25
563/563 [==============================] - 52s 92ms/step - loss: 10.3702 -
val_loss: 6.4094
Epoch 20/25
563/563 [==============================] - 54s 96ms/step - loss: 5.1338 -
val_loss: 5.1066
Epoch 21/25
563/563 [==============================] - 60s 107ms/step - loss: 4.1413 -
val_loss: 6.3927
Epoch 22/25
563/563 [==============================] - 55s 97ms/step - loss: 3.8659 -
val_loss: 4.6390
Epoch 23/25
563/563 [==============================] - 55s 98ms/step - loss: 3.4385 -
val_loss: 4.1845
Epoch 24/25
563/563 [==============================] - 54s 95ms/step - loss: 3.2399 -
val_loss: 4.0449
Epoch 25/25
563/563 [==============================] - 53s 94ms/step - loss: 3.2823 -
val_loss: 4.4899
Elapsed time: 0:22:22.78
```

This code will run very slowly if you do not use a GPU. The above code takes approximately 13 minutes with a GPU.

Score Regression Image Data

Scoring/predicting from a generator is a bit different than training. We do not want augmented images, and we do not wish to have the dataset shuffled. For scoring, we want a prediction for each input. We construct the generator as follows:

- shuffle=False
- batch_size=1
- class_mode=None

We use a **batch_size** of 1 to guarantee that we do not run out of GPU memory if our prediction set is large. You can increase this value for better performance. The **class_mode** is None because there is no *y*, or label. After all, we are predicting.

In [9]:
```
df_test = pd.read_csv(
```

```
    os.path.join(SOURCE,"test.csv"),
    na_values=['NA', '?'])

df_test['filename']="clips-"+df_test["id"].astype(str)+".jpg"

test_datagen = ImageDataGenerator(rescale = 1./255)

test_generator = validation_datagen.flow_from_dataframe(
        dataframe=df_test,
        directory=SOURCE,
        x_col="filename",
        batch_size=1,
        shuffle=False,
        target_size=(256, 256),
        class_mode=None)
```

```
Found 5000 validated image filenames.
```

We need to reset the generator to ensure we are always at the beginning.

In [10]:
```
test_generator.reset()
pred = model.predict(test_generator,steps=len(df_test))
```

We can now generate a CSV file to hold the predictions.

In [11]:
```
df_submit = pd.DataFrame({'id':df_test['id'],'clip_count':pred.flatten()}
df_submit.to_csv(os.path.join(PATH,"submit.csv"),index=False)
```

Classification Neural Networks

Just like earlier in this module, we will load data. However, this time we will use a dataset of images of three different types of the iris flower. This zip file contains three different directories that specify each image's label. The directories are named the same as the labels:

- iris-setosa
- iris-versicolour
- iris-virginica

In [12]:
```
import os

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

if COLAB:
  PATH = "/content"
  EXTRACT_TARGET = os.path.join(PATH,"iris")
  SOURCE = EXTRACT_TARGET # In this case its the same, no subfolder
else:
  # I used this locally on my machine, you may need different
  PATH = "/Users/jeff/temp"
  EXTRACT_TARGET = os.path.join(PATH,"iris")
  SOURCE = EXTRACT_TARGET # In this case its the same, no subfolder
```

Just as before, we unzip the images.

In [13]:
```
# HIDE OUTPUT
!wget -O {os.path.join(PATH,DOWNLOAD_NAME)} {DOWNLOAD_SOURCE}
!mkdir -p {SOURCE}
!mkdir -p {TARGET}
!mkdir -p {EXTRACT_TARGET}
!unzip -o -d {EXTRACT_TARGET} {os.path.join(PATH, DOWNLOAD_NAME)} >/dev/
```

```
--2022-03-01 23:08:29--  https://github.com/jeffheaton/data-mirror/release
s/download/v1/iris-image.zip
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-
asset-2e65be/408419764/d548babd-36c3-414e-add2-a5d9ab941e6e?X-Amz-Algorith
m=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20220301%2Fus-e
ast-1%2Fs3%2Faws4_request&X-Amz-Date=20220301T230829Z&X-Amz-Expires=300&X-
Amz-Signature=f9315f39373d1b8e6ae60a618db5da58714c8bab017e0f55b38c7c0a55d2
cb20&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=408419764&respon
se-content-disposition=attachment%3B%20filename%3Diris-image.zip&response-
content-type=application%2Foctet-stream [following]
--2022-03-01 23:08:30--  https://objects.githubusercontent.com/github-prod
uction-release-asset-2e65be/408419764/d548babd-36c3-414e-add2-a5d9ab941e6
e?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2
F20220301%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20220301T230829Z&X-Am
z-Expires=300&X-Amz-Signature=f9315f39373d1b8e6ae60a618db5da58714c8bab017e
0f55b38c7c0a55d2cb20&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=
408419764&response-content-disposition=attachment%3B%20filename%3Diris-ima
ge.zip&response-content-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)...
185.199.109.133, 185.199.110.133, 185.199.108.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.co
m)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5587253 (5.3M) [application/octet-stream]
Saving to: '/content/iris-image.zip'

/content/iris-image 100%[===================>]   5.33M  6.65MB/s    in 0.8
s

2022-03-01 23:08:31 (6.65 MB/s) - '/content/iris-image.zip' saved [558725
3/5587253]
```

You can see these folders with the following command.

In [14]:
```
!ls /content/iris
```

```
iris-setosa  iris-versicolour  iris-virginica
```

We set up the generator, similar to before. This time we use flow_from_directory to get the labels from the directory structure.

In [15]:
```
import tensorflow as tf
import keras_preprocessing
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator

training_datagen = ImageDataGenerator(
  rescale = 1./255,
```

```
  horizontal_flip=True,
  vertical_flip=True,
  width_shift_range=[-200,200],
  rotation_range=360,

  fill_mode='nearest')

train_generator = training_datagen.flow_from_directory(
    directory=SOURCE, target_size=(256, 256),
    class_mode='categorical', batch_size=32, shuffle=True)

validation_datagen = ImageDataGenerator(rescale = 1./255)

validation_generator = validation_datagen.flow_from_directory(
    directory=SOURCE, target_size=(256, 256),
    class_mode='categorical', batch_size=32, shuffle=True)
```

```
Found 421 images belonging to 3 classes.
Found 421 images belonging to 3 classes.
```

Training the neural network with classification is similar to regression.

In [16]:

```
from tensorflow.keras.callbacks import EarlyStopping

class_count = len(train_generator.class_indices)

model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image
    # 300x300 with 3 bytes color
    # This is the first convolution
    tf.keras.layers.Conv2D(16, (3,3), activation='relu',
        input_shape=(256, 256, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # The second convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.MaxPooling2D(2,2),
    # The third convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fourth convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fifth convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    # Only 1 output neuron. It will contain a value from 0-1
    tf.keras.layers.Dense(class_count, activation='softmax')
])

model.summary()

model.compile(loss = 'categorical_crossentropy', optimizer='adam')

model.fit(train_generator, epochs=50, steps_per_epoch=10,
```

```
                    verbose = 1)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)            (None, 254, 254, 16)      448

 max_pooling2d_2 (MaxPooling  (None, 127, 127, 16)     0
 2D)

 conv2d_3 (Conv2D)            (None, 125, 125, 32)      4640

 dropout (Dropout)            (None, 125, 125, 32)      0

 max_pooling2d_3 (MaxPooling  (None, 62, 62, 32)       0
 2D)

 conv2d_4 (Conv2D)            (None, 60, 60, 64)        18496

 dropout_1 (Dropout)          (None, 60, 60, 64)        0

 max_pooling2d_4 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_5 (Conv2D)            (None, 28, 28, 64)        36928

 max_pooling2d_5 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_6 (Conv2D)            (None, 12, 12, 64)        36928

 max_pooling2d_6 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 flatten_1 (Flatten)          (None, 2304)              0

 dropout_2 (Dropout)          (None, 2304)              0

 dense_2 (Dense)              (None, 512)               1180160

 dense_3 (Dense)              (None, 3)                 1539

=================================================================
Total params: 1,279,139
Trainable params: 1,279,139
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
10/10 [==============================] - 6s 486ms/step - loss: 1.0254
Epoch 2/50
10/10 [==============================] - 5s 472ms/step - loss: 0.9060
Epoch 3/50
10/10 [==============================] - 5s 474ms/step - loss: 0.9712
Epoch 4/50
10/10 [==============================] - 5s 520ms/step - loss: 0.9099
Epoch 5/50
10/10 [==============================] - 5s 517ms/step - loss: 0.9061
Epoch 6/50
10/10 [==============================] - 5s 510ms/step - loss: 0.8965
Epoch 7/50
10/10 [==============================] - 5s 458ms/step - loss: 0.8909
Epoch 8/50
10/10 [==============================] - 5s 514ms/step - loss: 0.8941
Epoch 9/50
```

