Data Input and Augmentation for Nilgai Object Classification

Matthew Kutugata

Dec 2019

Abstract

Nilgai in south Texas are both a pest and economic driver. As a result, an intensive USDA supported research project has been put in place to monitor nilgai using trail camera data. Analyzing this type of data, however, is burdensome and incredibly time consuming. The problem is primed for a computer vision-deep learning approach. Training a model to automatically classify trail cam images is complex. The first step to this process is creating a data input pipeline and performing various preprocessing techniques. We develop methods for 1)inputting data via batch-processing and 2) augment, expand, and diversify a limited dataset. Both use the Keras deep learning library supported by a Tensorflow 2 backend. We use the `ImageDataGenerator` class as our main tool to perform these tasks. As result, we were able to generate an additional 300 training images by using only a fraction of the available data augmentation tools. This method provides an easy, efficient, and viable solution to input data that can later be used to train a deep learning classifier.

1 Introduction

Nilgai in south Texas are both a pest and an economic driver. Ranchers aim to protect their cattle from the tick carry and cause cattle-tick fever, while landowners welcome the large trophy animal for its ability to attract hunters and tourism. As a result, the United State Department of Agriculture Agricultural Research Project (USDA ARS) is involved in pursuing a viable solution that placates the demands of both ranchers and landowners (Foley *et al.* , 2017).

An important component of this project is monitoring nilgai and other animals via trail cams throughout the greater lower Rio Grande Valley. ARS staff are employed to sift through masses of geo-located images in hopes of monitoring movement of nilgai. This sifting of trail cam data occupies inordinate amounts of time for staff. The bottle necking of data analysis causes wasted resources and lost hours. This, however, can be resolved by using state-of-the-art computer vision and processing techniques.

Object classification, the critical task in computer vision applications that classifies objects from different categories or classes, provides an efficient, fast,

and time-saving solution to this problem (Shanmugamani, 2018). Spending large amounts of time in front of the computer can instead be replaced by an automatic process. This process is performed by convolutional neural networks, a class of deep neural networks that extract patterns or features from unstructured data (images) (Brownlee, 2019). Before training a predictive object classifier for nilgai, however, we must develop a process for organizing, loading, and preprocessing images.

Data input is an important process of any data science project and often the most arduous. The goal of this project is to develop a method that organizes, loads, and applies data augmentation techniques that can then be used in a convolutional neural network for object classification. We will review the available tools and packages, provide information about organizing files and image directories, and perform data augmentation that both diversifies and expands our data set.

2 Methods

Data input and preprocessing is an important process for any data science project and especially true of one that uses many images files. This process begins before any code is written, and in our case starts with the organization of directories, or folders, on our workstation. We then move onto the actual python code, packages, and methods that bring images in, converting them to tensor pakcages, and preparing them for any preprocessing. Finally, our method ends with applying five major data augmentation techniques that expand and diversify our data set. This method is one that is used in any deep learning process that use images to gather information.

2.1 Data Organization

For this project, we began our data organization process by using a structure that is preferred by the application programming interface (API) of our main python library, Keras, that data be divided into specific train/ and test/ folders and within each, a separate folder individual cl assess (e.g. train/cow, test/pig, etc.)(see fig. 1) (Gulli & Pal, 2017). We acquired a small dataset by using a simple google search filtered to only include images labeled for reuse and with no known copyright restrictions. Images of all type were selected including partially obscured objects, different sized images, and with partial lighting. Images were selected, however, to only include objects belonging to a single class (see figure 2). Files where then converted to .PNG format and renamed to "classnumber.png" (cow1.png, cow2.png, etc). Image files are now ready to be brought into to our data input pipeline.

