Listing of original (and were noted modified from other source) code used in this project

```
#####################################################
# 1. augmentation.py
# Note: code modified from original in
# https://github.com/naokishibuya/car-behavioral-cloning/blob/master/utils.py
# Available for audit in audit_files/naoki from sharepoint link
#####################################################


import cv2, os
import numpy as np
import matplotlib.image as mpimg
import conf

IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = conf.image_height, conf.image_width, conf.image_
depth
INPUT_SHAPE = (IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)
IMAGE_HEIGHT_NET, IMAGE_WIDTH_NET =  conf.image_height_net, conf.image_width_net

def load_image(image_path):
    """
    Load RGB images from a file
    """
    return mpimg.imread(image_path)


def crop(image):
    """
    Crop the image (removing the sky at the top and the car front at the bottom)
    """
    # this breaks nvidia_baseline
    return image[60:-25, :, :] # remove the sky and the car front


def resize(image):
    """
    Resize the image to the input shape used by the network model
    """
    return cv2.resize(image, (IMAGE_WIDTH_NET, IMAGE_HEIGHT_NET), cv2.INTER_AREA)


def rgb2yuv(image):
    """
    Convert the image from RGB to YUV (This is what the NVIDIA model does)
    """
    return cv2.cvtColor(image, cv2.COLOR_RGB2YUV)


def preprocess(image):
    """
    Combine all preprocess functions into one
    """
    image = crop(image)
```

```
    image = resize(image)
    image = rgb2yuv(image)
    return image

'''
# only one camera in our case
def choose_image(data_dir, center, left, right, steering_angle):
    """
    Randomly choose an image from the center, left or right, and adjust
    the steering angle.
    """
    choice = np.random.choice(3)
    if choice == 0:
        return load_image(data_dir, left), steering_angle + 0.2
    elif choice == 1:
        return load_image(data_dir, right), steering_angle - 0.2
    return load_image(data_dir, center), steering_angle
'''

def random_flip(image, steering_angle):
    """
    Randomly flipt the image left <-> right, and adjust the steering angle.
    """
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 1)
        steering_angle = -steering_angle
    return image, steering_angle


def random_translate(image, steering_angle, range_x, range_y):
    """
    Randomly shift the image vertically and horizontally (translation).
    """
    trans_x = range_x * (np.random.rand() - 0.5)
    trans_y = range_y * (np.random.rand() - 0.5)
    steering_angle += trans_x * 0.002
    trans_m = np.float32([[1, 0, trans_x], [0, 1, trans_y]])
    height, width = image.shape[:2]
    image = cv2.warpAffine(image, trans_m, (width, height))
    return image, steering_angle


def random_shadow(image):
    """
    Generates and adds random shadow
    """
    # (x1, y1) and (x2, y2) forms a line
    # xm, ym gives all the locations of the image
    x1, y1 = IMAGE_WIDTH * np.random.rand(), 0
    x2, y2 = IMAGE_WIDTH * np.random.rand(), IMAGE_HEIGHT
    xm, ym = np.mgrid[0:IMAGE_HEIGHT, 0:IMAGE_WIDTH]

    # mathematically speaking, we want to set 1 below the line and zero otherwise
    # Our coordinate is up side down.  So, the above the line:
```

```
    # (ym-y1)/(xm-x1) > (y2-y1)/(x2-x1)
    # as x2 == x1 causes zero-division problem, we'll write it in the below form:
    # (ym-y1)*(x2-x1) - (y2-y1)*(xm-x1) > 0
    mask = np.zeros_like(image[:, :, 1])
    mask[(ym - y1) * (x2 - x1) - (y2 - y1) * (xm - x1) > 0] = 1

    # choose which side should have shadow and adjust saturation
    cond = mask == np.random.randint(2)
    s_ratio = np.random.uniform(low=0.2, high=0.5)

    # adjust Saturation in HLS(Hue, Light, Saturation)
    hls = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
    hls[:, :, 1][cond] = hls[:, :, 1][cond] * s_ratio
    return cv2.cvtColor(hls, cv2.COLOR_HLS2RGB)


def random_brightness(image):
    """
    Randomly adjust brightness of the image.
    """
    # HSV (Hue, Saturation, Value) is also called HSB ('B' for Brightness).
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    ratio = 1.0 + 0.4 * (np.random.rand() - 0.5)
    hsv[:,:,2] =  hsv[:,:,2] * ratio
    return cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)

def augment(image, steering_angle, range_x=100, range_y=10):
    """
    Generate an augumented image and adjust steering angle.
    (The steering angle is associated with the center image)
    """
    # resize according to expected input shape e.g. AlexNet 224x224, Udacity 320x160, Unity 160x120, etc
    # set in conf.py
    image = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
    #image, steering_angle = choose_image(data_dir, center, left, right, steering_angle)
    image, steering_angle = random_flip(image, steering_angle)
    image, steering_angle = random_translate(image, steering_angle, range_x, range_y)
    image = random_shadow(image)
    image = random_brightness(image)
    return image, steering_angle


######################################################
# 2. Augmentation.py
# Note: code modified from original in
# https://github.com/naokishibuya/car-behavioral-cloning/blob/master/utils.py
# Available for audit in audit_files/naoki from sharepoint link
######################################################


import cv2, os
import numpy as np
import matplotlib.image as mpimg
import conf
```

```
class Augmentation():
    """
    Augmentation methods
    """
    img_dims = []

    def __init__(self, model):
        """
        Set image dimensions for model
        Inputs
            model: string, model name
        """
        self.img_dims = self.get_image_dimensions(model)

    def get_image_dimensions(self, model):
        """
        Get the required dimensions for image model, used for resizing and cropping images
        Inputs
            model: string, name of network model
        Output
            int: IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_WIDTH_NET, IMAGE_HEIGHT_NET, TOP_CROP,
BOTTOM_CROP
        """
        if (model == conf.ALEXNET):
            return conf.alexnet_img_dims
        elif (model == conf.NVIDIA1):
            return conf.nvidia1_img_dims
        elif (model == conf.NVIDIA2):
            return conf.nvidia2_img_dims
        elif (model == conf.NVIDIA_BASELINE):
            return conf.nvidia_baseline_img_dims
        else:
            # default to nvidia1
            return conf.nvidia1_img_dims

    def load_image(self, image_path):
        """
        Load RGB images from a file
        """
        return mpimg.imread(image_path)

    def crop(self, image):
        """
        Crop the image (removing the sky at the top and the car front at the bottom)
        """
        # this breaks nvidia_baseline
        # return image[60:-25, :, :] # remove the sky and the car front
        return image[self.img_dims[conf.IMG_TOP_CROP_IDX]:self.img_dims[conf.IMG_BOTTOM_CROP_IDX],
:,
                :]  # remove the sky and the car front

    def resize(self, image):
```

```
        """
        Resize the image to the input shape used by the network model
        """
        return cv2.resize(image, (self.img_dims[conf.IMG_WIDTH_NET_IDX], self.img_dims[conf.IMG_HEIGHT_
NET_IDX]),
                          cv2.INTER_AREA)

    def resize_expected(self, image):
        """
        Resize the image to the expected original shape
        """
        return cv2.resize(image, (self.img_dims[conf.IMG_WIDTH_IDX], self.img_dims[conf.IMG_HEIGHT_IDX]),
 cv2.INTER_AREA)

    def rgb2yuv(self, image):
        """
        Convert the image from RGB to YUV (This is what the NVIDIA model does)
        """
        return cv2.cvtColor(image, cv2.COLOR_RGB2YUV)

    def preprocess(self, image):
        """
        Combine all preprocess functions into one
        """
        image = self.crop(image)
        image = self.resize(image)
        image = self.rgb2yuv(image)
        return image

    def random_flip(self, image, steering_angle):
        """
        Randomly flipt the image left <-> right, and adjust the steering angle.
        """
        if np.random.rand() < 0.5:
            image = cv2.flip(image, 1)
            steering_angle = -steering_angle
        return image, steering_angle

    def random_translate(self, image, steering_angle, range_x, range_y):
        """
        Randomly shift the image vertically and horizontally (translation).