```
10/10 [==============================] - 5s 493ms/step - loss: 0.9248
Epoch 10/50
10/10 [==============================] - 5s 500ms/step - loss: 0.8780
Epoch 11/50
10/10 [==============================] - 5s 453ms/step - loss: 0.8724
Epoch 12/50
10/10 [==============================] - 5s 448ms/step - loss: 0.8901
Epoch 13/50
10/10 [==============================] - 5s 456ms/step - loss: 0.8817
Epoch 14/50
10/10 [==============================] - 5s 465ms/step - loss: 0.9040
Epoch 15/50
10/10 [==============================] - 5s 449ms/step - loss: 0.8779
Epoch 16/50
10/10 [==============================] - 4s 441ms/step - loss: 0.8479
Epoch 17/50
10/10 [==============================] - 5s 499ms/step - loss: 0.8713
Epoch 18/50
10/10 [==============================] - 5s 456ms/step - loss: 0.8432
Epoch 19/50
10/10 [==============================] - 4s 444ms/step - loss: 0.8816
Epoch 20/50
10/10 [==============================] - 5s 508ms/step - loss: 0.8791
Epoch 21/50
10/10 [==============================] - 5s 497ms/step - loss: 0.8553
Epoch 22/50
10/10 [==============================] - 5s 448ms/step - loss: 0.8275
Epoch 23/50
10/10 [==============================] - 5s 502ms/step - loss: 0.8216
Epoch 24/50
10/10 [==============================] - 5s 456ms/step - loss: 0.8739
Epoch 25/50
10/10 [==============================] - 5s 510ms/step - loss: 0.8650
Epoch 26/50
10/10 [==============================] - 5s 456ms/step - loss: 0.8405
Epoch 27/50
10/10 [==============================] - 5s 456ms/step - loss: 0.8729
Epoch 28/50
10/10 [==============================] - 5s 499ms/step - loss: 0.8618
Epoch 29/50
10/10 [==============================] - 5s 500ms/step - loss: 0.8125
Epoch 30/50
10/10 [==============================] - 5s 504ms/step - loss: 0.8813
Epoch 31/50
10/10 [==============================] - 5s 508ms/step - loss: 0.8392
Epoch 32/50
10/10 [==============================] - 5s 449ms/step - loss: 0.8377
Epoch 33/50
10/10 [==============================] - 5s 499ms/step - loss: 0.8509
Epoch 34/50
10/10 [==============================] - 5s 454ms/step - loss: 0.8647
Epoch 35/50
10/10 [==============================] - 5s 466ms/step - loss: 0.8874
Epoch 36/50
10/10 [==============================] - 5s 502ms/step - loss: 0.9221
Epoch 37/50
10/10 [==============================] - 5s 511ms/step - loss: 0.9186
Epoch 38/50
10/10 [==============================] - 5s 496ms/step - loss: 0.8549
Epoch 39/50
10/10 [==============================] - 5s 493ms/step - loss: 0.9194
Epoch 40/50
10/10 [==============================] - 5s 496ms/step - loss: 0.8528
```

```
Epoch 41/50
10/10 [==============================] - 5s 453ms/step - loss: 0.9105
Epoch 42/50
10/10 [==============================] - 5s 454ms/step - loss: 0.8462
Epoch 43/50
10/10 [==============================] - 5s 459ms/step - loss: 0.8858
Epoch 44/50
10/10 [==============================] - 5s 497ms/step - loss: 0.9119
Epoch 45/50
10/10 [==============================] - 5s 458ms/step - loss: 0.8799
Epoch 46/50
10/10 [==============================] - 5s 499ms/step - loss: 0.8582
Epoch 47/50
10/10 [==============================] - 5s 493ms/step - loss: 0.8536
Epoch 48/50
10/10 [==============================] - 5s 490ms/step - loss: 0.8669
Epoch 49/50
10/10 [==============================] - 5s 458ms/step - loss: 0.7957
Epoch 50/50
10/10 [==============================] - 5s 501ms/step - loss: 0.8670
```

The iris image dataset is not easy to predict; it turns out that a tabular dataset of measurements is more manageable. However, we can achieve a 63%.

In [22]:

```
from sklearn.metrics import accuracy_score
import numpy as np

validation_generator.reset()
pred = model.predict(validation_generator)

predict_classes = np.argmax(pred,axis=1)
expected_classes = validation_generator.classes

correct = accuracy_score(expected_classes,predict_classes)
print(f"Accuracy: {correct}")
```

```
Accuracy: 0.6389548693586699
```

Other Resources

- Imagenet:Large Scale Visual Recognition Challenge 2014
- Andrej Karpathy - PhD student/instructor at Stanford.
- CS231n Convolutional Neural Networks for Visual Recognition - Stanford course on computer vision/CNN's.
- CS231n - GitHub
- ConvNetJS - JavaScript library for deep learning.

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 [1]:
```
# Detect Colab if present
try:
    from google.colab import drive
    COLAB = True
    print("Note: using Google CoLab")
    %tensorflow_version 2.x
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 6.3: Transfer Learning for

Computer Vision

Many advanced prebuilt neural networks are available for computer vision, and Keras provides direct access to many networks. Transfer learning is the technique where you use these prebuilt neural networks. Module 9 takes a deeper look at transfer learning.

There are several different levels of transfer learning.

- Use a prebuilt neural network in its entirety
- Use a prebuilt neural network's structure
- Use a prebuilt neural network's weights

We will begin by using the MobileNet prebuilt neural network in its entirety. MobileNet will be loaded and allowed to classify simple images. We can already classify 1,000 images through this technique without ever having trained the network.

In [2]:
```
import pandas as pd
import numpy as np
import os
import tensorflow.keras
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense,GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

We begin by downloading weights for a MobileNet trained for the imagenet dataset, which will take some time to download the first time you train the network.

In [3]:
```
# HIDE OUTPUT
model = MobileNet(weights='imagenet',include_top=True)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-appl
ications/mobilenet/mobilenet_1_0_224_tf.h5
17227776/17225924 [==============================] - 0s 0us/step
17235968/17225924 [==============================] - 0s 0us/step
```

The loaded network is a Keras neural network. However, this is a neural network that a third party engineered on advanced hardware. Merely looking at the structure of an advanced state-of-the-art neural network can be educational.

In [4]:
```
model.summary()
```