2.2 Data Input

After organizing images into our directories we load images progressively using the Keras Python library and its main data loader class and method (respectively). This approach is preferred over the in-memory approach that loads the entire training dataset into its memory when training a model. While we only have a small dataset for illustrative purposes, typical deep learning models require large amounts of training data that could cause a machine with insufficient RAM to crash. The in-memory approach would require RAM of 64GB or more. Cloud computing, like Amazon EC2 or TAKS, may be an option and this method would be slightly faster. Instead the progressive or as-needed approach will be used during future training and will be performed batch by batch using a built-in Keras data input tools (Gulli & Pal, 2017):

```
from keras.preprocessing.image
import ImageDataGenerator
from keras.preprocessing.image
import flow_from_directory
```

Training models using this method is slower, however, it can be performed locally on workstations with less RAM and limited access to high-performance computing system. We decided to pursue the former because many deep learning libraries have functions similar to these. Our goal is to become familiar with and develop proficiency in common deep learning approaches. Additionally, the progressive loading method can automatically scale pixel values, standardize image size, and automatically apply data augmentation to generate augmented versions of original images. The pattern for using the ImageDataGenerator class to load images from files is as follows:

1. Create an instance, or object, using the ImageDataGenerator class.
2. Use a tool for iterating, or built-in looping, over files by calling the function:

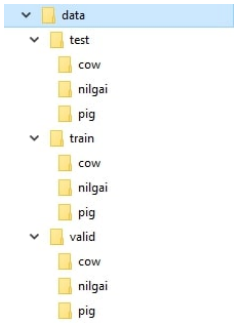

Figure 1: Directory organizational structure

`flow_from_directory()`

3. And lastly, use the iterator tool to call images during the training or evaluation of a model.

After using the built-in Keras tools and bringing in the images, it is important to inspect and ensure that the above tools functioned properly. We did this by loading images and plotting multiple images in a single figure using Matplotlib (see fig 2). Running the many training, validation, and test samples we see that images are in fact in the appropriate directory and being uploaded properly. Now that images are loaded to a preprocessing of images, specifically data augmentation, can occur.

2.3 Data Augmentation

Deep learning models are a data hungry technique that require many and diverse sets of training examples. Data augmentation, changing images slightly so as not drastically alter their main components, is one way that users can increase their training set size. The Keras python package gives users the ability to augment training data automatically using ImageDataGenerator. Following the methods above, a instance or object of the ImageDataGenerator must be called then a range of arguments can be specified to the instance. Five augmentation techniques will be discussed.

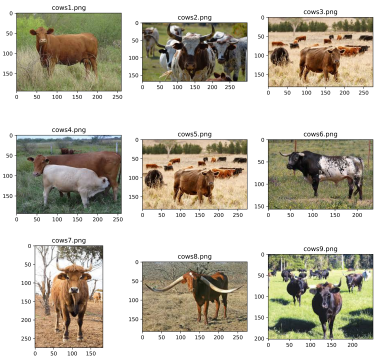

Figure 2: Train sample cows

2.3.1 Horizontal and Vertical Shift Augmentation

Shifting an images refers to the movement of pixels to one direction either horizontally or vertically while retaining the same image dimensions. Image will both be removed to the shifting direction while pixels on the opposite side re given a specified value. `width_shift_range and height_shift_range` in ImageDataGenerator instance determines the floating point value of that dictates the percentage of width or height that the image will shift (see fig **??**).

2.3.2 Horizontal and Vertical Flip

Flipping an images refers to reversing the rows and columns of pixels either vertically or horizontally. The flip argument in the ImageDataGenerator instance is a Boolean and specified by either applying `vertical_flip` or `horizontal_flip`.

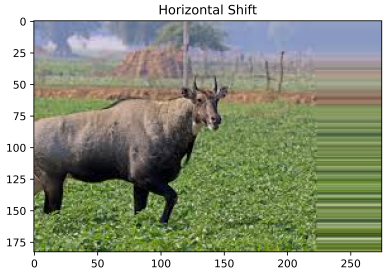

Figure 3: Horizontal shift with range [-50, 50]

2.3.3 Random Rotation

Randomly rotating an image can be performed by giving a single integer value between 0 to 360 to denote the degrees of rotation. The rotation will inevitably rotate certain pixels out of the picture frame and will thus have to specified to fill in the empty areas. Random rotation can be used by applying the `rotation_range` argument and inputting a single number (see fig 4).

2.3.4 Random Brightness

Augmenting an image by way of random brightness includes either brightening or darkening and image or both. Randomly changing the brightness of an images greatly enhances a model's ability to be trained and tested on different image types. The brightness augment can be used by specifying a `brightness_range` above or below 1.0. Values above 1.0 brighten the image, while ranges below darken it (see figure 5).