        """
        trans_x = range_x * (np.random.rand() - 0.5)
        trans_y = range_y * (np.random.rand() - 0.5)
        steering_angle += trans_x * 0.002
        trans_m = np.float32([[1, 0, trans_x], [0, 1, trans_y]])
        height, width = image.shape[:2]
        image = cv2.warpAffine(image, trans_m, (width, height))
        return image, steering_angle

    def random_shadow(self, image):
        """
        Generates and adds random shadow
        """
```

```
        # (x1, y1) and (x2, y2) forms a line
        # xm, ym gives all the locations of the image
        x1, y1 = self.img_dims[conf.IMG_WIDTH_IDX] * np.random.rand(), 0
        x2, y2 = self.img_dims[conf.IMG_WIDTH_IDX] * np.random.rand(), self.img_dims[conf.IMG_HEIGHT_ID
X]
        xm, ym = np.mgrid[0:self.img_dims[conf.IMG_HEIGHT_IDX], 0:self.img_dims[conf.IMG_WIDTH_IDX]]

        # mathematically speaking, we want to set 1 below the line and zero otherwise
        # Our coordinate is up side down.  So, the above the line:
        # (ym-y1)/(xm-x1) > (y2-y1)/(x2-x1)
        # as x2 == x1 causes zero-division problem, we'll write it in the below form:
        # (ym-y1)*(x2-x1) - (y2-y1)*(xm-x1) > 0
        mask = np.zeros_like(image[:, :, 1])
        mask[(ym - y1) * (x2 - x1) - (y2 - y1) * (xm - x1) > 0] = 1

        # choose which side should have shadow and adjust saturation
        cond = mask == np.random.randint(2)
        s_ratio = np.random.uniform(low=0.2, high=0.5)

        # adjust Saturation in HLS(Hue, Light, Saturation)
        hls = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
        hls[:, :, 1][cond] = hls[:, :, 1][cond] * s_ratio
        return cv2.cvtColor(hls, cv2.COLOR_HLS2RGB)

    def random_brightness(self, image):
        """
        Randomly adjust brightness of the image.
        """
        # HSV (Hue, Saturation, Value) is also called HSB ('B' for Brightness).
        hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
        ratio = 1.0 + 0.4 * (np.random.rand() - 0.5)
        hsv[:, :, 2] = hsv[:, :, 2] * ratio
        return cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)

    def augment(self, image, steering_angle, range_x=100, range_y=10):
        """
        Generate an augumented image and adjust steering angle.
        (The steering angle is associated with the center image)
        """
        # resize according to expected input shape e.g. AlexNet 224x224, Udacity 320x160, Unity 160x120, etc
        # set in conf.py
        #image = cv2.resize(image, (self.img_dims[conf.IMG_WIDTH_IDX], self.img_dims[conf.IMG_HEIGHT_ID
X]),
        #                   cv2.INTER_AREA)
        image = self.resize_expected(image)
        # image, steering_angle = choose_image(data_dir, center, left, right, steering_angle)
        image, steering_angle = self.random_flip(image, steering_angle)
        image, steering_angle = self.random_translate(image, steering_angle, range_x, range_y)
        image = self.random_shadow(image)
        image = self.random_brightness(image)
        return image, steering_angle


##############################################################
```

```
# 3. augment.ipynb.py
##################################################


#!/usr/bin/env python
# coding: utf-8

# In[1]:


# start jupyter notebooks from prompt to load all required libraries
# jupyter notebook
import cv2


# In[3]:


import matplotlib.image as mpimg
fp = '/home/simbox/git/msc-data/unity/log2/logs_Fri_Jul_10_09_16_18_2020/10000_cam-image_array_.jpg'
# fp = '/home/simbox/Downloads/IMG/center_2020_11_10_22_02_48_622.jpg'
img = mpimg.imread(fp)
# adapt to naoki net, we have 160w x 120h, first scale to 200
img = cv2.resize(img, (320,160), cv2.INTER_AREA)
import matplotlib.pyplot as plt
plt.imshow(img)
#plt.imshow(img)\n",
# plt.imshow(img[61:-25, :, :]) # image is 120h160w3d: 60:-25 ~ start at h pixel index 60, end at index (120) - 25
                        # equivalent to img[60:95, :, :]
                        # plain english: remove sky and car shadow
# print(img[59:-25, :, :].shape)
   # img[50:-25, :, :].shape"


# In[8]:


# first resize
image = load_image(fp)
image = cv2.resize(image, (320, 160), cv2.INTER_AREA)
image = crop(image)
plt.imshow(image)


# In[33]:


image = load_image(fp)
image = cv2.resize(image, (320, 160), cv2.INTER_AREA)
steering_angle = 0.07
# image, steering_angle = augment(image, steering_angle)
# image = crop(image)
# image = crop(image)
# image = resize(image)
```

```
# image = rgb2yuv(image)
# print(steering_angle)
plt.imshow(image)



# In[2]:


import cv2, os
import numpy as np
import matplotlib.image as mpimg

# Udacity
IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 160, 320, 3
# Alexnet
#IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 224, 224, 3
# IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 66, 200, 3
# IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 120, 160, 3
INPUT_SHAPE = (IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)
# Dimensions expected by network
# Udacity
IMAGE_WIDTH_NET, IMAGE_HEIGHT_NET = 200, 66
# Alexnet
# IMAGE_WIDTH_NET, IMAGE_HEIGHT_NET = 224, 224



def load_image(image_path):
    """
    Load RGB images from a file
    """
    return mpimg.imread(image_path)


def crop(image):
    """
    Crop the image (removing the sky at the top and the car front at the bottom)
    """
    # unity
    # return image[60:-25, :, :] # remove the sky and the car front
    # alexnet
    return image[109:-40, :, :] # remove the sky and the car front


def resize(image):
    """
    Resize the image to the input shape used by the network model
    """
    return cv2.resize(image, (IMAGE_WIDTH_NET, IMAGE_HEIGHT_NET), cv2.INTER_AREA)


def rgb2yuv(image):
    """
```

```
    Convert the image from RGB to YUV (This is what the NVIDIA model does)
    """
    return cv2.cvtColor(image, cv2.COLOR_RGB2YUV)


def preprocess(image):
    """
    Combine all preprocess functions into one
    """
    image = crop(image)
    image = resize(image)
    image = rgb2yuv(image)
    return image


def choose_image(data_dir, center, left, right, steering_angle):
    """
    Randomly choose an image from the center, left or right, and adjust
    the steering angle.
    """
    choice = np.random.choice(3)
    if choice == 0:
        return load_image(data_dir, left), steering_angle + 0.2
    elif choice == 1:
        return load_image(data_dir, right), steering_angle - 0.2
    return load_image(data_dir, center), steering_angle


def random_flip(image, steering_angle):
    """
    Randomly flipt the image left <-> right, and adjust the steering angle.
    """
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 1)
        steering_angle = -steering_angle
    return image, steering_angle


def random_translate(image, steering_angle, range_x, range_y):
    """
    Randomly shift the image virtially and horizontally (translation).
    """
    trans_x = range_x * (np.random.rand() - 0.5)
    trans_y = range_y * (np.random.rand() - 0.5)
    steering_angle += trans_x * 0.002
    trans_m = np.float32([[1, 0, trans_x], [0, 1, trans_y]])
    height, width = image.shape[:2]
    image = cv2.warpAffine(image, trans_m, (width, height))
    return image, steering_angle


def random_shadow(image):
    """
    Generates and adds random shadow
```

```
    """
    # (x1, y1) and (x2, y2) forms a line
    # xm, ym gives all the locations of the image
    x1, y1 = IMAGE_WIDTH * np.random.rand(), 0
    x2, y2 = IMAGE_WIDTH * np.random.rand(), IMAGE_HEIGHT
    # could this be a bug?
    xm, ym = np.mgrid[0:IMAGE_HEIGHT, 0:IMAGE_WIDTH]
    # xm, ym = np.mgrid[0:IMAGE_WIDTH, 0:IMAGE_HEIGHT]

    # mathematically speaking, we want to set 1 below the line and zero otherwise
    # Our coordinate is up side down.  So, the above the line:
    # (ym-y1)/(xm-x1) > (y2-y1)/(x2-x1)
    # as x2 == x1 causes zero-division problem, we'll write it in the below form:
    # (ym-y1)*(x2-x1) - (y2-y1)*(xm-x1) > 0
    mask = np.zeros_like(image[:, :, 1])
    mask[(ym - y1) * (x2 - x1) - (y2 - y1) * (xm - x1) > 0] = 1

    # choose which side should have shadow and adjust saturation
    cond = mask == np.random.randint(2)
    s_ratio = np.random.uniform(low=0.2, high=0.5)

    # adjust Saturation in HLS(Hue, Light, Saturation)
    hls = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
    hls[:, :, 1][cond] = hls[:, :, 1][cond] * s_ratio
    return cv2.cvtColor(hls, cv2.COLOR_HLS2RGB)


def random_brightness(image):
    """
    Randomly adjust brightness of the image.