```
Model: "mobilenet_1.00_224"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv1 (Conv2D)              (None, 112, 112, 32)      864

 conv1_bn (BatchNormalizatio  (None, 112, 112, 32)     128
 n)

 conv1_relu (ReLU)           (None, 112, 112, 32)      0

 conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)     288

 conv_dw_1_bn (BatchNormaliz  (None, 112, 112, 32)     128
 ation)

 conv_dw_1_relu (ReLU)       (None, 112, 112, 32)      0

 conv_pw_1 (Conv2D)          (None, 112, 112, 64)      2048

 conv_pw_1_bn (BatchNormaliz  (None, 112, 112, 64)     256
 ation)

 conv_pw_1_relu (ReLU)       (None, 112, 112, 64)      0

 conv_pad_2 (ZeroPadding2D)  (None, 113, 113, 64)      0

 conv_dw_2 (DepthwiseConv2D)  (None, 56, 56, 64)       576

 conv_dw_2_bn (BatchNormaliz  (None, 56, 56, 64)       256
 ation)

 conv_dw_2_relu (ReLU)       (None, 56, 56, 64)        0

 conv_pw_2 (Conv2D)          (None, 56, 56, 128)       8192

 conv_pw_2_bn (BatchNormaliz  (None, 56, 56, 128)      512
 ation)

 conv_pw_2_relu (ReLU)       (None, 56, 56, 128)       0

 conv_dw_3 (DepthwiseConv2D)  (None, 56, 56, 128)      1152

 conv_dw_3_bn (BatchNormaliz  (None, 56, 56, 128)      512
 ation)

 conv_dw_3_relu (ReLU)       (None, 56, 56, 128)       0

 conv_pw_3 (Conv2D)          (None, 56, 56, 128)       16384

 conv_pw_3_bn (BatchNormaliz  (None, 56, 56, 128)      512
 ation)

 conv_pw_3_relu (ReLU)       (None, 56, 56, 128)       0

 conv_pad_4 (ZeroPadding2D)  (None, 57, 57, 128)       0

 conv_dw_4 (DepthwiseConv2D)  (None, 28, 28, 128)      1152

 conv_dw_4_bn (BatchNormaliz  (None, 28, 28, 128)      512
 ation)
```

```
conv_dw_4_relu (ReLU)       (None, 28, 28, 128)       0

conv_pw_4 (Conv2D)          (None, 28, 28, 256)       32768

conv_pw_4_bn (BatchNormaliz  (None, 28, 28, 256)      1024
ation)

conv_pw_4_relu (ReLU)       (None, 28, 28, 256)       0

conv_dw_5 (DepthwiseConv2D)  (None, 28, 28, 256)      2304

conv_dw_5_bn (BatchNormaliz  (None, 28, 28, 256)      1024
ation)

conv_dw_5_relu (ReLU)       (None, 28, 28, 256)       0

conv_pw_5 (Conv2D)          (None, 28, 28, 256)       65536

conv_pw_5_bn (BatchNormaliz  (None, 28, 28, 256)      1024
ation)

conv_pw_5_relu (ReLU)       (None, 28, 28, 256)       0

conv_pad_6 (ZeroPadding2D)  (None, 29, 29, 256)       0

conv_dw_6 (DepthwiseConv2D)  (None, 14, 14, 256)      2304

conv_dw_6_bn (BatchNormaliz  (None, 14, 14, 256)      1024
ation)

conv_dw_6_relu (ReLU)       (None, 14, 14, 256)       0

conv_pw_6 (Conv2D)          (None, 14, 14, 512)       131072

conv_pw_6_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_pw_6_relu (ReLU)       (None, 14, 14, 512)       0

conv_dw_7 (DepthwiseConv2D)  (None, 14, 14, 512)      4608

conv_dw_7_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_dw_7_relu (ReLU)       (None, 14, 14, 512)       0

conv_pw_7 (Conv2D)          (None, 14, 14, 512)       262144

conv_pw_7_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_pw_7_relu (ReLU)       (None, 14, 14, 512)       0

conv_dw_8 (DepthwiseConv2D)  (None, 14, 14, 512)      4608

conv_dw_8_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_dw_8_relu (ReLU)       (None, 14, 14, 512)       0

conv_pw_8 (Conv2D)          (None, 14, 14, 512)       262144
```

```
conv_pw_8_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_pw_8_relu (ReLU)       (None, 14, 14, 512)       0

conv_dw_9 (DepthwiseConv2D)  (None, 14, 14, 512)      4608

conv_dw_9_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_dw_9_relu (ReLU)       (None, 14, 14, 512)       0

conv_pw_9 (Conv2D)          (None, 14, 14, 512)       262144

conv_pw_9_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_pw_9_relu (ReLU)       (None, 14, 14, 512)       0

conv_dw_10 (DepthwiseConv2D  (None, 14, 14, 512)      4608
)

conv_dw_10_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)

conv_dw_10_relu (ReLU)      (None, 14, 14, 512)       0

conv_pw_10 (Conv2D)         (None, 14, 14, 512)       262144

conv_pw_10_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)

conv_pw_10_relu (ReLU)      (None, 14, 14, 512)       0

conv_dw_11 (DepthwiseConv2D  (None, 14, 14, 512)      4608
)

conv_dw_11_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)

conv_dw_11_relu (ReLU)      (None, 14, 14, 512)       0

conv_pw_11 (Conv2D)         (None, 14, 14, 512)       262144

conv_pw_11_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)

conv_pw_11_relu (ReLU)      (None, 14, 14, 512)       0

conv_pad_12 (ZeroPadding2D)  (None, 15, 15, 512)      0

conv_dw_12 (DepthwiseConv2D  (None, 7, 7, 512)        4608
)

conv_dw_12_bn (BatchNormali  (None, 7, 7, 512)        2048
zation)

conv_dw_12_relu (ReLU)      (None, 7, 7, 512)         0

conv_pw_12 (Conv2D)         (None, 7, 7, 1024)        524288

conv_pw_12_bn (BatchNormali  (None, 7, 7, 1024)       4096
zation)
```

```
conv_pw_12_relu (ReLU)       (None, 7, 7, 1024)        0

conv_dw_13 (DepthwiseConv2D  (None, 7, 7, 1024)        9216
)

conv_dw_13_bn (BatchNormali  (None, 7, 7, 1024)        4096
zation)

conv_dw_13_relu (ReLU)       (None, 7, 7, 1024)        0

conv_pw_13 (Conv2D)          (None, 7, 7, 1024)        1048576

conv_pw_13_bn (BatchNormali  (None, 7, 7, 1024)        4096
zation)

conv_pw_13_relu (ReLU)       (None, 7, 7, 1024)        0

global_average_pooling2d (G  (None, 1, 1, 1024)        0
lobalAveragePooling2D)

dropout (Dropout)            (None, 1, 1, 1024)        0

conv_preds (Conv2D)          (None, 1, 1, 1000)        1025000

reshape_2 (Reshape)          (None, 1000)              0

predictions (Activation)     (None, 1000)              0

=================================================================
Total params: 4,253,864
Trainable params: 4,231,976
Non-trainable params: 21,888
_________________________________________________________________
```

Several clues to neural network architecture become evident when examining the above structure.

We will now use the MobileNet to classify several image URLs below. You can add additional URLs of your own to see how well the MobileNet can classify.

In [5]:

```
%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
from tensorflow.keras.applications.mobilenet import decode_predictions

IMAGE_WIDTH = 224
IMAGE_HEIGHT = 224
IMAGE_CHANNELS = 3

ROOT = "https://data.heatonresearch.com/data/t81-558/images/"

def make_square(img):
    cols,rows = img.size

    if rows>cols:
        pad = (rows-cols)/2
```

```
        img = img.crop((pad,0,cols,cols))
    else:
        pad = (cols-rows)/2
        img = img.crop((0,pad,rows,rows))

    return img


def classify_image(url):
  x = []
  ImageFile.LOAD_TRUNCATED_IMAGES = False
  response = requests.get(url)
  img = Image.open(BytesIO(response.content))
  img.load()
  img = img.resize((IMAGE_WIDTH,IMAGE_HEIGHT),Image.ANTIALIAS)

  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)
  x = preprocess_input(x)
  x = x[:,:,:,:3] # maybe an alpha channel
  pred = model.predict(x)

  display(img)
  print(np.argmax(pred,axis=1))

  lst = decode_predictions(pred, top=5)
  for itm in lst[0]:
      print(itm)
```

We can now classify an example image. You can specify the URL of any image you wish to classify.

In [6]:
```
classify_image(ROOT+"soccer_ball.jpg")
```

