MMAI 5500 Assignment 3 – Anomaly detection

The goal of this assignment is to implement an autoencoder for anomaly detection (aka outlier detection). Anomaly detection is the identification of rare items, events or observations which deviate significantly from the majority of the data and do not conform to a well defined notion of normal behaviour. Anomaly detection have applications in many domains including cyber security, medicine, machine vision, statistics, neuroscience, law enforcement and financial fraud.

The core idea in using autoencoders for anomaly detection is that the observations that are most common (i.e. that there are most of) will be encoded better (with lower loss) that uncommon (outlier/anomalous) observations. By setting a threshold for the loss, observations exceeding it can be marked as anomalous.

Here, you will detect rare events in video.

Submission

The assignment should be submitted as Python 3 code and uploaded to Canvas as a single `PY` file (**not** a Jupyter Notebook) and the trained model. The due date is on November 27 at 2:30 pm.

Data

You will use a short video and in it detect frames where something unusual happens. The video is from the Anomalous Behavior Data Set, compiled by Andrei Zaharescu and Richard P. Wildes at the Vision Lab at York University.

The individual video frames need to be extracted from video and stored as JPEG images. Then, the images can be loaded into a Numpy array and used to train the model.

To extract the frames you can use the following function:

```python
import os
import cv2

def convert_video_to_images(img_folder, filename='assignment3_video.avi'):
    """
    Converts the video file (assignment3_video.avi) to JPEG images.
    Once the video has been converted to images, then this function doesn't
    need to be run again.

    Arguments
    ---------
    filename    : (string) file name (absolute or relative path) of video file.
    img_folder  : (string) folder where the video frames will be
                  stored as JPEG images.
    """
    # Make the img_folder if it doesn't exist.'
    try:
        if not os.path.exists(img_folder):
            os.makedirs(img_folder)
    except OSError:
        print('Error')

    # Make sure that the abscense/prescence of path
    # separator doesn't throw an error.
    img_folder = f'{img_folder.rstrip(os.path.sep)}{os.path.sep}'
    # Instantiate the video object.
    video = cv2.VideoCapture(filename)

    # Check if the video is opened successfully
    if not video.isOpened():
        print("Error opening video file")

    i = 0
    while video.isOpened():
        ret, frame = video.read()
        if ret:
            im_fname = f'{img_folder}frame{i:0>4}.jpg'
            print('Captured...', im_fname)
            cv2.imwrite(im_fname, frame)
            i += 1
        else:
            break

    video.release()
    cv2.destroyAllWindows()

    if i:
        print(f'Video converted\n{i} images written to {img_folder}')
```

To load the extracted image files use the function below. It returns the images both as a Numpy array of flattened images (i.e. the images with the 3-d shape (`im_width`, `im_height`, `num_channels`) are reshaped into the 1-d shape (`im_width` x `im_height` x `num_channels`)) and a list with the images with their original number of dimensions suitable for display. Either image format can be used for the autoencoder, but it is probably easiest using the flattened images.

```python
from PIL import Image
from glob import glob
import numpy as np

def load_images(img_dir, im_width=60, im_height=44):
    """
    Reads, resizes and normalizes the extracted image frames from a folder.

    The images are returned both as a Numpy array of flattened images
    (i.e. the images with the 3-d shape (im_width, im_height, num_channels)
    are reshaped into the 1-d shape (im_width x im_height x num_channels))
    and a list with the images with their original number of dimensions
    suitable for display.

    Arguments
    ---------
    img_dir   : (string) the directory where the images are stored.
    im_width  : (int) The desired width of the image.
                      The default value works well.
    im_height : (int) The desired height of the image.
                      The default value works well.

    Returns
    X       : (numpy.array) An array of the flattened images.
    images  : (list) A list of the resized images.
    """
    images = []
    fnames = glob(f'{img_dir}{os.path.sep}frame*.jpg')
    fnames.sort()

    for fname in fnames:
        im = Image.open(fname)
        # resize the image to im_width and im_height.
        im_array = np.array(im.resize((im_width, im_height)))
        # Convert uint8 to decimal and normalize to 0 - 1.
        images.append(im_array.astype(np.float32) / 255.)
        # Close the PIL image once converted and stored.
        im.close()

    # Flatten the images to a single vector
    X = np.array(images).reshape(-1, np.prod(images[0].shape))

    return X, images
```

Task

You will train an autoencoder to encode the frames in a video. The network should be implemented using Keras.However, you are free to decide on what network architecture to use. I recommended taking a look at the Keras blog post Building Autoencoders in Keras for inspiration and help.

Once the network is trained outlier/anomalous frames can be found by measuring the reconstruction loss and seeing whether it exceeds a certain threshold. The loss can be calculated using `autoencoder.evaluate()`. Remember to reshape the frame to have one row:

```
frame = frame.reshape((1, -1))
loss = autoencoder.evaluate(frame, frame, verbose=0)
```

Use the computed losses to set a threshold that will detect outlying frames. The exact level of the threshold will have to be a rough guess, given the small size size of the video. I recommend you to plot the losses and look at the video to find a reasonable threshold.

Finally, the script should include a function (named `predict`) that predicts whether a new frame is anomalous or not. This function should take as an argument a frame in the original 3-d shape [`shape == (44, 60, 3)`] and return `True` or `False` depending on whether the frame is anomalous or not (see below).

```
def predict(frame):
    """
    Argument
    --------
    frame   : Video frame with shape == (44, 60, 3) and dtype == float.

    Return
    anomaly : A boolean indicating whether the frame is an anomaly or not.
    ------
    """

    # Your fancy computations here!!

    return anomaly
```

Deliverable

You need to submit the trained model and a single Python file (`PY` **NOT** `IPYNB`) that does the following:

1. Loads the model.
2. Contains a function that takes as argument a frame and returns `True` or `False` depending on whether the frame is anomalous or not.

Addtional requirements

- Your code will be tested on a local machine, so no Colab-specific code should be included.
- All paths should be relative and defined at the top of the `PY` file, right below the imports.
- Your code should follow the PEP 8 style guide. See the original PEP 8 style guide, an easier to read version, or a short PEP 8 YouTube intro. Practically, adding a PEP 8 plugin to your text editor (e.g. Flake8) will make it easier to follow to style guide.

Grading

For full marks the submitted code needs to be bug free, execute the two steps described under **Deliverable**, and fullfill the **addtional requirements**. Step two will be tested on new frames that were not part of the original video.

Help

See MMAI 5500 lecture 6 slides.

See the Keras blog about autoencoders for hints about the implementation and the Keras model API for ideas about how to train and get the reconstruction loss for individual frames.

Good luck!