    """
    # HSV (Hue, Saturation, Value) is also called HSB ('B' for Brightness).
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    ratio = 1.0 + 0.4 * (np.random.rand() - 0.5)
    hsv[:,:,2] =  hsv[:,:,2] * ratio
    return cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)


def augment(image, steering_angle, range_x=100, range_y=10):
    """
    Generate an augumented image and adjust steering angle.
    (The steering angle is associated with the center image)
    """
    # resize - we start with assumed image capture size
    image = cv2.resize(image, (320,160), cv2.INTER_AREA)
    # image = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
    image, steering_angle = random_flip(image, steering_angle)
    image, steering_angle = random_translate(image, steering_angle, range_x, range_y)
    image = random_shadow(image)
    image = random_brightness(image)
    return image, steering_angle


def batch_generator(data_dir, image_paths, steering_angles, batch_size, is_training):
```

```
    """
    Generate training image give image paths and associated steering angles
    """
    images = np.empty([batch_size, IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS])
    steers = np.empty(batch_size)
    while True:
        i = 0
        for index in np.random.permutation(image_paths.shape[0]):
            center, left, right = image_paths[index]
            steering_angle = steering_angles[index]
            # argumentation
            if is_training and np.random.rand() < 0.6:
                image, steering_angle = augument(data_dir, center, left, right, steering_angle)
            else:
                image = load_image(data_dir, center)
            # add the image and steering angle to the batch
            images[i] = preprocess(image)
            steers[i] = steering_angle
            i += 1
            if i == batch_size:
                break
        yield images, steers


# In[11]:


# cropping test nvidia1

def crop(image, top_crop, bot_crop):
    """
    Crop the image (removing the sky at the top and the car front at the bottom)
    """
    # unity
    return image[top_crop:bot_crop, :, :] # remove the sky and the car front
    # alexnet
    # return image[100:-50, :, :] # remove the sky and the car front

import matplotlib.pyplot as plt
IMAGE_WIDTH, IMAGE_HEIGHT = 160, 120
fp = '/home/simbox/git/msc-data/unity/log2/logs_Fri_Jul_10_09_16_18_2020/10000_cam-image_array_.jpg'
image = load_image(fp)
image_resized = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
# plt.imshow(image)
plt.imshow(image_resized)
print(image_resized.shape)
print("Expected (120, 160, 3) nvidia1")


# In[12]:


image_nvidia1_crop = crop(image_resized, 60, -25)
# cropping test
```

```
print("nvidia1 crop (35, 160, 3)")
print(image_nvidia1_crop.shape)
plt.imshow(image_nvidia1_crop)


# In[14]:


image_nvidia_crop_resized = cv2.resize(image_nvidia1_crop, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_
AREA)
print(image_nvidia_crop_resized.shape)
print("nvidia1 resized")
plt.imshow(image_nvidia_crop_resized)


# In[25]:


# cropping test nvidia2

import matplotlib.pyplot as plt
IMAGE_WIDTH, IMAGE_HEIGHT = 320, 160
image_resized = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
# plt.imshow(image)
plt.imshow(image_resized)
print(image_resized.shape)
print("Expected (160, 320, 3) nvidia1")


# In[26]:


image_nvidia2_crop_70_35 = crop(image_resized, 70, -35)
# cropping test
print("nvidia2 70 -35 crop (55, 320, 3)")
print(image_nvidia2_crop_70_35.shape)
plt.imshow(image_nvidia2_crop_70_35)


# In[27]:


image_nvidia_crop_70_35_resized = cv2.resize(image_nvidia2_crop_70_35, (IMAGE_WIDTH, IMAGE_HEIGHT
), cv2.INTER_AREA)
print(image_nvidia_crop_70_35_resized.shape)
print("nvidia1 crop 70 -35 resized")
plt.imshow(image_nvidia_crop_70_35_resized)


# In[20]:


image_nvidia2_crop_91_35 = crop(image_resized, 91, -35)
# cropping test
```

```
print("nvidia2 91 -35 crop (35, 160, 3)")
print(image_nvidia2_crop_91_35.shape)
plt.imshow(image_nvidia2_crop_91_35)


# In[35]:


IMAGE_WIDTH, IMAGE_HEIGHT = 320, 160
image_resized = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
# plt.imshow(image)
plt.imshow(image_resized)
print(image_resized.shape)
print("Expected (160, 320, 3) nvidia2")


# In[36]:


image_nvidia2_crop_77_35 = crop(image_resized, 77, -35)
# cropping test
print("nvidia2 77 -35 crop (55, 320, 3)")
print(image_nvidia2_crop_77_35.shape)
plt.imshow(image_nvidia2_crop_77_35)


# In[37]:


image_resized = cv2.resize(image_nvidia2_crop_77_35, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_ARE
A)
# plt.imshow(image)
plt.imshow(image_resized)
print(image_resized.shape)
print("Expected (160, 320, 3) nvidia2")


# In[21]:


# cropping test nvidia2 - higher crop to include all road markings


import matplotlib.pyplot as plt
IMAGE_WIDTH, IMAGE_HEIGHT = 320, 160
image_resized = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
# plt.imshow(image)
plt.imshow(image_resized)
print(image_resized.shape)
print("Expected (160, 320, 3) nvidia1")


# In[32]:
```

```
image_nvidia2_crop_81_35 = crop(image_resized,81, -35)
# cropping test
print("nvidia2 81 -35 crop (35, 160, 3)")
print(image_nvidia2_crop_81_35.shape)
plt.imshow(image_nvidia2_crop_81_35)


# In[33]:


IMAGE_WIDTH, IMAGE_HEIGHT = 320, 160
image_resized = cv2.resize(image_nvidia2_crop_81_35, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_ARE
A)
# plt.imshow(image)
plt.imshow(image_resized)
print(image_resized.shape)
print("Expected (160, 320, 3) nvidia2")


# In[34]:


IMAGE_WIDTH, IMAGE_HEIGHT = 320, 160
image_resized = cv2.resize(image_nvidia2_crop_81_35, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_ARE
A)
image_nvidia2_crop_77_35 = crop(image_resized,77, -35)
# cropping test
print("nvidia2 77 -35 crop (35, 160, 3)")
print(image_nvidia2_crop_77_35.shape)
plt.imshow(image_nvidia2_crop_77_35)


# In[64]:


# images for dissertation - Augmentation section
# Finally move from RGB to YUV colour space
image = load_image(fp)
image = cv2.resize(image, (320, 160), cv2.INTER_AREA)
steering_angle = 0.07
image, steering_angle = augment(image, steering_angle)
image = crop(image)
image = resize(image)
image = rgb2yuv(image)
# print(steering_angle)
plt.imshow(image)
print(image.shape) # original size of this image: 120 x 160


# In[103]:


# images for dissertation - Augmentation section
```

```
image = load_image(fp)
image1 = cv2.resize(image, (320, 160), cv2.INTER_AREA) # original NVIDIA capture size
steering_angle = 0.07
image2, steering_angle = augment(image1, steering_angle) # augmented
image3 = crop(image2) # cropped
image4 = resize(image3) # resized to network design
image5 = rgb2yuv(image4) # RGB to YUV transform
# print(steering_angle)
#plt.imshow(image)
#print(image.shape) # original size of this image: 120 x 160

# plot for dissertation - cannot adjust padding between rows so further image processing required
import matplotlib.pyplot as plt
import numpy as np
import matplotlib
matplotlib.rcParams['font.size'] = 12.0

fig, axs = plt.subplots(2, 3)

fig.set_figheight(15)
fig.set_figwidth(15)

# Raw
axs[0, 0].title.set_text('Raw ' + str(image1.shape))
axs[0, 0].imshow(image1)

# Augmented
axs[0, 1].title.set_text('Augmented ' + str(image2.shape))
axs[0, 1].imshow(image2)

# cropped
axs[0, 2].title.set_text('Cropped ' + str(image3.shape))
axs[0, 2].imshow(image3)

# resized
axs[1, 0].title.set_text('Resized ' + str(image4.shape))
axs[1, 0].imshow(image4)

# RGBtoYUV
axs[1, 1].title.set_text('RGB to YUV ' + str(image5.shape))
axs[1, 1].imshow(image5)

# Dummy to better format
axs[1, 2].title.set_text('RGB to YUV ' + str(image5.shape))
axs[1, 2].imshow(image5)


plt.show()


# In[91]:


import numpy as np
```

```
import matplotlib.pyplot as plt
from matplotlib import gridspec

nrow = 2
ncol = 3

fig = plt.figure(figsize=(10, 10))

gs = gridspec.GridSpec(nrow, ncol, width_ratios=[1, 1, 1],
         wspace=0.0, hspace=0.0, top=0.95, bottom=0.05, left=0.17, right=0.845)

for i in range(2):
    for j in range(3):
        im = np.random.rand(28,28)
        ax= plt.subplot(gs[i,j])
        ax.imshow(image)
        ax.set_xticklabels([])
        ax.set_yticklabels([])

#plt.tight_layout() # do not use this!!