```
[805]
Downloading data from https://storage.googleapis.com/download.tensorflow.o
rg/data/imagenet_class_index.json
40960/35363 [==================================] - 0s 0us/step
49152/35363 [==========================================] - 0s 0us/step
('n04254680', 'soccer_ball', 0.9999938)
('n03530642', 'honeycomb', 3.862412e-06)
('n03255030', 'dumbbell', 4.442458e-07)
('n02782093', 'balloon', 3.7038987e-07)
('n04548280', 'wall_clock', 3.143911e-07)
```

In [7]:
```
classify_image(ROOT+"race_truck.jpg")
```

Overall, the neural network is doing quite well.

For many applications, MobileNet might be entirely acceptable as an image classifier. However, if you need to classify very specialized images, not in the 1,000 image types supported by imagenet, it is necessary to use transfer learning.

Using the Structure of ResNet

We will train a neural network to count the number of paper clips in images. We will make use of the structure of the ResNet neural network. There are several significant changes that we will make to ResNet to apply to this task. First, ResNet is a classifier; we wish to perform a regression to count. Secondly, we want to change the image resolution that ResNet uses. We will not use the weights from ResNet; changing this resolution invalidates the current weights. Thus, it will be necessary to retrain the network.

In [8]:
```
import os
URL = "https://github.com/jeffheaton/data-mirror/"
DOWNLOAD_SOURCE = URL+"releases/download/v1/paperclips.zip"
DOWNLOAD_NAME = DOWNLOAD_SOURCE[DOWNLOAD_SOURCE.rfind('/')+1:]

if COLAB:
  PATH = "/content"
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(EXTRACT_TARGET, "paperclips")
```

```
[751]
('n04037443', 'racer', 0.7131951)
('n03100240', 'convertible', 0.100896776)
('n04285008', 'sports_car', 0.0770768)
('n03930630', 'pickup', 0.02635305)
('n02704792', 'amphibian', 0.011636169)
```

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 [9]:
```
# 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
```

```
--2021-11-28 08:45:31--  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%2F20211128%2Fus-e
ast-1%2Fs3%2Faws4_request&X-Amz-Date=20211128T084531Z&X-Amz-Expires=300&X-
Amz-Signature=6db9f72de68b167bd44bfec7661073997b48ed04708a827f50189f622b03
95cd&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]
--2021-11-28 08:45:31--  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
F20211128%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20211128T084531Z&X-Am
z-Expires=300&X-Amz-Signature=6db9f72de68b167bd44bfec7661073997b48ed04708a
827f50189f622b0395cd&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  25.5MB/s    in 5.9
s

2021-11-28 08:45:37 (26.6 MB/s) - ‘/content/paperclips.zip’ saved [1635906
91/163590691]
```

The labels are contained in a CSV file named **train.csv** for the regression. This file has just two labels, **id** and **clip_count**. The ID specifies the filename; for example, row id 1 corresponds to the file **clips-1.jpg**. The following code loads the labels for the training set and creates a new column, named **filename**, that contains the filename of each image, based on the **id** column.

In [11]:
```
df_train = pd.read_csv(os.path.join(SOURCE, "train.csv"))
df_train['filename'] = "clips-" + df_train.id.astype(str) + ".jpg"
```

We want to use early stopping. To do this, we need a validation set. We will break the data into 80 percent test data and 20 validation. Do not confuse this validation

data with the test set provided by Kaggle. This validation set is unique to your program and is for early stopping.

In [12]:
```
TRAIN_PCT = 0.9
TRAIN_CUT = int(len(df_train) * TRAIN_PCT)

df_train_cut = df_train[0:TRAIN_CUT]
df_validate_cut = df_train[TRAIN_CUT:]

print(f"Training size: {len(df_train_cut)}")
print(f"Validate size: {len(df_validate_cut)}")
```

```
Training size: 18000
Validate size: 2000
```

Next, we create the generators that will provide the images to the neural network during training. We normalize the images so that the RGB colors between 0-255 become ratios between 0 and 1. We also use the **flow_from_dataframe** generator to connect the Pandas dataframe to the actual image files. We see here a straightforward implementation; you might also wish to use some of the image transformations provided by the data generator.

The **HEIGHT** and **WIDTH** constants specify the dimensions to which the image will be scaled (or expanded). It is probably not a good idea to expand the images.

In [13]:
```
import tensorflow as tf
import keras_preprocessing
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator

WIDTH = 256
HEIGHT = 256

training_datagen = ImageDataGenerator(
  rescale = 1./255,
  horizontal_flip=True,
  #vertical_flip=True,
  fill_mode='nearest')

train_generator = training_datagen.flow_from_dataframe(
        dataframe=df_train_cut,
        directory=SOURCE,
        x_col="filename",
        y_col="clip_count",
        target_size=(HEIGHT, WIDTH),
        # Keeping the training batch size small
        # USUALLY increases performance
        batch_size=32,
        class_mode='raw')

validation_datagen = ImageDataGenerator(rescale = 1./255)

val_generator = validation_datagen.flow_from_dataframe(
        dataframe=df_validate_cut,
        directory=SOURCE,
        x_col="filename",
        y_col="clip_count",
```

```
        target_size=(HEIGHT, WIDTH),
        # Make the validation batch size as large as you
        # have memory for
        batch_size=256,
        class_mode='raw')
```

```
Found 18000 validated image filenames.
Found 2000 validated image filenames.
```

We will now use a ResNet neural network as a basis for our neural network. We will redefine both the input shape and output of the ResNet model, so we will not transfer the weights. Since we redefine the input, the weights are of minimal value. We begin by loading, from Keras, the ResNet50 network. We specify **include_top** as False because we will change the input resolution. We also specify **weights** as false because we must retrain the network after changing the top input layers.

In [14]:

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Input

input_tensor = Input(shape=(HEIGHT, WIDTH, 3))

base_model = ResNet50(
    include_top=False, weights=None, input_tensor=input_tensor,
    input_shape=None)
```

Now we must add a few layers to the end of the neural network so that it becomes a regression model.

In [15]:

```
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model

x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x)
x=Dense(1024,activation='relu')(x)
model=Model(inputs=base_model.input,outputs=Dense(1)(x))
```

We train like before; the only difference is that we do not define the entire neural network here.

