ITERATION - 1

Study Project: The Camera Trap Challenge

Computer Vision & Machine Learning Algorithms to Analyse Remote Sensing Camera Trap Data

Group 07

Mohamed Shamsudeen (536944)

Ram Kumar Muthusamy(536951)

Coherent Sequence of Images

```
import os
from PIL import Image
import datetime
```

```
BADGER_DIR = 'E:\WWU\project\SP\og\meles_meles_dachs\dayvision'
DEER_DIR = 'E:\WWU\project\SP\og\dama\dayvision'
```

```
def extract_sequences(image_dir):
    images = [image for image in os.listdir(image_dir)]
    images.sort(key=lambda x: extract_timestamp(os.path.join(image_dir, x)))
    sequences = []
    current_sequence = []
    previous_timestamp = None
    for image in images:
        timestamp = extract_timestamp(os.path.join(image_dir, image))
        if previous_timestamp is None or (
                timestamp - previous_timestamp).total_seconds() <= 600:  # Time interval <= 10 minutes
            current_sequence.append(image)
        else:
            sequences.append(current_sequence)
            current_sequence = [image]
        previous_timestamp = timestamp
    return sequences

def extract_timestamp(image_path):
    image = Image.open(image_path)
    exif_data = image.getexif()
    # print(exif_data)
    # Extract the date and time from the metadata
    date_str = exif_data.get(306)  # DateTimeOriginal tag
    if date_str:
        timestamp = datetime.datetime.strptime(date_str, '%Y:%m:%d %H:%M:%S')
        if timestamp:
            return timestamp
    return None
```

```
badger_sequences = extract_sequences(BADGER_DIR)
deer_sequences = extract_sequences(DEER_DIR)

for i in badger_sequences:
    print(i)
```

Locate the Animals

MegaDetector model files

We'll download both MegaDetector v5a and v5b.

MDv5a-> designed to detect animal presence in camera trap images. It uses a convolutional neural network (CNN) to classify each image as containing an animal or not.

MDv5b-> designed to classify species of animal detected in image. It uses a different CNN to identify the species based on features such as color, size, and shape.

```
pip install humanfriendly jsonpickle
```

```
pip install torch==1.10.1 torchvision==0.11.2
```

```
!wget -O /content/md_v5a.0.0.pt https://github.com/microsoft/CameraTraps/releases/download/v5.0/md_v5a.0.0.pt
!wget -O /content/md_v5b.0.0.pt https://github.com/microsoft/CameraTraps/releases/download/v5.0/md_v5b.0.0.pt
```

Required GIT repos Cloning

To run MegaDetector, the latest versions of the Microsoft AI for Earth "utilities" and "CameraTraps" repositories and the YOLOv5 repository are required.

```
!git clone https://github.com/microsoft/CameraTraps
!git clone https://github.com/microsoft/ai4eutils
!git clone https://github.com/ultralytics/yolov5/
!cd yolov5 && git checkout c23a441c9df7ca9b1f275e8c8719c949269160d1
```

Set `PYTHONPATH` to include `CameraTraps`, `ai4eutils`, and `yolov5`

Add cloned git folders to the `PYTHONPATH` environment variable so that we can import their modules from any working directory.

```
import os
from PIL import Image
os.environ['PYTHONPATH'] += ":/content/ai4eutils"
os.environ['PYTHONPATH'] += ":/content/CameraTraps"
os.environ['PYTHONPATH'] += ":/content/yolov5"
```

Google Drive Mount

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

MegaDetector batch processing

```
data = '/content/drive/My Drive/Colab Notebooks/dayvision'
# Save output JSON file
result = '/content/drive/My Drive/dayvision/results.json'
```

Run detector batch

If running in Colab session with a GPU accelerator, It process around five images per second.

```
!python /content/CameraTraps/detection/run_detector_batch.py md_v5a.0.0.pt "$data" "$result" --recursive --output_filename --quiet
```

Visualize batch processing

The `visualize_detector_output.py` in `visual` folder of Camera Traps repo to see the output of MegaDetector visualized on our images. It will save images annotated with results.

```
# Render bounding boxes on our images
visual = '/content/visual_img'
!python /content/CameraTraps/visualization/visualize_detector_output.py "$result" "$visual" --confidence 0.2 --data "$data"
```

```
# Show the images with bounding boxes in Colab
for visual_file in os.listdir(visual):
  print(visual_file)
  image = Image.open(os.path.join(visual, visual_file))
  display(image)
```

Classification of Animals

```
!pip install split-folders
```

```
import os
import cv2
import glob
import numpy as np
import splitfolders
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from google.colab.patches import cv2_imshow
```

```
from google.colab import drive
```

```
# dataset import
drive.mount('/content/drive')
data = '/content/drive/My Drive/data'
```

```
# counting the number of files in train folder
path, dirs, files = next(os.walk('/content/drive/My Drive/data/Dee_badg'))
count = len(files)
print('Number of images: ', count)
```

```
# Importing both the deer and badger as a single file
dee_badg = os.listdir('/content/drive/My Drive/data/Dee_badg')
print(dee_badg)
```