plt.show()



#####################################################
# 4. conf.py
# Note: code based on
# https://github.com/tawnkramer/sdsandbox/blob/master/src/conf.py
# Available for audit in audit_files/tawn from sharepoint link
#####################################################


import math

training_patience = 6
training_default_epochs = 100
training_default_aug_mult = 1
training_default_aug_percent = 0.0
learning_rate = 0.00001 # 0.00001

# default model name
# model_name = 'nvidia1'
#nvidia 1 - use this size for both
#nvidia 1 and 2, just change nvidia size before
# presenting to neural network
#image_width = 160
#image_height = 120
#nvidia 2

# size augmentation process is expecting, i.e. what came from camera
# AlexNet 224x224, Udacity 320x160, Unity 160x120, etc

# IMAGE DIMS INDEXES
# expected original image size
```

```
IMG_WIDTH_IDX = 0
IMG_HEIGHT_IDX = 1
IMG_DEPTH_IDX = 2
# size to be presented to network
IMG_WIDTH_NET_IDX = 3
IMG_HEIGHT_NET_IDX = 4
# to crop road from image
IMG_TOP_CROP_IDX = 5
IMG_BOTTOM_CROP_IDX = 6

# What these lists mean:
# Expected width, height and depth of acquired image,
# width and height of image expected by network
# top crop and bottom crop to remove car and sky from image
# ALEXNET
ALEXNET = 'alexnet'
alexnet_img_dims = [224,224,3,224,224,60,-25]
# NVIDIA1
NVIDIA1 = 'nvidia1' # a.k.a. TawnNe,
nvidia1_img_dims = [160,120,3,160,120,60,-25]
# NVIDIA2
NVIDIA2 = 'nvidia2' # a.k.a. NaokiNet
nvidia2_img_dims = [320,160,3,200,66,81,-35]
# NVIDIA_BASELINE
NVIDIA_BASELINE = 'nvidia_baseline' # a.k.a. NaokiNet
nvidia_baseline_img_dims = [160,120,3,200,66,60,-25]

# Alexnet
image_width_alexnet = 224
image_height_alexnet = 224
# nvidia
image_width = 160
image_height = 120
#nvidia2 (Udacity NaokiNet)
#image_width = 160
#image_height = 120

# size network is expecting
image_width_net = 160
image_height_net = 120
# same for all
image_depth = 3

row = image_height_net
col = image_width_net
ch = image_depth

# training for steering and throttle:
num_outputs = 2
# steering alone:
# num_outputs = 1

throttle_out_scale = 1.0
# alexnet
```

```
batch_size = 64

# Using class members to avoid passing same parameters through various functions
# The original NVIDIA paper mentions augmentation but no cropping i.e. road only
# augmentation
aug = False
# pre-process image: crop, resize and rgb2yuv
preproc = False
# image normalization constant, Unity model maximum steer
norm_const = 25

# rain type and slant
rt = ''
st = 0

# video recording
VIDEO_WIDTH, VIDEO_HEIGHT = 800, 600
IMAGE_STILL_WIDTH, IMAGE_STILL_HEIGHT = 800, 600
record = False

def setdims(modelname):
    """
    Set image dimensions for training and predicting
    Inputs
        modelname: string, network name
    Outputs
        none
    Example
    setdims('alexnet') # set width and height to 224
    """
    if(modelname=='alexnet'):
        conf.row = image_width_alexnet
        self.col = image_height_alexnet


#####################################################
# 5. data_predict.py
#####################################################


# Predict steering angles for a dataset for which a ground truth exists
# dsikar@gmail.com

from __future__ import print_function

import argparse
import fnmatch
import json
import os
import pickle
import random
from datetime import datetime
from time import strftime
import numpy as np
```

```
from PIL import Image
from tensorflow import keras
import tensorflow as tf
import conf
import models
from helper_functions import hf_mkdir
from augmentation import augment, preprocess
import cv2
from train import load_json, get_files
from augmentation import preprocess
from utils.steerlib import gos, plotSteeringAngles
from pathlib import Path

from tensorflow.python.keras.models import load_model



# 1. get a list of files to predict (sequential)
# 2. read the steering angle (json file)
# 3. generate a prediction
# 4. Store results in list
# 5. Generate a "goodness of steer" value (average steering error)
# 6. Generate graph

def predict_drive(datapath, modelpath, nc):
    """
    Generate predictions from a model for a dataset
    Inputs
        datapath: string, path to data
        modelpath: string, path to trained model
        nc: steering angle normalization constant
    """
    print("loading model", modelpath)
    model = load_model(modelpath)

    # In this mode, looks like we have to compile it
    # NB this is a bit tricky, do need to use optimizer and loss function used to train model?
    model.compile("sgd", "mse")

    files = get_files(datapath, True)
    outputs = []
    for fullpath in files:
        frame_number = os.path.basename(fullpath).split("_")[0]
        json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
        data = load_json(json_filename)
        # ground truth
        steering = float(data["user/angle"]) # normalized - divided by nc by simulator
        # prediction
        image = cv2.imread(fullpath)
        # The image will be 1. resized to expected pre-processing size and 2.resized to expected
        # size to be presented to network. This is network architecture and dataset dependant and
        # currently managed in conf.py
        image = preprocess(image)
        image = image.reshape((1,) + image.shape)
```

```
        mod_pred = model.predict(image)
        # append prediction and ground truth to list
        outputs.append([mod_pred[0][1], steering])
    # get goodness of steer
    sarr = np.asarray(outputs)
    p = sarr[:, 0]
    g = sarr[:, 1]
    gs = gos(p,g,nc)
    print(gs)
    # def plotSteeringAngles(p, g=None, n=1, save=False, track= "Track Name", mname="model name", title='title'):
    gss = "{:.2f}".format(gs)
    modelpath = modelpath.split('/')
    datapath = datapath.split('/')
    plotSteeringAngles(p, g, nc, True, datapath[-2], modelpath[-1], 'Gs ' + gss)


# dataset ../dataset/unity/jungle1/
# model ../trained_models/nvidia2/20201124032017_nvidia2.h5
    # calculate gos (average steering error)
    # plot graph unormalized angles. + average steering error
    # save graph.
    # done

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='prediction server')
    parser.add_argument('--datapath', type=str, default='/home/simbox/git/msc-data/unity/genTrackOneLap_3/*.jpg',
help='model filename')
    parser.add_argument('--modelpath', type=str, default='/home/simbox/git/sdsandbox/trained_models/sanity/202011
20171015_sanity.h5', help='Model')
    parser.add_argument('--nc', type=int, default=1, help='Steering Angle Normalization Constant')
    # time allowing, set image sizes based on model name. For now, these have to be managed in conf.py
    #parser.add_argument('--model', type=str, default='nvidia1', help='model name')

    # set dimensions

    args = parser.parse_args()

    predict_drive(args.datapath, args.modelpath, args.nc)
    # max value for slant is 20
    # Example
    # python3 predict_client.py --model=../trained_models/sanity/20201120171015_sanity.h5 --rain=light --slant=0
```

```
#####################################################
# 6. GetSteeringAnglesFromtcpflow.ipynb.py
#####################################################


#!/usr/bin/env python
```

```
# coding: utf-8

# In[22]:


import json
import numpy as np
import matplotlib.pyplot as plt
import statistics
import seaborn as sns
import os
import fnmatch

def GetSteeringFromtcpflow(filename):
    """
    Get a tcpflow log and extract steering values obtained from network communication between.