In [ ]:

```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.metrics import RootMeanSquaredError

# Important, calculate a valid step size for the validation dataset
STEP_SIZE_VALID=val_generator.n//val_generator.batch_size

model.compile(loss = 'mean_squared_error', optimizer='adam',
              metrics=[RootMeanSquaredError(name="rmse")])
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
        patience=50, verbose=1, mode='auto',
        restore_best_weights=True)

history = model.fit(train_generator, epochs=100, steps_per_epoch=250,
                    validation_data = val_generator, callbacks=[monitor]
                    verbose = 1, validation_steps=STEP_SIZE_VALID)
```

```
Epoch 1/100
250/250 [==============================] - 70s 256ms/step - loss: 73.1411
- rmse: 8.5523 - val_loss: 701.4966 - val_rmse: 26.4858
Epoch 2/100
250/250 [==============================] - 61s 243ms/step - loss: 27.7530
- rmse: 5.2681 - val_loss: 365.0618 - val_rmse: 19.1066
Epoch 3/100
250/250 [==============================] - 61s 243ms/step - loss: 28.6821
- rmse: 5.3556 - val_loss: 130.9240 - val_rmse: 11.4422
Epoch 4/100
250/250 [==============================] - 61s 243ms/step - loss: 18.8626
- rmse: 4.3431 - val_loss: 55.8694 - val_rmse: 7.4746
Epoch 5/100
250/250 [==============================] - 61s 242ms/step - loss: 14.1957
- rmse: 3.7677 - val_loss: 554.3814 - val_rmse: 23.5453
Epoch 6/100
250/250 [==============================] - 61s 242ms/step - loss: 12.8428
- rmse: 3.5837 - val_loss: 79.6855 - val_rmse: 8.9267
Epoch 7/100
250/250 [==============================] - 61s 242ms/step - loss: 13.2751
- rmse: 3.6435 - val_loss: 316.9753 - val_rmse: 17.8038
Epoch 8/100
250/250 [==============================] - 61s 242ms/step - loss: 11.9826
- rmse: 3.4616 - val_loss: 466.4104 - val_rmse: 21.5965
Epoch 9/100
250/250 [==============================] - 61s 243ms/step - loss: 12.0956
- rmse: 3.4779 - val_loss: 4.5767 - val_rmse: 2.1393
Epoch 10/100
250/250 [==============================] - 60s 242ms/step - loss: 9.6629 -
rmse: 3.1085 - val_loss: 82.4498 - val_rmse: 9.0802
Epoch 11/100
250/250 [==============================] - 60s 242ms/step - loss: 6.0348 -
rmse: 2.4566 - val_loss: 134.9830 - val_rmse: 11.6182
Epoch 12/100
250/250 [==============================] - 61s 242ms/step - loss: 9.1004 -
rmse: 3.0167 - val_loss: 13.1667 - val_rmse: 3.6286
Epoch 13/100
250/250 [==============================] - 60s 242ms/step - loss: 9.2808 -
rmse: 3.0464 - val_loss: 372.9783 - val_rmse: 19.3126
Epoch 14/100
250/250 [==============================] - 61s 242ms/step - loss: 5.7128 -
rmse: 2.3901 - val_loss: 26.7188 - val_rmse: 5.1690
Epoch 15/100
250/250 [==============================] - 61s 242ms/step - loss: 5.8171 -
rmse: 2.4119 - val_loss: 15.2567 - val_rmse: 3.9060
Epoch 16/100
250/250 [==============================] - 61s 242ms/step - loss: 5.2777 -
rmse: 2.2973 - val_loss: 61.7677 - val_rmse: 7.8592
Epoch 17/100
250/250 [==============================] - 61s 242ms/step - loss: 8.9798 -
rmse: 2.9966 - val_loss: 116.6043 - val_rmse: 10.7983
Epoch 18/100
250/250 [==============================] - 60s 242ms/step - loss: 6.0367 -
rmse: 2.4570 - val_loss: 11.1855 - val_rmse: 3.3445
Epoch 19/100
250/250 [==============================] - 60s 241ms/step - loss: 7.0206 -
rmse: 2.6496 - val_loss: 157.4581 - val_rmse: 12.5482
Epoch 20/100
250/250 [==============================] - 60s 240ms/step - loss: 7.4522 -
rmse: 2.7299 - val_loss: 210.9105 - val_rmse: 14.5228
Epoch 21/100
250/250 [==============================] - 60s 241ms/step - loss: 10.0948
- rmse: 3.1772 - val_loss: 18.0399 - val_rmse: 4.2473
```

```
Epoch 22/100
250/250 [==============================] - 61s 242ms/step - loss: 5.0645 -
rmse: 2.2505 - val_loss: 21.2375 - val_rmse: 4.6084
Epoch 23/100
250/250 [==============================] - 60s 241ms/step - loss: 5.7177 -
rmse: 2.3912 - val_loss: 28.5047 - val_rmse: 5.3390
Epoch 24/100
250/250 [==============================] - 60s 241ms/step - loss: 5.1064 -
rmse: 2.2597 - val_loss: 47.6090 - val_rmse: 6.8999
Epoch 25/100
250/250 [==============================] - 60s 241ms/step - loss: 4.4656 -
rmse: 2.1132 - val_loss: 51.3690 - val_rmse: 7.1672
Epoch 26/100
250/250 [==============================] - 60s 240ms/step - loss: 3.6463 -
rmse: 1.9095 - val_loss: 93.4837 - val_rmse: 9.6687
Epoch 27/100
250/250 [==============================] - 60s 241ms/step - loss: 3.6216 -
rmse: 1.9031 - val_loss: 49.9860 - val_rmse: 7.0701
Epoch 28/100
250/250 [==============================] - 60s 241ms/step - loss: 3.9304 -
rmse: 1.9825 - val_loss: 4.8757 - val_rmse: 2.2081
Epoch 29/100
250/250 [==============================] - 60s 240ms/step - loss: 4.6160 -
rmse: 2.1485 - val_loss: 159.4939 - val_rmse: 12.6291
Epoch 30/100
250/250 [==============================] - 60s 240ms/step - loss: 5.9745 -
rmse: 2.4443 - val_loss: 31.3900 - val_rmse: 5.6027
Epoch 31/100
250/250 [==============================] - 60s 241ms/step - loss: 4.9073 -
rmse: 2.2152 - val_loss: 44.5920 - val_rmse: 6.6777
Epoch 32/100
250/250 [==============================] - 60s 241ms/step - loss: 4.4296 -
rmse: 2.1047 - val_loss: 6.8120 - val_rmse: 2.6100
Epoch 33/100
250/250 [==============================] - 60s 241ms/step - loss: 6.6059 -
rmse: 2.5702 - val_loss: 103.0320 - val_rmse: 10.1505
Epoch 34/100
250/250 [==============================] - 60s 242ms/step - loss: 3.9264 -
rmse: 1.9815 - val_loss: 318.6042 - val_rmse: 17.8495
Epoch 35/100
250/250 [==============================] - 61s 242ms/step - loss: 3.7293 -
rmse: 1.9311 - val_loss: 245.8616 - val_rmse: 15.6800
Epoch 36/100
250/250 [==============================] - 61s 244ms/step - loss: 3.5809 -
rmse: 1.8923 - val_loss: 3.9251 - val_rmse: 1.9812
Epoch 37/100
250/250 [==============================] - 61s 243ms/step - loss: 3.6419 -
rmse: 1.9084 - val_loss: 23.3965 - val_rmse: 4.8370
Epoch 38/100
250/250 [==============================] - 61s 243ms/step - loss: 3.6437 -
rmse: 1.9089 - val_loss: 22.4549 - val_rmse: 4.7387
Epoch 39/100
250/250 [==============================] - 61s 242ms/step - loss: 3.5197 -
rmse: 1.8761 - val_loss: 103.7435 - val_rmse: 10.1855
Epoch 40/100
250/250 [==============================] - 61s 242ms/step - loss: 5.8539 -
rmse: 2.4195 - val_loss: 272.6473 - val_rmse: 16.5120
Epoch 41/100
250/250 [==============================] - 61s 242ms/step - loss: 2.8808 -
rmse: 1.6973 - val_loss: 97.9878 - val_rmse: 9.8989
Epoch 42/100
250/250 [==============================] - 61s 242ms/step - loss: 3.9501 -
rmse: 1.9875 - val_loss: 237.1111 - val_rmse: 15.3984
```

```
Epoch 43/100
250/250 [==============================] - 61s 242ms/step - loss: 5.9793 -
rmse: 2.4453 - val_loss: 102.9308 - val_rmse: 10.1455
Epoch 44/100
250/250 [==============================] - 61s 242ms/step - loss: 3.2876 -
rmse: 1.8132 - val_loss: 13.3443 - val_rmse: 3.6530
Epoch 45/100
250/250 [==============================] - 61s 243ms/step - loss: 2.8473 -
rmse: 1.6874 - val_loss: 4.4881 - val_rmse: 2.1185
Epoch 46/100
250/250 [==============================] - 61s 243ms/step - loss: 2.9382 -
rmse: 1.7141 - val_loss: 13.9019 - val_rmse: 3.7285
Epoch 47/100
250/250 [==============================] - 61s 242ms/step - loss: 3.5568 -
rmse: 1.8860 - val_loss: 59.7056 - val_rmse: 7.7269
Epoch 48/100
250/250 [==============================] - 61s 243ms/step - loss: 3.0542 -
rmse: 1.7476 - val_loss: 48.3846 - val_rmse: 6.9559
Epoch 49/100
250/250 [==============================] - 61s 242ms/step - loss: 4.0696 -
rmse: 2.0173 - val_loss: 21.7313 - val_rmse: 4.6617
Epoch 50/100
250/250 [==============================] - 61s 242ms/step - loss: 3.7122 -
rmse: 1.9267 - val_loss: 118.0979 - val_rmse: 10.8673
Epoch 51/100
250/250 [==============================] - 61s 242ms/step - loss: 2.6829 -
rmse: 1.6379 - val_loss: 10.7841 - val_rmse: 3.2839
Epoch 52/100
250/250 [==============================] - 61s 243ms/step - loss: 2.8031 -
rmse: 1.6742 - val_loss: 13.8609 - val_rmse: 3.7230
Epoch 53/100
250/250 [==============================] - 61s 242ms/step - loss: 2.7726 -
rmse: 1.6651 - val_loss: 21.6723 - val_rmse: 4.6553
Epoch 54/100
250/250 [==============================] - 61s 243ms/step - loss: 3.4007 -
rmse: 1.8441 - val_loss: 77.9120 - val_rmse: 8.8268
Epoch 55/100
250/250 [==============================] - 61s 243ms/step - loss: 3.0227 -
rmse: 1.7386 - val_loss: 17.7745 - val_rmse: 4.2160
Epoch 56/100
250/250 [==============================] - 61s 243ms/step - loss: 4.1116 -
rmse: 2.0277 - val_loss: 20.5534 - val_rmse: 4.5336
Epoch 57/100
250/250 [==============================] - 61s 243ms/step - loss: 2.5196 -
rmse: 1.5873 - val_loss: 1.6131 - val_rmse: 1.2701
Epoch 58/100
250/250 [==============================] - 61s 243ms/step - loss: 3.0357 -
rmse: 1.7423 - val_loss: 72.6971 - val_rmse: 8.5263
Epoch 59/100
250/250 [==============================] - 61s 243ms/step - loss: 2.4410 -
rmse: 1.5624 - val_loss: 300.2112 - val_rmse: 17.3266
Epoch 60/100
250/250 [==============================] - 61s 242ms/step - loss: 2.2377 -
rmse: 1.4959 - val_loss: 4.8804 - val_rmse: 2.2092
Epoch 61/100
250/250 [==============================] - 61s 243ms/step - loss: 2.5355 -
rmse: 1.5923 - val_loss: 3.1464 - val_rmse: 1.7738
Epoch 62/100
250/250 [==============================] - 61s 243ms/step - loss: 2.4223 -
rmse: 1.5564 - val_loss: 149.8977 - val_rmse: 12.2433
Epoch 63/100
250/250 [==============================] - 61s 243ms/step - loss: 2.3303 -
rmse: 1.5265 - val_loss: 97.8213 - val_rmse: 9.8905
```

```
Epoch 64/100
250/250 [==============================] - 61s 242ms/step - loss: 2.6361 -
rmse: 1.6236 - val_loss: 7.0856 - val_rmse: 2.6619
Epoch 65/100
250/250 [==============================] - 61s 243ms/step - loss: 2.2068 -
rmse: 1.4855 - val_loss: 42.9824 - val_rmse: 6.5561
Epoch 66/100
250/250 [==============================] - 61s 243ms/step - loss: 2.2291 -
rmse: 1.4930 - val_loss: 27.3345 - val_rmse: 5.2282
Epoch 67/100
250/250 [==============================] - 61s 242ms/step - loss: 2.2970 -
rmse: 1.5156 - val_loss: 5.9973 - val_rmse: 2.4489
Epoch 68/100
250/250 [==============================] - 61s 243ms/step - loss: 2.8215 -
rmse: 1.6797 - val_loss: 12.4237 - val_rmse: 3.5247
Epoch 69/100
250/250 [==============================] - 61s 243ms/step - loss: 2.4158 -
rmse: 1.5543 - val_loss: 12.4950 - val_rmse: 3.5348
Epoch 70/100
250/250 [==============================] - 61s 243ms/step - loss: 2.4888 -
rmse: 1.5776 - val_loss: 27.5749 - val_rmse: 5.2512
Epoch 71/100
250/250 [==============================] - 61s 243ms/step - loss: 1.9211 -
rmse: 1.3860 - val_loss: 17.0489 - val_rmse: 4.1290
Epoch 72/100
250/250 [==============================] - 61s 243ms/step - loss: 2.3726 -
rmse: 1.5403 - val_loss: 167.8536 - val_rmse: 12.9558
```