Resize Image

```
#Image resized
os.mkdir('/content/resize_img')
```

```
data = '/content/drive/My Drive/data/Dee_badg/'
res = '/content/resize_img/'

for i in range(299):
  files = os.listdir(data)[i]
  image_dir = data + files
  img = Image.open(image_dir)
  img = img.resize((299, 299))
  img = img.convert('RGB')
  newImgPath = res + files
  img.save(newImgPath)
```

```
img_dir = '/content/resize_img/'
# get a list of all the file names in the directory
file = os.listdir(img_dir)

# store the labels
labels = []
# loop through each file name in directory
for file_name in file:
    # first character of file name
    first_char = file_name[0]
    # set label to 1 if first character is 'B', otherwise set it to 0
    if first_char == 'B':
        label = 1
    else:
        label = 0
    # append the label to the list of labels
    labels.append(label)
```

```
print(file[0:5])
print(labels[0:5])
print(len(file))
```

```
val, cnt = np.unique(labels, return_counts=True)
print(val)
print(cnt)
```

```
img_dir = '/content/resize_img/'
img_ext = ['JPG', 'jpg']  # image extensions to search
files = []                # empty list to store file paths

# loop through each extension in the list of extensions
for ext in img_ext:
    ext_files = glob.glob(img_dir + '*.' + ext)  # find all files in directory with given extension
    files.extend(ext_files)                      # append the file paths to the list of files
# list comprehension to read images and convert to numpy array
bd_img = np.asarray([cv2.imread(file) for file in files])
```

```
print(bd_img)
```

```
type(bd_img)
```

```
print(bd_img.shape)
```

```
X = bd_img
Y = np.asarray(labels)
```

Train Test - Split

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

239 - Training Images

60 - Test Images

```
# scaling the data
X_train_scl = X_train/255

X_test_scl = X_test/255
```

```
print(X_train_scl)
```

Building Neural Network

```
import tensorflow as tf
import tensorflow_hub as hub
```

```
incp_model = ('https://tfhub.dev/google/inaturalist/inception_v3/feature_vector/5')
pretrain_model = hub.KerasLayer(incp_model, input_shape=(299,299,3), trainable=False)
```

```
classes = 2
model = tf.keras.Sequential([pretrain_model,tf.keras.layers.Dense(classes)])
model.summary()
```

```
model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics = ['acc'])
```

```
history = model.fit(X_train_scl, Y_train, epochs=25, validation_data=(X_test_scl, Y_test))
```

```
score, acc = model.evaluate(X_test_scl, Y_test)
print('Test Loss =', score)
print('Test Accuracy =', acc)
```

```
score, acc = model.evaluate(X_train_scl, Y_train)
print('Train Loss =', score)
print('Train Accuracy =', acc)
```

```
# plot the training and validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()

# plot the training and validation accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')
plt.show()
```

```
samp_img = '/content/drive/My Drive/data/Dee_badg/B_BBIMG_1783.JPG'

# Process the first image
input = cv2.imread(samp_img)
cv2_imshow(input)
input_resize = cv2.resize(input, (299,299))
input_scl = input_resize/255
image_reshape = np.reshape(input_scl, [1,299,299,3])
input_pred = model.predict(image_reshape)
input_pred_label = np.argmax(input_pred)

if input_pred_label == 0:
    print('Badger Image')
else:
    print('Deer Image')
```

ITERATION-II

Create Split Folders

```
import os
import shutil
from sklearn.model_selection import train_test_split

# Set the path to the original folder containing all the images
original_folder = 'C:/Users/ramkumar_m1768/Pictures/model'

# Set the path to the output folder where the split folders will be created
output_folder = 'C:/Users/ramkumar_m1768/Pictures/model/trainoutput'

# Set the proportion of images to be used for the training, validation, and testing sets
train_size = 0.7
val_size = 0.2
test_size = 0.1

# Define a list of classes for which the subfolders will be created
classes = ['deer', 'fox', 'badger','boar','hare','rabbit','sheep','red_deer','roh']

# Loop through the list of classes
for cls in classes:
    # Get a list of all the image file names in the original folder for this class
    class_folder = os.path.join(original_folder, cls)
    image_files = os.listdir(class_folder)
    # Split the image file names into training, validation, and testing sets
    train_files, test_files = train_test_split(image_files, test_size=test_size, random_state=42)
    train_files, val_files = train_test_split(train_files, test_size=val_size/(1-test_size), random_state=42)
    # Create the output folders if they don't exist already
    os.makedirs(os.path.join(output_folder, 'train_' + cls), exist_ok=True)
    os.makedirs(os.path.join(output_folder, 'val_' + cls), exist_ok=True)
    os.makedirs(os.path.join(output_folder, 'test_' + cls), exist_ok=True)
    # Copy the training set images to the training folder
    for file in train_files:
        shutil.copy2(os.path.join(class_folder, file), os.path.join(output_folder, 'train_' + cls))
    # Copy the validation set images to the validation folder
    for file in val_files:
        shutil.copy2(os.path.join(class_folder, file), os.path.join(output_folder, 'val_' + cls))
    # Copy the testing set images to the testing folder
    for file in test_files:
        shutil.copy2(os.path.join(class_folder, file), os.path.join(output_folder, 'test_' + cls))
```

```
#WE USED MEGADETECTOR TO FIND BOUNDING BOX OF ANIMALS
python detection\run_detector_batch.py "c:\megadetector\md_v5a.0.0.pt" "c:\some_image_folder" "c:\megadetector\test_output.json" --output
```