    Note, we only plot the predicted steering angle jsondict['steering']
    and the value of jsondict['steering_angle'] is ignored. Assumed to be the steering angle
    calculated by PID given the current course.
    sim and prediction engine (predict_client.py)
    Inputs
        filename: string, name of tcpflow log
    Returns
        sa: list of arrays, steering angle predicton and actual value tuple.
    Example


    """
    # open file
    sa = []
    # initialize prediction
    pred = ''
    f = open(filename, "r")
    file = f.read()
    try:
        #readline = f.read()
        lines = file.splitlines()
        for line in lines:
            # print(line)
            start = line.find('{')
            if(start == -1):
                continue
            jsonstr = line[start:]
            # print(jsonstr)
            jsondict = json.loads(jsonstr)
            if "steering" in jsondict:
                # predicted
                pred = jsondict['steering']
                # jsondict['steering_angle']
                # sa.append([float(pred), act])
                sa.append([float(pred), float(pred)]) # append twice to keep code from breaking
            #if "steering_angle" in jsondict:
                # actual
             #   act = jsondict['steering_angle']
```

```
            # save pair, only keep last pred in case two were send as it does happen i.e.:
            # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.071960375", "t
hrottle": "0.08249988406896591", "brake": "0.0"}
            # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.079734944", "t
hrottle": "0.08631626516580582", "brake": "0.0"}
            # 127.000.000.001.09091-127.000.000.001.59460: {"msg_type":"telemetry","steering_angle":-0.0719603
7,(...)
          #   if(len(pred) > 0):
           #      sa.append([float(pred), act])
           #      pred = '' # need to save this image
             # deal with image later, sort out plot first
             #imgString = jsondict["image"]
             #image = Image.open(BytesIO(base64.b64decode(imgString)))
             #img_arr = np.asarray(image, dtype=np.float32)
    except Exception as e:
        print("Exception raise: " + str(e))
    # file should be automatically closed but will close for good measure
    f.close()
    return sa

def plotBinsFromArray(svals, nc=25, pname=None, logname = "tcpflow log name"):
    svalscp = [element * nc for element in svals]
    values = len(svalscp)
    mymean = ("%.2f" % statistics.mean(svalscp))
    mystd = ("%.2f" % statistics.stdev(svalscp))
    plt.title=(pname)
    # NB Plotted as normalized histogram
    sns.distplot(svalscp, bins=nc*2, kde=False, norm_hist=True,
    axlabel=  "tcpflow: " + logname + ", model: " + pname + ' steer. degs. norm. hist. ' + str(values) + ' values, mean =
 ' + mymean + ' std = ' + mystd)
    #if(save):
    #    sns.save("output.png")
    plt.savefig(pname + '.png')
# Steering angle predictions by model 20201107210627_nvidia1.h5

def plotSteeringAngles(p, g, n, save=False, track= "Track Name", mname="model.h5"):
    """
    Plot predicted steering angles
    """
    plt.rcParams["figure.figsize"] = (18,3)

    plt.plot(p*25)
    # plt.plot(sarr[:,1]*25, label="simulator")

    plt.ylabel('Steering angle')
    plt.xlabel('Frame number')
    # Set a title of the current axes.
    mytitle = 'tcpflow log predicted steering angles: track ' + str(track) + ' model ' + str(mname)
    plt.title(mytitle)
    # show a legend on the plot
    #plt.legend()
    # Display a figure.
    # horizontal grid only
    plt.grid(axis='y')
```

```
    # set limit
    plt.xlim([-5,len(p)+5])
    plt.gca().invert_yaxis()
    plt.show()

def plotMultipleSteeringAngles(p, n, save=False, track= "Track Name", mname="model.h5", w=18, h=3):
    """
    Plot multiple predicted steering angles
    Inputs
        p: list of tuples, steering angles and labels
        n: integer,
    """
    plt.rcParams["figure.figsize"] = (18,3)


    plt.plot(p*25)
    # plt.plot(sarr[:,1]*25, label="simulator")

    plt.ylabel('Steering angle')
    plt.xlabel('Frame number')
    # Set a title of the current axes.
    mytitle = 'tcpflow log predicted steering angles: track ' + str(track) + ' model ' + str(mname)
    plt.title(mytitle)
    # show a legend on the plot
    #plt.legend()
    # Display a figure.
    # horizontal grid only
    plt.grid(axis='y')
    # set limit
    plt.xlim([-5,len(p)+5])
    plt.gca().invert_yaxis()
    plt.show()

def GetJSONSteeringAngles(filemask):
    """
    Get steering angles stored as 'user/angle' attributes in .json files
    Inputs:
        filemask: string, path and mask
    Outputs
        svals: list, steering values
    """
    filemask = os.path.expanduser(filemask)
    path, mask = os.path.split(filemask)

    matches = []
    for root, dirnames, filenames in os.walk(path):
        for filename in fnmatch.filter(filenames, mask):
            matches.append(os.path.join(root, filename))
    # sort by create date
    matches = sorted(matches, key=os.path.getmtime)
    # steering values
    svals = []
    for fullpath in matches:
            frame_number = os.path.basename(fullpath).split("_")[0]
```

```
        json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
        jobj = load_json(json_filename)
        svals.append(jobj['user/angle'])
    return svals

def load_json(filepath):
    """
    Load a json file
    Inputs
        filepath: string, path to file
    Outputs
        data: dictionary, json key, value pairs
    Example
    path = "~/git/msc-data/unity/roboRacingLeague/log/logs_Sat_Nov_14_12_36_16_2020/record_11640.json"
    js = load_json(path)
    """
    with open(filepath, "rt") as fp:
        data = json.load(fp)
    return data


# In[2]:


sa = GetSteeringFromtcpflow('../trained_models/nvidia2/tcpflow/20201207124146_nvidia2_tcpflow.log')
sarr = np.asarray(sa)
p = sarr[:,0]
g = sarr[:,0]
plotSteeringAngles(p, g, 25, False, "Generated Track", "20201207124146_nvidia2.h5")
plotBinsFromArray(p, 25, "20201207124146_nvidia2.h5", "20201207124146_nvidia2_tcpflow.log")


# In[28]:


# plot ground truth steering angles for
filemask = '../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020/*.jpg'
g = GetJSONSteeringAngles(filemask)
# print(type(g)) # list
g = np.asarray(g)
# print(type(g)) # <class 'numpy.ndarray'>
plt.rcParams["figure.figsize"] = (18,3)
nc = 25 # norm. constant, maximum steering angle

plt.plot(g*nc)
# plt.plot(sarr[:,1]*25, label="simulator")

plt.ylabel('Steering angle')
plt.xlabel('Frame number')
# Set a title of the current axes.
mytitle = 'Ground truth steering for logs_Wed_Nov_25_23_39_22_2020 - One lap recorded from Generated Track i
n "Auto Drive w Rec" mode'
plt.title(mytitle)
plt.grid(axis='y')
```

```
# set limit
plt.xlim([-5,len(g)+5])
plt.gca().invert_yaxis()
plt.show()


# In[2]:


# Steering angle predictions by model 20201107210627_nvidia1.h5
def plotSteeringAngles(p, g, n, save=False, track= "Track Name", mname="model.h5"):
    """
    Plot predicted steering angles
    """
    import matplotlib.pyplot as plt

    plt.rcParams["figure.figsize"] = (18,3)

    plt.plot(p*25)
    # plt.plot(sarr[:,1]*25, label="simulator")

    plt.ylabel('Steering angle')
    plt.ylabel('Frame number')
    # Set a title of the current axes.
    plt.title('tcpflow log predicted steering angles: track ' + track + ' model ' + mname)
    # show a legend on the plot
    #plt.legend()
    # Display a figure.
    # horizontal grid only
    plt.grid(axis='y')
    # set limit
    plt.xlim([-5,len(p)+5])
    plt.gca().invert_yaxis()
    plt.show()

sa = GetSteeringFromtcpflow('../dataset/unity/genRoad/tcpflow/20201120184912_sanity.log')
sarr = np.asarray(sa)
p = sarr[:,0]
g = sarr[:,1]

plotSteeringAngles(p, g, 25, False, "Generated Road, ", "20201120184912_sanity.h5")
#sa = GetSteeringFromtcpflow('../dataset/unity/genRoad/tcpflow/20201120184912_sanity.log')
#plotBinsFromArray(p, 25, "20201120184912_sanity.h5", "tcpflow/20201120184912_sanity.log")
#type(g)

# TypeError: 'str' object is not callable - cleared by restarting kernel (clearing variables?)


# In[3]:


def gos(p, g, n):
    """
    Calculate the goodness-of-steer between a prediction and a ground truth array.