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 [1]:
```
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 6.4: Inside Augmentation

The ImageDataGenerator class provides many options for image augmentation.

Deciding which augmentations to use can impact the effectiveness of your model. This part will visualize some of these augmentations that you might use to train your neural network. We begin by loading a sample image to augment.

In [2]:
```
import urllib.request
import shutil
from IPython.display import Image

URL =  "https://github.com/jeffheaton/t81_558_deep_learning/" +\
  "blob/master/photos/landscape.jpg?raw=true"
LOCAL_IMG_FILE = "/content/landscape.jpg"

with urllib.request.urlopen(URL) as response, \
  open(LOCAL_IMG_FILE, 'wb') as out_file:
    shutil.copyfileobj(response, out_file)

Image(filename=LOCAL_IMG_FILE)
```

Out[2]:

Next, we introduce a simple utility function to visualize four images sampled from any generator.

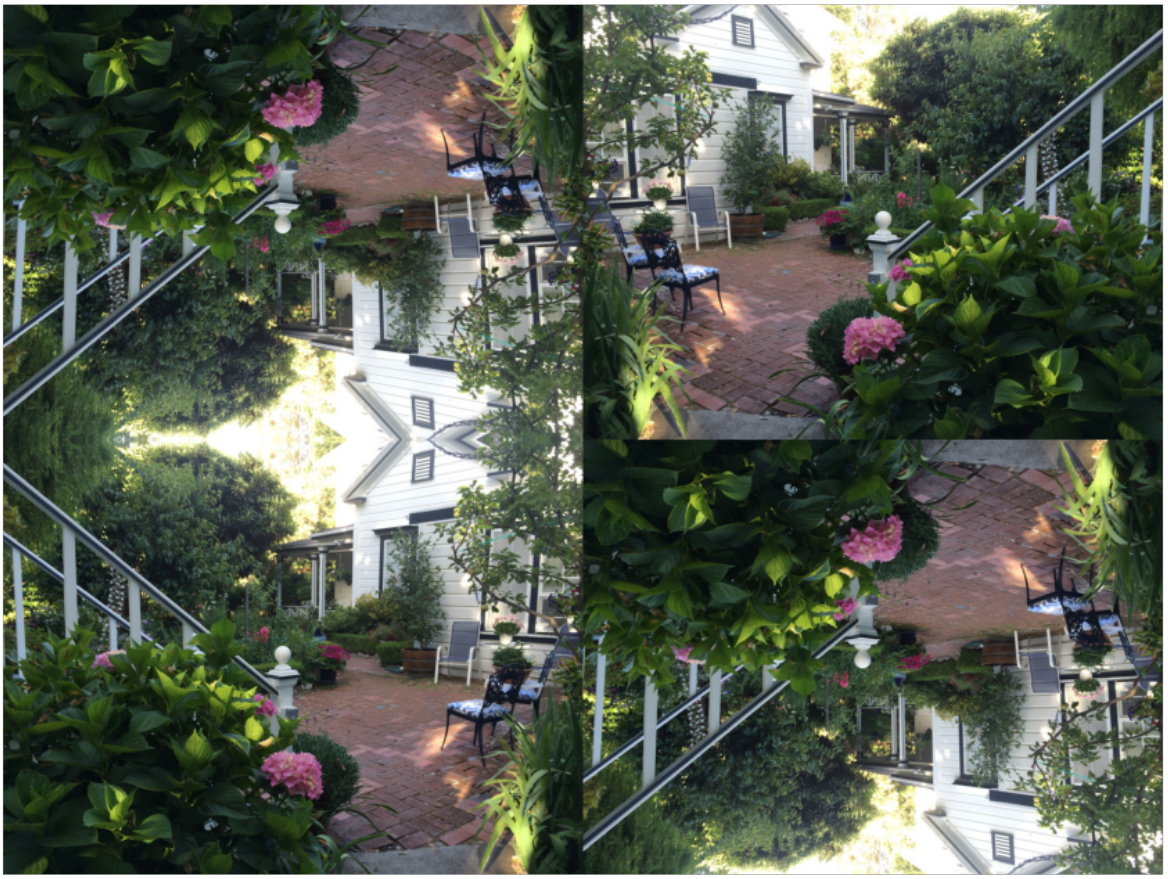
In [3]:
```
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
import matplotlib.pyplot as plt
import numpy as np
import matplotlib

def visualize_generator(img_file, gen):
```

```
      # Load the requested image
img = load_img(img_file)
data = img_to_array(img)
samples = expand_dims(data, 0)