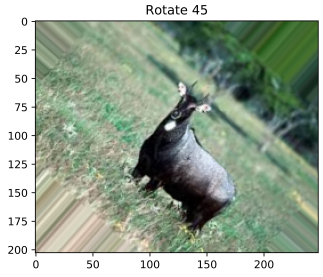

Figure 4: Random rotation

2.3.5 Random Zoom

Lastly, random zoom can be applied to images by either adding new pixel values around the image or by interpolating pixel values together. This techniques can likewise be applied by adding the `zoom_range` argument to the data instance and by providing either a range as an array or as a single float to specify the percentage (see fig **??**).

3 Results

As a result we discovered how to successfully organize, input, and augment image data to be used in training a deep learning neural network. Specifically, we learned that image augmentation is used to to expand and diversify a training dataset. Doing so improves the performance and ability of the training model to correctly predict the class of never before seen images. Secondly, we learned that various image data augmentation tools are supported in the Keras library and that `ImageDataGenerator` is an efficient class for both inputting and automatically preprocessing images. More specifically, we learned the details of using `ImageDataGenerator` shift, flip, brighten, and zoom arguments to substantially increase our training dataset. Our image dataset went from 20 training examples per class to 120 images per class by only using five data augmentation techniques. This can be significantly increased by varying the ranges of rotation, shift, zoom, and brightness.

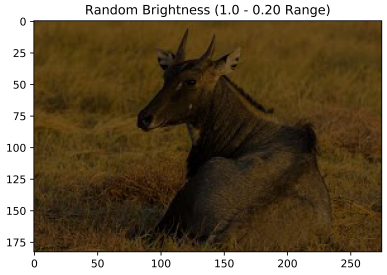

Figure 5: Random Brightness [0.2, 1.0]

4 Discussion

Image data input is a major component of any data science project and requires forethought and organization. Preprocessing image data is also an important process of formatting and augmenting data that will enhance the ability of your model to generalize on unseen examples. Train deep learning neural networks on more data create more accurate models and can easily be said to be better. The Keras deep learning library provides a number of use full tools to do this in a way that is efficient and standardize.

References

Brownlee, Jason. 2019. *Deep Learning for Computer Vision: Image Classification, Object Detection, and Face Recognition in Python*. Machine Learning Mastery.

Foley, Aaron M, Goolsby, John A, Ortega-S Jr, Alfonso, Ortega-S, J Alfonso, de León, A Pérez, Singh, Nirbhay K, Schwartz, Andy, Ellis, Dee, Hewitt, David G, & Campbell, Tyler A. 2017. Movement patterns of nilgai antelope in South Texas: Implications for cattle fever tick management. *Preventive veterinary medicine*, **146**, 166–172.

Gulli, Antonio, & Pal, Sujit. 2017. *Deep Learning with Keras*. Packt Publishing Ltd.

Shanmugamani, Rajalingappaa. 2018. *Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras*. Packt Publishing Ltd.

Appendices

A Code Examples

Listing 1: Data Input ImageDateGenerator Instance

```
# Import dependencies and TF for good measure
import tensorflow as tf
from keras.datasets import mnist
from keras.preprocessing.image import ImageDataGenerator

#check tf version
tf.__version__


# create paths to images sets
train_path = 'C:\\Users\\ncu116\\Desktop\\data\\train'
val_path = 'C:\\Users\\ncu116\\Desktop\\data\\valid'
test_path =  'C:\\Users\\ncu116\\Desktop\\data\\test'

# create data generator instance
datagen = ImageDataGenerator()

# prepare an iterators for each dataset arg = data directory and
#class_mode = classification type - either multiclass 'categorical' or 'binary
train_it = datagen.flow_from_directory(train_path, class_mode='categorical')
val_it = datagen.flow_from_directory(val_path, class_mode='categorical')
test_it = datagen.flow_from_directory(test_path, class_mode='categorical')

# confirm iterator works
batchX, batchy = train_it.next()
print('Batch shape=%s, min=%.3f, max=%.3f' % (batchX.shape, batchX.min(), batchX.max()))
batchX.shape
```