```

```
    Inputs
        p: array of floats, steering angle prediction
        g: array of floats, steering angle ground truth.
        n: float, normalization constant
    Output
        gos: float, average of absolute difference between ground truth and prediction arrays
    """
    # todo add type assertion
    assert len(p) == len(g), "Arrays must be of equal length"
    return sum(abs(p - g)) / len(p) * n

#p = sarr[:,0]
#g = sarr[:,1]

#sterr = gos(p,g, 25)

#print("Goodness of steer: {:.2f}".format(sterr))


# In[4]:


import statistics
import matplotlib.pyplot as plt
import seaborn as sns
def plotBinsFromArray(svals, nc=25, pname=None, logname = "tcpflow log name"):
    svalscp = [element * nc for element in svals]
    values = len(svalscp)
    mean = ("%.2f" % statistics.mean(svalscp))
    std = ("%.2f" % statistics.stdev(svalscp))
    plt.title=(pname)
    # NB Plotted as normalized histogram
    sns.distplot(svalscp, bins=nc*2, kde=False, norm_hist=True,
    axlabel=  "tcpflow: " + logname + ", model: " + pname + ' steer. degs. norm. hist. ' + str(values) + ' values, mean =
 ' + mean + ' std = ' + std)
    #if(save):
    #    sns.save("output.png")
    plt.savefig(pname + '.png')

#sa = GetSteeringFromtcpflow('../dataset/unity/genRoad/tcpflow/20201120184912_sanity.log')
#sarr = np.asarray(sa)
#p = sarr[:,0]
#p = sarr[:,0]
#plotBinsFromArray(p, 25, "20201120184912_sanity.h5", "tcpflow/20201120184912_sanity.log")


# In[5]:


#### ../dataset/unity/genRoad/tcpflow/20201123162643_sanity.log
sa = GetSteeringFromtcpflow('../dataset/unity/genRoad/tcpflow/20201120184912_sanity.log')
sarr = np.asarray(sa)
p = sarr[:,0]
p = sarr[:,0]
```

```
plotSteeringAngles(p, g, 25, False, "Generated Road", "20201120184912_sanity.h5")
plotBinsFromArray(p, 25, "20201120184912_sanity.h5", "tcpflow/20201123162643_sanity.log")


# In[6]:


#### ../dataset/unity/genRoad/tcpflow/20201123162643_sanity.log
sa = GetSteeringFromtcpflow('../dataset/unity/genRoad/tcpflow/20201123162643_sanity.log')
sarr = np.asarray(sa)
p = sarr[:,0]
p = sarr[:,0]
plotSteeringAngles(p, g, 25, False, "Generated Road", "20201123162643_sanity.h5")
plotBinsFromArray(p, 25, "20201123162643_sanity.h5", "tcpflow/20201123162643_sanity.log")


# In[ ]:


#### ../dataset/unity/genRoad/tcpflow/20201123162643_sanity_pp.log
sa = GetSteeringFromtcpflow('../dataset/unity/genRoad/tcpflow/20201123162643_sanity_pp.log')
sarr = np.asarray(sa)
p = sarr[:,0]
p = sarr[:,0]
plotSteeringAngles(p, g, 25, False, "Generated Road", "20201123162643_sanity_pp.h5")
plotBinsFromArray(p, 25, "20201123162643_sanity.h5", "tcpflow/20201123162643_sanity_pp.log")


# In[7]:


sa = GetSteeringFromtcpflow('../trained_models/nvidia2/tcpflow/20201207111940_nvidia2_tcpflow.log')
sarr = np.asarray(sa)
p = sarr[:,0]
p = sarr[:,0]
plotSteeringAngles(p, g, 25, False, "Generated Track", "20201207111940_nvidia2.h5")
plotBinsFromArray(p, 25, "20201207111940_nvidia2.h5", "20201207111940_nvidia2_tcpflow.log")


# In[ ]:




##################################################
# 7. GetSteering.py
##################################################


import argparse
import fnmatch
import json
```

```
import os
from io import BytesIO
from PIL import Image
import base64
import numpy as np
import matplotlib.pyplot as plt
from augmentation import preprocess

def GetSteering(filename):
    """
    Get a tcpflow log and extract steering values received from and sent to sim
    Inputs
        filename: string, name of tcpflow log
    """
    # open file
    sa = []
    # initialize prediction
    pred = ''
    f = open(filename, "r")
    file = f.read()
    try:
        #readline = f.read()
        lines = file.splitlines()
        for line in lines:
            #print(line)
            start = line.find('{')
            if(start == -1):
                continue
            jsonstr = line[start:]
            #print(jsonstr)
            jsondict = json.loads(jsonstr)
            if "steering" in jsondict:
                # predicted
                pred = jsondict['steering']
            if "steering_angle" in jsondict:
                # actual
                act = jsondict['steering_angle']
                # save pair, only keep last pred in case two were send as it does happen i.e.:
                # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.071960375", "t
hrottle": "0.08249988406896591", "brake": "0.0"}
                # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.079734944", "t
hrottle": "0.08631626516580582", "brake": "0.0"}
                # 127.000.000.001.09091-127.000.000.001.59460: {"msg_type":"telemetry","steering_angle":-0.0719603
7,(...)
                if(len(pred) > 0):
                    sa.append([float(pred), act])
                    pred = '' # need to save this image
                # deal with image later, sort out plot first
                    imgString = jsondict["image"]
                    image = Image.open(BytesIO(base64.b64decode(imgString)))
                    img_arr = np.asarray(image, dtype=np.float32)
                    img_arr_proc = preprocess(img_arr)
                    stitch = stitchImages(img_arr, img_arr_proc, 160, 120)
                    plt.imshow(stitch)
```

```
    except Exception as e:
        print("Exception raise: " + str(e))
    # file should be automatically closed but will close for good measure
    f.close()


def stitchImages(a, b, w, h):
    """
    Stitch two images together side by side
    Inputs
        a, b: floating point image arrays
        w, h: integer width and height dimensions
    Output
        c: floating point stitched image array
    """
    # https://stackoverflow.com/questions/30227466/combine-several-images-horizontally-with-python
    total_width = w * 2
    max_height = h

    a = Image.fromarray(a.astype('uint8'), 'RGB')
    b = Image.fromarray(b.astype('uint8'), 'RGB')

    new_im = Image.new('RGB', (total_width, max_height))
    new_im.paste(a, (0,0))
    new_im.paste(b, (w,0))

    return new_im # new_im



if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='train script')
    parser.add_argument('--filename', type=str, help='tcpflow log')
    args = parser.parse_args()
    GetSteering(args.filename)

######################################################
# 8. jsonclean.py
######################################################


"""
Go though Unity3D generated files, delete .jpg if a corresponding .json file does not exist
$ python3 jsonclean.py --inputs=../../dataset/unity/log/*.jpg --delete=[False|True]
"""

import os
import argparse
import fnmatch
import json


def load_json(filename):
    with open(filename, "rt") as fp:
```

```
        data = json.load(fp)
    return data

def cleanjson(filemask, delete):
    # filemask = '~/git/sdsandbox/dataset/unity/log/*.jpg'
    filemask = os.path.expanduser(filemask)
    path, mask = os.path.split(filemask)

    matches = []
    for root, dirnames, filenames in os.walk(path):
        for filename in fnmatch.filter(filenames, mask):
            matches.append(os.path.join(root, filename))

    # deleted file count
    dc = 0
    for fullpath in matches:
            frame_number = os.path.basename(fullpath).split("_")[0]
            json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
            try:
                load_json(json_filename)
            except:
                print('No matching .json file for: ', fullpath)
                # No matching .json file for:  ../../dataset/unity/log/logs_Mon_Jul_13_09_03_21_2020/35095_cam-image
_array_.jpg
                if(delete):
                    print("File deleted.")