      # Generat augumentations from the generator
it = gen.flow(samples, batch_size=1)
images = []
for i in range(4):
  batch = it.next()
  image = batch[0].astype('uint8')
  images.append(image)

images = np.array(images)

      # Create a grid of 4 images from the generator
index, height, width, channels = images.shape
nrows = index//2

grid = (images.reshape(nrows, 2, height, width, channels)
          .swapaxes(1,2)
          .reshape(height*nrows, width*2, 3))

fig = plt.figure(figsize=(15., 15.))
plt.axis('off')
plt.imshow(grid)
```

We begin by flipping the image. Some images may not make sense to flip, such as this landscape. However, if you expect "noise" in your data where some images may be flipped, then this augmentation may be useful, even if it violates physical reality.

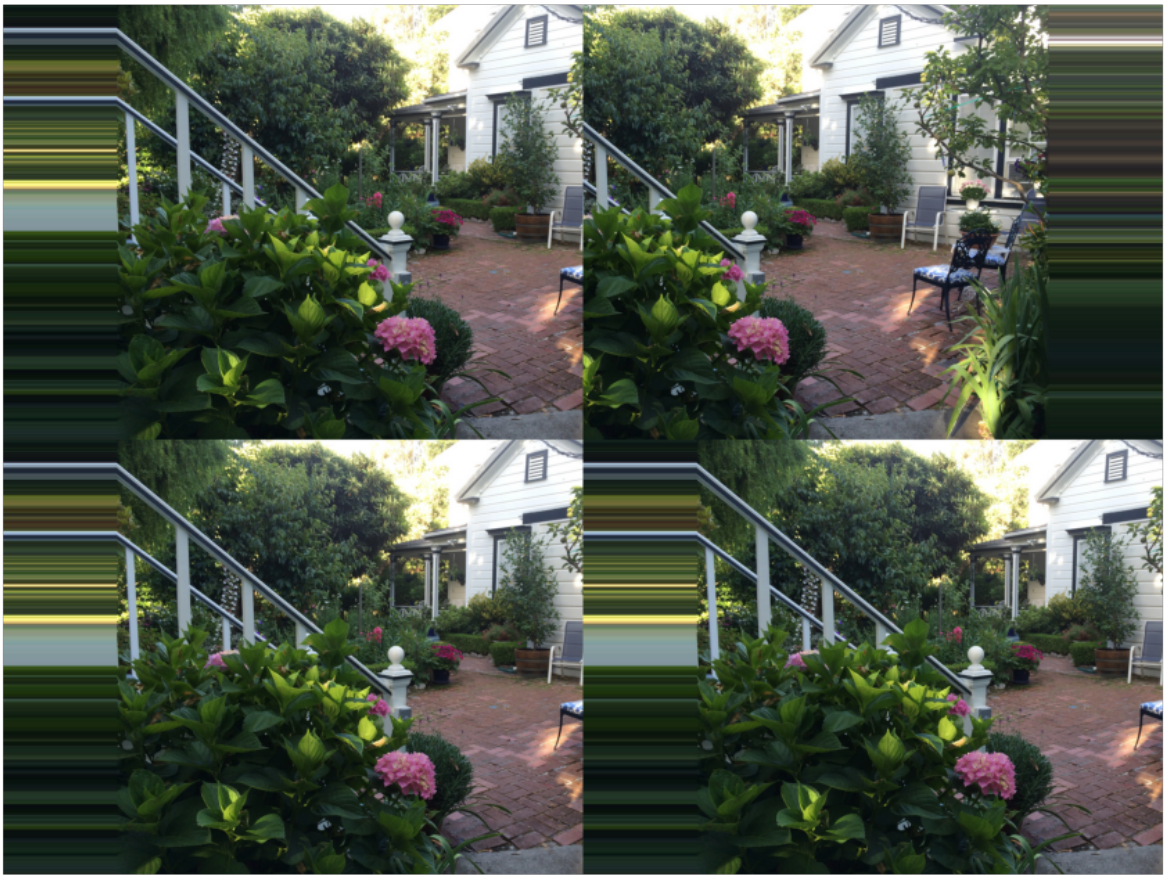
In [4]:

```
visualize_generator(
  LOCAL_IMG_FILE,
  ImageDataGenerator(horizontal_flip=True, vertical_flip=True))
```



Next, we will try moving the image. Notice how part of the image is missing? There are various ways to fill in the missing data, as controlled by **fill_mode**. In this case, we simply use the nearest pixel to fill. It is also possible to rotate images.

In [5]:
```
visualize_generator(
    LOCAL_IMG_FILE,
    ImageDataGenerator(width_shift_range=[-200,200],
        fill_mode='nearest'))
```



We can also adjust brightness.

In [6]:
```
visualize_generator(
  LOCAL_IMG_FILE,
  ImageDataGenerator(brightness_range=[0,1]))

# brightness_range=None, shear_range=0.0
```

Shearing may not be appropriate for all image types, it stretches the image.

In [7]:
```
visualize_generator(
  LOCAL_IMG_FILE,
  ImageDataGenerator(shear_range=30))
```

It is also possible to rotate images.

In [8]:
```
visualize_generator(
  LOCAL_IMG_FILE,
  ImageDataGenerator(rotation_range=30))
```

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. Running the following code will map your GDrive to `/content/drive`.

In [1]:
```
try:
    from google.colab import drive
    COLAB = True
    print("Note: using Google CoLab")
    %tensorflow_version 2.x
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

Part 6.5: Recognizing Multiple Images with YOLO5

Programmers typically design convolutional neural networks to classify a single item centered in an image. However, as humans, we can recognize many items in

our field of view in real-time. It is advantageous to recognize multiple items in a single image. One of the most advanced means of doing this is YOLOv5. You Only Look Once (YOLO) was introduced by Joseph Redmon, who supported YOLO up through V3. [Cite:redmon2016you] The fact that YOLO must only look once speaks to the efficiency of the algorithm. In this context, to "look" means to perform one scan over the image. It is also possible to run YOLO on live video streams.

Joseph Redmon left computer vision to pursue other interests. The current version, YOLOv5 is supported by the startup company Ultralytics, who released the open-source library that we use in this class.[Cite:zhu2021tph]

Researchers have trained YOLO on a variety of different computer image datasets. The version of YOLO weights used in this course is from the dataset Common Objects in Context (COCO). [Cite: lin2014microsoft] This dataset contains images labeled into 80 different classes. COCO is the source of the file coco.txt used in this module.

Using YOLO in Python

To use YOLO in Python, we will use the open-source library provided by Ultralytics.

- YOLOv5 GitHub

The code provided in this notebook works equally well when run either locally or from Google CoLab. It is easier to run YOLOv5 from CoLab, which is recommended for this course.

We begin by obtaining an image to classify.

In [2]:
```
import urllib.request
import shutil
from IPython.display import Image
!mkdir /content/images/

URL = "https://github.com/jeffheaton/t81_558_deep_learning"
URL += "/raw/master/photos/jeff_cook.jpg"
LOCAL_IMG_FILE = "/content/images/jeff_cook.jpg"

with urllib.request.urlopen(URL) as response, \
  open(LOCAL_IMG_FILE, 'wb') as out_file:
    shutil.copyfileobj(response, out_file)

Image(filename=LOCAL_IMG_FILE)
```

Out[2]:

Installing YOLOv5

YOLO is not available directly through either PIP or CONDA. Additionally, YOLO is not installed in Google CoLab by default. Therefore, whether you wish to use YOLO through CoLab or run it locally, you need to go through several steps to install it. This section describes the process of installing YOLO. The same steps apply to either CoLab or a local install. For CoLab, you must repeat these steps each time the system restarts your virtual environment. You must perform these steps only once for your virtual Python environment for a local install. If you are installing locally, install to the same virtual environment you created for this course. The following commands install YOLO directly from its GitHub repository.