Listing 2: Data Augmentation: Horizontal Shift

```
# install dependincies
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

# load the image
single_nilgai_path = 'C:\\Users\\ncu116\\Desktop\\data\\train\\nilgai\\nilgai1.png'
#img = load_img('C:\\Users\\ncu116\\Desktop\\data\\chapter_09\\bird.jpg')
save_here = 'C:\\Users\\ncu116\\Desktop\\data\\test\\test_augmented'
```

```
img = load_img(single_nilgai_path)
# convert to numpy array
data = img_to_array(img)
# expand dimension to one smaple
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(width_shift_range=[-50,50])

# prepare iterator and save new examples to file "save here"
it = datagen.flow(samples, batch_size=1 ) #additional "save" argument can be used
pyplot.figure(figsize=(12, 12))
for i in range(1):
    pyplot.subplot(330 + 1 + i)
    batch = it.next()
    image = batch[0].astype('uint8')
    #fig = pyplot.figure()
    pyplot.imshow(image)
    pyplot.title('Horizontal Shift')

# show the figure

pyplot.show()
fig.savefig('hhhoz_shift.png', dpi=800)
```

Listing 3: Data Aug.: Vertical Flip

```
single_nilgai_path = 'C:\\Users\\ncu116\\Desktop\\data\\train\\nilgai\\nilgai1.png'

img = load_img(single_nilgai_path)
# convert to numpy array
data = img_to_array(img)
# expand dimension to one smaple
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(vertical_flip=True)
it = datagen.flow(samples, batch_size=1)
pyplot.figure(figsize=(19, 12))

#generate samples and plot
for i in range(1):
    #define subplot
    pyplot.subplot(330 +1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    fig = pyplot.figure()
    pyplot.imshow(image)
    pyplot.title('Vertical Flip')
    fig.savefig('1vert_flip.png', dpi=800)
```

Listing 4: Data Aug: Rotation

```
single_nilgai_path = 'C:\\Users\\ncu116\\Desktop\\data\\train\\nilgai\\nilgai1.png'
#save_here = 'C:\\Users\\ncu116\\Desktop\\data\\test\\test_augmented'
```

```
img = load_img(single_nilgai_path)
# convert to numpy array
data = img_to_array(img)
# expand dimension to one smaple
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(rotation_range=45)
it = datagen.flow(samples, batch_size=1)
pyplot.figure(figsize=(19, 12))

#generate samples and plot
for i in range(1):
    #define subplot
    pyplot.subplot(330 +1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    fig = pyplot.figure()
    pyplot.imshow(image)
    pyplot.title('Rotate 45')
    fig.savefig('1Rotate_flip.png', dpi=800)
```

Listing 5: Data Aug: Random Brightness

```
single_nilgai_path = 'C:\\Users\\ncu116\\Desktop\\data\\train\\nilgai\\nilgai1.png'

img = load_img(single_nilgai_path)
# convert to numpy array
data = img_to_array(img)
# expand dimension to one smaple
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(brightness_range=[0.2, 1.0])
it = datagen.flow(samples, batch_size=1)
pyplot.figure(figsize=(19, 12))

#generate samples and plot
for i in range(1):
    #define subplot
    pyplot.subplot(330 +1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    fig = pyplot.figure()
    pyplot.imshow(image)
    pyplot.title('Random Brightness (1.0 - 0.20 Range)')
    fig.savefig('1Rando_Bright.png', dpi=800)
```

Listing 6: Data Aug: Random Zoom

```
single_nilgai_path = 'C:\\Users\\ncu116\\Desktop\\data\\train\\nilgai\\nilgai1.png'
#save location
save_here = 'C:\\Users\\ncu116\\Desktop\\data\\train\\nilgai'
```

```
img = load_img(single_nilgai_path)
# convert to numpy array
data = img_to_array(img)
# expand dimension to one smaple
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(zoom_range=[0.5, 1.0])
#batch flow form directory
it = datagen.flow(samples, batch_size=1,
                  save_to_dir=save_here,
                  save_prefix='aug',
                  save_format='png')

pyplot.figure(figsize=(19, 12))

#generate samples and plot
for i in range(1):
    #define subplot
    pyplot.subplot(330 +1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    fig = pyplot.figure()
    pyplot.imshow(image)
    pyplot.title('Random Zoom (0.5 - 1.0 Range)')
    fig.savefig('1rando_zoom.png', dpi=800)
```