                    os.remove(fullpath)
                    dc += 1
                continue

    print("Files deleted:", dc)
def parse_bool(b):
    return b == "True"

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='JSON missing file handler/cleaner')
    parser.add_argument('--inputs', default='../dataset/unity/jungle1/log/*.jpg', help='input mask to gather images')
    parser.add_argument('--delete', type=parse_bool, default=False, help='image deletion flag')
    args = parser.parse_args()

    cleanjson(args.inputs, args.delete)

#######################################################
# 9. makebins.py
#######################################################


"""
Go though Unity3D generated files, get all steering angles and plot a histogram
$ python3 makebins.py --inputs=../../dataset/unity/log/*.jpg
"""

import os
import argparse
```

```
import fnmatch
import json
import seaborn as sns
import os


def load_json(filename):
    with open(filename, "rt") as fp:
        data = json.load(fp)
    return data

def cleanjson(filemask):

    filemask = os.path.expanduser(filemask)
    path, mask = os.path.split(filemask)

    matches = []
    for root, dirnames, filenames in os.walk(path):
        for filename in fnmatch.filter(filenames, mask):
            matches.append(os.path.join(root, filename))

    # steering values
    svals = []
    for fullpath in matches:
            frame_number = os.path.basename(fullpath).split("_")[0]
            json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
            jobj = load_json(json_filename)
            svals.append(jobj['user/angle'])

    print("Files deleted:", dc)
def parse_bool(b):
    return b == "True"

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='JSON missing file handler/cleaner')
    parser.add_argument('--inputs', default='../dataset/unity/jungle1/log/*.jpg', help='input mask to gather images')
    args = parser.parse_args()

    cleanjson(args.inputs)

#######################################################
# 10. MakeVideoFromtcpflow.ipynb.py
#######################################################


#!/usr/bin/env python
# coding: utf-8

# In[123]:


#import argparse
#import fnmatch
import json
```

```
#import os
import base64
import datetime
from io import BytesIO
from PIL import Image
import sys
import numpy as np
import matplotlib.pyplot as plt

# convert image array to PIL image, to see if we can concatenate them
# from PIL import Image
from matplotlib import cm


def GetSteeringFromtcpflow(filename):
    """
    Get a tcpflow log and extract steering values obtained from network communication between
    sim and prediction engine (predict_client.py)
    Inputs
        filename: string, name of tcpflow log
    Returns
        sa: list of arrays, steering angle predicton and actual value tuple.
    """
    # open file
    sa = []
    # initialize prediction
    pred = ''
    f = open(filename, "r")
    file = f.read()
    try:
        #readline = f.read()
        lines = file.splitlines()
        for line in lines:
            # print(line)
            start = line.find('{')
            if(start == -1):
                continue
            jsonstr = line[start:]
            # print(jsonstr)
            jsondict = json.loads(jsonstr)
            if "steering" in jsondict:
                # predicted
                pred = jsondict['steering']
            if "steering_angle" in jsondict:
                # actual
                act = jsondict['steering_angle']
                # save pair, only keep last pred in case two were send as it does happen i.e.:
                # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.071960375", "t
hrottle": "0.08249988406896591", "brake": "0.0"}
                # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.079734944", "t
hrottle": "0.08631626516580582", "brake": "0.0"}
                # 127.000.000.001.09091-127.000.000.001.59460: {"msg_type":"telemetry","steering_angle":-0.0719603
7,(...)
                if(len(pred) > 0):
```

```
                sa.append([float(pred), act])
                pred = '' # need to save this image

                imgString = jsondict["image"]
                image = Image.open(BytesIO(base64.b64decode(imgString)))

                img_arr = np.asarray(image, dtype=np.float32)

                img_arr_proc = preprocess(img_arr)
                img_arr = crop(img_arr)
                img_arr = resize(img_arr)
                # something happens here which "fixes" the image, i.e.
                img_arr = rgb2yuv(img_arr)
                #im = Image.fromarray(np.uint8(cm.gist_earth(img_arr)))
                #stitch = stitchImages(img_arr, img_arr_proc, 160, 120)
                #print(img_arr_proc.shape)
                #print(type(image))
                # plt.imshow(img_arr_proc)
                print(type(img_arr))
                myconc = np.concatenate((img_arr_proc, img_arr), axis = 1)
                # plt.imshow(img_arr)
                plt.imshow(myconc)
                # print(image.size[0])
                sys.exit(0)
    except Exception as e:
        print("Exception raise: " + str(e))
    # file should be automatically closed but will close for good measure
    f.close()
    return sa

sa = GetSteeringFromtcpflow('/tmp/tcpflow.log')


# In[52]:


def stitchImages(a, b, w, h):
    """
    Stitch two images together side by side
    Inputs
        a, b: floating point image arrays
        w, h: integer width and height dimensions
    Output
        c: floating point stitched image array
    """
    # https://stackoverflow.com/questions/30227466/combine-several-images-horizontally-with-python
    total_width = w * 2
    max_height = h
    # convert to PIL image to paste
    a = Image.fromarray(a.astype('uint8'), 'RGB')
    b = Image.fromarray(b.astype('uint8'), 'YCbCr')

    new_im = Image.new('RGB', (total_width, max_height))
    new_im.paste(a, (0,0))
```

```
    new_im.paste(b, (w,0))
    # convert back to float array
    return new_im # np.asarray(new_im, dtype=np.float32) # new_im



# In[124]:


myimg = np.asarray([[[1,1,1], [1,1,1], [1,1,1]],
                    [[1,1,1], [1,1,1], [1,1,1]],
                    [[1,1,1], [1,1,1], [1,1,1]]])


myimg2 = np.asarray([[[2,2,2], [2,2,2], [2,2,2]],
                    [[2,2,2], [2,2,2], [2,2,2]],
                    [[2,2,2], [2,2,2], [2,2,2]]])

myimg3 = np.asarray([[[1,1,1], [1,1,1], [1,1,1]],
                    [[1,1,1], [1,1,1], [1,1,1]],
                    [[1,1,1], [1,1,1], [1,1,1]],
                    [[2,2,2], [2,2,2], [2,2,2]],
                    [[2,2,2], [2,2,2], [2,2,2]],
                    [[2,2,2], [2,2,2], [2,2,2]]])

myimg4 = np.asarray([[[1,1,1], [1,1,1], [1,1,1],[2,2,2], [2,2,2], [2,2,2]],
                    [[1,1,1], [1,1,1], [1,1,1],[2,2,2], [2,2,2], [2,2,2]],
                    [[1,1,1], [1,1,1], [1,1,1],[2,2,2], [2,2,2], [2,2,2]]])

print(myimg.shape)
print(myimg2.shape)
print(myimg3.shape)
print(myimg4.shape)
# np.append(myimg, myimg2, axis=0)

import numpy as np
myconc = np.concatenate((myimg, myimg2), axis = 1)
myconc.shape


# In[127]:


# quick test, can we concatenate if we read from disk?
myimg = load_image('../dataset/unity/jungle1/log/logs_Sat_Nov__7_11_11_06_2020/10000_cam-image_array_.jpg'
)

myprocimg = preprocess(myimg)
myconc = np.concatenate((myimg, myprocimg), axis = 1)
plt.imshow(myconc)


# In[4]:
```

```
import cv2, os
import numpy as np
import matplotlib.image as mpimg

IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 160, 120, 3
# IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 66, 200, 3
# IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 120, 160, 3
INPUT_SHAPE = (IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)
# Dimensions expected by network
IMAGE_WIDTH_NET, IMAGE_HEIGHT_NET = 160, 120


def load_image(image_path):
    """
    Load RGB images from a file
    """
    return mpimg.imread(image_path)


def crop(image):
    """
    Crop the image (removing the sky at the top and the car front at the bottom)
    """
    return image[60:-25, :, :] # remove the sky and the car front


def resize(image):
    """
    Resize the image to the input shape used by the network model
    """
    return cv2.resize(image, (IMAGE_WIDTH_NET, IMAGE_HEIGHT_NET), cv2.INTER_AREA)


def rgb2yuv(image):
    """
    Convert the image from RGB to YUV (This is what the NVIDIA model does)
    """
    return cv2.cvtColor(image, cv2.COLOR_RGB2YUV)


def preprocess(image):
    """
    Combine all preprocess functions into one
    """
    image = crop(image)
    image = resize(image)
    image = rgb2yuv(image)
    return image


def choose_image(data_dir, center, left, right, steering_angle):
    """
    Randomly choose an image from the center, left or right, and adjust
```

```
    the steering angle.
    """
    choice = np.random.choice(3)
    if choice == 0:
        return load_image(data_dir, left), steering_angle + 0.2
    elif choice == 1:
        return load_image(data_dir, right), steering_angle - 0.2
    return load_image(data_dir, center), steering_angle


def random_flip(image, steering_angle):
    """
    Randomly flipt the image left <-> right, and adjust the steering angle.