Modified Detected Category and Created Label

```
import os
import json

def add_category_to_detections(json_path, category_dict):
    with open(json_path, 'r') as f:
        data = json.load(f)

    for image in data['images']:
        directory_name = os.path.dirname(image['file'])

        if 'detections' not in image:
            continue
       else:
            for detection in image['detections']:
                category_name = category_dict.get(directory_name)
                if category_name is not None:
                    detection['category'] = category_name

    # save the modified JSON data to a new file
    with open('valid.json', 'w') as f:
        json.dump(data, f, indent=2)

category_dict = {
    'val_deer'    : 'deer',
    'val_badger'  : 'badger',
    'val_roh'     : 'roh',
    'val_boar'    : 'boar',
    'val_fox'     : 'fox',
   'val_hare'     : 'hare',
    'val_rabbit'  : 'rabbit',
    'val_red_deer': 'red_deer',
    'val_sheep'   : 'sheep'}
add_category_to_detections(‘md_valid.json', category_dict)
add_category_to_detections('md_train.json', category_dict)
```

YOLO annotations from json file

```
# MEGADETECTOR GIVES CUSTOMISED JSON -- So we converted it
import os
from PIL import Image
import json

class_labels = {'badger': 0, 'deer': 1, 'roh': 2, 'boar': 3, 'fox': 4, 'hare': 5, 'rabbit': 6, 'red_deer': 7, 'sheep': 8}

def create_yolo_annotations(json_path, class_labels):
    # Open JSON file and load data
    with open(json_path, 'r') as f:
        data = json.load(f)
    # Loop through each image in the JSON data
    for image_data in data['images']:
        # Get image filename and dimensions
        image_filename = os.path.basename(image_data['file'])
        image_path = os.path.join(os.getcwd(), os.path.dirname(image_data['file']), os.path.basename(image_data['file']))
        try:
            image_width, image_height = Image.open(image_path).size
        except FileNotFoundError:
            print(f"Skipping {image_path} as it is not found.")
            continue
        # Loop through each detection in the image
        annotations = []
        if 'detections' in image_data:
            for detection in image_data['detections']:
                # Get class label and bounding box coordinates
                class_label = detection['category']
                bbox = detection['bbox']
                x = bbox[0]
                y = bbox[1]
                w = bbox[2]
                h = bbox[3]
                x_centre = (x + (x+w))/2.0
                y_centre = (y + (y+h))/2.0
                x_centre = x_centre * img_w
                y_centre = y_centre * img_h
                w = w * img_w
                h = h * img_h
                x = x_centre / img_w
                y = y_centre / img_h
                w = w / img_w
                h = h / img_h
                annotation = f"{class_labels[class_label]} {x} {y} {w} {h}\n"
                annotations.append(annotation)
        else:
            print("No detections found in", image_path)

        # Write annotations to text file
        txt_filename = os.path.splitext(image_filename)[0] + ".txt"
        txt_path = os.path.join(os.path.dirname(image_path), txt_filename)
        with open(txt_path, 'w') as out_f:
            out_f.write("".join(annotations))
```

Classification

```
import os
!git clone https://github.com/ultralytics/yolov5
!pip install -r requirements.txt
```

```
!python train.py --weights /kaggle/working/yolov5/md_v5a.0.0.pt --epochs 100 --batch-size 32  --data /kaggle/working/yolo.yaml --freeze 1
```

```
!python detect.py --name test --weights /kaggle/input/yolov5/best_v5.pt --exist-ok --conf-thres 0.1 --source /kaggle/working/testdata
```

```
model= YOLO(yolov8n.pt)
model.train(data=data, epoch=200, batch = -1, patience = 30)
```

```
model(path)
```

```
!pip install ultralytics
from ultralytics import YOLO
```

```
model.train(data=data, epoch=200, batch = -1, patience = 30)
```

```
model2.predict(source = path, show = True, save = True , conf = 0.5 )
```