In [3]:

```
import sys

!git clone https://github.com/ultralytics/yolov5 --tag 6.2  # clone
!mv /content/6.2 /content/yolov5
%pip install -qr /content/yolov5/requirements.txt  # install
sys.path.insert(0,'/content/yolov5')

import torch
import utils
display = utils.notebook_init()  # checks
```

```
YOLOv5 🚀 v6.2-195-gdf80e7c Python-3.7.14 torch-1.12.1+cu113 CUDA:0 (Tesla
T4, 15110MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 38.8/166.8 GB disk)
```

Next, we will run YOLO from the command line and classify the previously downloaded kitchen picture. You can run this classification on any image you choose.

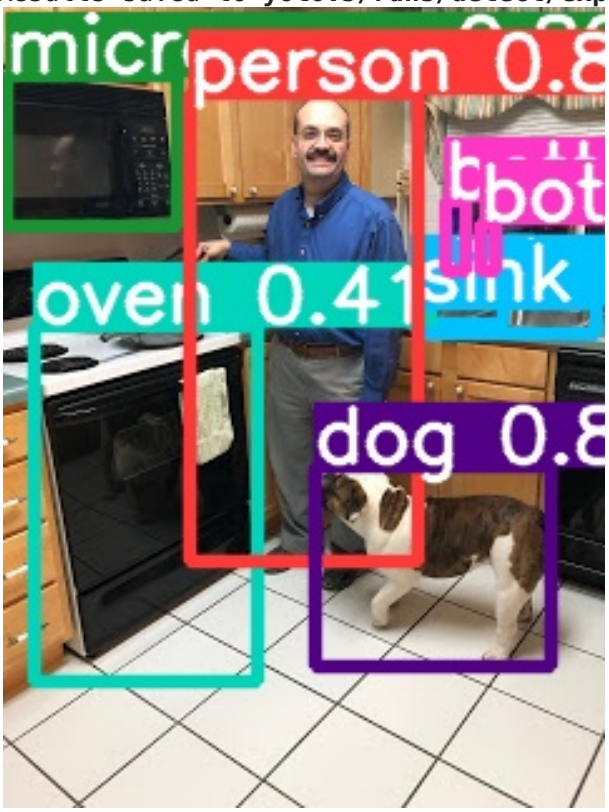
In [4]:

```
# Prepare directories for YOLO command line
!rm -R /content/yolov5/runs/detect/*
!mkdir /content/images
```

```
!cp /content/street/jeff_cook.jpg /content/images

# Run YOLO to classify
!python /content/yolov5/detect.py --weights yolov5s.pt --img 1024 \
  --conf 0.25 --source /content/images/

# Display the images
from IPython.display import Image

URL = '/content/yolov5/runs/detect/exp/jeff_cook.jpg'
Image(filename=URL, width=300)
```

```
rm: cannot remove '/content/yolov5/runs/detect/*': No such file or directo
ry
mkdir: cannot create directory ‘/content/images’: File exists
cp: cannot stat '/content/street/jeff_cook.jpg': No such file or directory
detect: weights=['yolov5s.pt'], source=/content/images/, data=yolov5/data/
coco128.yaml, imgsz=[1024, 1024], conf_thres=0.25, iou_thres=0.45, max_det
=1000, device=, view_img=False, save_txt=False, save_conf=False, save_crop
=False, nosave=False, classes=None, agnostic_nms=False, augment=False, vis
ualize=False, update=False, project=yolov5/runs/detect, name=exp, exist_ok
=False, line_thickness=3, hide_labels=False, hide_conf=False, half=False,
dnn=False, vid_stride=1
YOLOv5 🚀 v6.2-195-gdf80e7c Python-3.7.14 torch-1.12.1+cu113 CUDA:0 (Tesla
T4, 15110MiB)

Downloading https://github.com/ultralytics/yolov5/releases/download/v6.2/y
olov5s.pt to yolov5s.pt...
100% 14.1M/14.1M [00:00<00:00, 236MB/s]

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients
image 1/1 /content/images/jeff_cook.jpg: 1024x768 1 person, 1 dog, 2 bottl
es, 1 microwave, 1 oven, 2 sinks, 21.3ms
Speed: 0.8ms pre-process, 21.3ms inference, 41.7ms NMS per image at shape
(1, 3, 1024, 1024)
Results saved to yolov5/runs/detect/exp
```

Out[4]:



In [5]:
```
!ls /content/yolov5/
```

```
benchmarks.py    detect.py   models            runs        tutorial.ipynb
classify         export.py   __pycache__       segment     utils
CONTRIBUTING.md  hubconf.py  README.md         setup.cfg   val.py
data             LICENSE     requirements.txt  train.py
```

Running YOLOv5

In addition to the command line execution, we just saw. The following code adds the downloaded YOLOv5 to Python's environment, allowing **yolov5** to be imported like a regular Python library.

In [6]:
```
import torch

# Model
yolo_model = torch.hub.load('ultralytics/yolov5', 'yolov5s')  # or yolov5

# Inference
results = yolo_model(LOCAL_IMG_FILE)

# Results
df = results.pandas().xyxy[0]
df
```

```
/usr/local/lib/python3.7/dist-packages/torch/hub.py:267: UserWarning: You
are about to download and run code from an untrusted repository. In a futu
re release, this won't be allowed. To add the repository to your trusted l
ist, change the command to {calling_fn}(..., trust_repo=False) and a comma
nd prompt will appear asking for an explicit confirmation of trust, or loa
d(..., trust_repo=True), which will assume that the prompt is to be answer
ed with 'yes'. You can also use load(..., trust_repo='check') which will o
nly prompt for confirmation if the repo is not already trusted. This will
eventually be the default behaviour
  "You are about to download and run code from an untrusted repository. In
a future release, this won't "
Downloading: "https://github.com/ultralytics/yolov5/zipball/master" to /ro
ot/.cache/torch/hub/master.zip
YOLOv5 🚀 v6.2-195-gdf80e7c Python-3.7.14 torch-1.12.1+cu113 CUDA:0 (Tesla
T4, 15110MiB)

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients
Adding AutoShape...
```

Out[6]:

	xmin	ymin	xmax	ymax	confidence	class	name
0	125.092232	182.010025	219.074036	264.044983	0.928736	16	dog
1	72.338425	36.174423	162.752075	229.957077	0.928245	0	person
2	0.428009	25.537472	68.613434	89.955139	0.891785	68	microwave
3	0.000000	98.033714	103.113159	266.426483	0.739207	69	oven
4	176.110916	76.847527	183.783249	105.030785	0.725925	39	bottle
5	189.972397	85.284508	196.409378	105.729591	0.593492	39	bottle
6	161.864563	115.693741	237.386475	131.211624	0.571422	71	sink
7	216.053223	137.275635	239.968109	230.737457	0.364453	69	oven
8	181.397934	82.266541	195.568832	105.023056	0.252385	39	bottle

It is important to note that the **yolo** class instantiated here is a callable object, which can fill the role of both an object and a function. Acting as a function, *yolo* returns a Pandas dataframe that contains the bounding boxes (xmin/xmax and ymin/ymax), confidence, and name/class of each item detected.

Your program should use these values to perform whatever actions you wish due to the input image. The following code displays the images detected above the threshold.

You can obtain the counts of images through the use of a Pandas groupby and pivot.

In [7]:

```
df2 = df[['name','class']].groupby(by=["name"]).count().reset_index()
df2.columns = ['name','count']
df2['image'] = 1
df2.pivot(index=['image'],columns='name',values='count').reset_index().f
```

Out[7]:

name	image	bottle	dog	microwave	oven	person	sink
0	1	3	1	1	2	1	1

Module 6 Assignment

You can find the first assignment here: assignment 6