    """
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 1)
        steering_angle = -steering_angle
    return image, steering_angle


def random_translate(image, steering_angle, range_x, range_y):
    """
    Randomly shift the image virtially and horizontally (translation).
    """
    trans_x = range_x * (np.random.rand() - 0.5)
    trans_y = range_y * (np.random.rand() - 0.5)
    steering_angle += trans_x * 0.002
    trans_m = np.float32([[1, 0, trans_x], [0, 1, trans_y]])
    height, width = image.shape[:2]
    image = cv2.warpAffine(image, trans_m, (width, height))
    return image, steering_angle


def random_shadow(image):
    """
    Generates and adds random shadow
    """
    # (x1, y1) and (x2, y2) forms a line
    # xm, ym gives all the locations of the image
    x1, y1 = IMAGE_WIDTH * np.random.rand(), 0
    x2, y2 = IMAGE_WIDTH * np.random.rand(), IMAGE_HEIGHT
    # could this be a bug?
    xm, ym = np.mgrid[0:IMAGE_HEIGHT, 0:IMAGE_WIDTH]
    # xm, ym = np.mgrid[0:IMAGE_WIDTH, 0:IMAGE_HEIGHT]

    # mathematically speaking, we want to set 1 below the line and zero otherwise
    # Our coordinate is up side down.  So, the above the line:
    # (ym-y1)/(xm-x1) > (y2-y1)/(x2-x1)
    # as x2 == x1 causes zero-division problem, we'll write it in the below form:
    # (ym-y1)*(x2-x1) - (y2-y1)*(xm-x1) > 0
    mask = np.zeros_like(image[:, :, 1])
    mask[(ym - y1) * (x2 - x1) - (y2 - y1) * (xm - x1) > 0] = 1

    # choose which side should have shadow and adjust saturation
```

```
    cond = mask == np.random.randint(2)
    s_ratio = np.random.uniform(low=0.2, high=0.5)

    # adjust Saturation in HLS(Hue, Light, Saturation)
    hls = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
    hls[:, :, 1][cond] = hls[:, :, 1][cond] * s_ratio
    return cv2.cvtColor(hls, cv2.COLOR_HLS2RGB)


def random_brightness(image):
    """
    Randomly adjust brightness of the image.
    """
    # HSV (Hue, Saturation, Value) is also called HSB ('B' for Brightness).
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    ratio = 1.0 + 0.4 * (np.random.rand() - 0.5)
    hsv[:,:,2] =  hsv[:,:,2] * ratio
    return cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)


def augment(image, steering_angle, range_x=100, range_y=10):
    """
    Generate an augumented image and adjust steering angle.
    (The steering angle is associated with the center image)
    """
    # resize - we start with assumed image capture size
    image = cv2.resize(image, (320,160), cv2.INTER_AREA)
    # image = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
    image, steering_angle = random_flip(image, steering_angle)
    image, steering_angle = random_translate(image, steering_angle, range_x, range_y)
    image = random_shadow(image)
    image = random_brightness(image)
    return image, steering_angle


def batch_generator(data_dir, image_paths, steering_angles, batch_size, is_training):
    """
    Generate training image give image paths and associated steering angles
    """
    images = np.empty([batch_size, IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS])
    steers = np.empty(batch_size)
    while True:
        i = 0
        for index in np.random.permutation(image_paths.shape[0]):
            center, left, right = image_paths[index]
            steering_angle = steering_angles[index]
            # argumentation
            if is_training and np.random.rand() < 0.6:
                image, steering_angle = augument(data_dir, center, left, right, steering_angle)
            else:
                image = load_image(data_dir, center)
            # add the image and steering angle to the batch
            images[i] = preprocess(image)
            steers[i] = steering_angle
```

```
            i += 1
            if i == batch_size:
                break
        yield images, steers


# In[5]:


# Steering angle predictions by model 20201107210627_nvidia1.h5
def plotSteering(p,g,n):
    """
    Plot
    Inputs
        p: array of floats, steering angle prediction
        g: array of floats, steering angle ground truth.
        n: float, normalization constant
    Output
    """
    import matplotlib.pyplot as plt

    plt.rcParams["figure.figsize"] = (18,3)

    plt.plot(p*n, label="model")
    plt.plot(g*n, label="simulator")

    plt.ylabel('Steering angle')
    # Set a title of the current axes.
    plt.title('Predicted and actual steering angles: SDSandbox simulator and model 20201107210627_nvidia1.h5')
    # show a legend on the plot
    plt.legend()
    # Display a figure.
    # horizontal grid only
    plt.grid(axis='y')
    # set limit
    plt.xlim([-5,len(sarr)+5])
    # invert axis so if seen sideways plot corresponds to direction of steering wheel
    plt.gca().invert_yaxis()
    plt.show()

p = sarr[:,0]
g = sarr[:,1]
n = 25 # 25 frames per second
plotSteering(p,g,n)


# $ g_s(p,g) = \frac{\sum_i^N \lvert p(i)-g(i) \rvert }{N} \times n_c $

# In[2]:


def gos(p, g, n):
    """
    Calculate the goodness-of-steer between a prediction and a ground truth array.
```

```
    Inputs
        p: array of floats, steering angle prediction
        g: array of floats, steering angle ground truth.
        n: float, normalization constant
    Output
        gos: float, average of absolute difference between ground truth and prediction arrays
    """
    # todo add type assertion
    assert len(p) == len(g), "Arrays must be of equal length"
    return sum(abs(p - g)) / len(p) * n

p = sarr[:,0]
g = sarr[:,1]

sterr = gos(p,g, 25)

print("Goodness of steer: {:.2f}".format(sterr))


# In[2]:


import os
path = '~/git/sdsandbox/dataset/ford/2017-08-04-V2-Log1-Center'
for root, dirnames, filenames in os.walk(path):
    print(os.path.join(root, filename))


# In[ ]:




##################################################
# 11. MakeVideo.py
##################################################


import argparse
import fnmatch
import json
import os
from io import BytesIO
from PIL import Image
import base64
import numpy as np
import matplotlib.pyplot as plt
from augmentation import preprocess
import cv2
import conf
# import debug.RecordVideo as RecordVideo
```

```
def MakeVideo(filename, model, preproc=False):
    """
    Make video from tcpflow logged images.
    video.avi is written to disk
    Inputs
        filename: string, name of tcpflow log
        model: name of model to stamp onto video
        preproc: boolean, show preprocessed image next to original
    Output
        none
    """
    # video name
    video_name = model + '.avi'
    VIDEO_WIDTH, VIDEO_HEIGHT = 800, 600
    IMAGE_WIDTH, IMAGE_HEIGHT = 800, 600
    if(preproc == True): # wide angle
        VIDEO_WIDTH = IMAGE_WIDTH*2
    video = cv2.VideoWriter(video_name, 0, 11, (VIDEO_WIDTH, VIDEO_HEIGHT)) # assumed 11fps
    # font
    font = cv2.FONT_HERSHEY_SIMPLEX

    # normalization constant
    # open file
    sa = []
    # initialize prediction
    pred = ''
    f = open(filename, "r")
    file = f.read()
    try:
        #readline = f.read()
        lines = file.splitlines()
        for line in lines:
            #print(line)
            start = line.find('{')
            if(start == -1):
                continue
            jsonstr = line[start:]
            #print(jsonstr)
            jsondict = json.loads(jsonstr)
            if "steering" in jsondict:
                # predicted
                pred = jsondict['steering']
            if "steering_angle" in jsondict:
                # actual
                act = jsondict['steering_angle']
                # save pair, only keep last pred in case two were send as it does happen i.e.:
                # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.071960375", "t
hrottle": "0.08249988406896591", "brake": "0.0"}
                # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.079734944", "t
hrottle": "0.08631626516580582", "brake": "0.0"}
                # 127.000.000.001.09091-127.000.000.001.59460: {"msg_type":"telemetry","steering_angle":-0.0719603
7,(...)
                if(len(pred) > 0):
                    # save steering angles
```

```
                sa.append([float(pred), act])
                pred = '' # need to save this image
                # process image
                imgString = jsondict["image"]
                # decode string
                image = Image.open(BytesIO(base64.b64decode(imgString)))
                # try to convert to jpg
                #image = np.array(image) # sky colour turns orange (TODO investigate)
                # save
                image.save('frame.jpg')
                # reopen with user-friendlier cv2
                image = cv2.imread('frame.jpg') # 120x160x3
                image_copy = image
                # resize so we can write some info onto image
                image = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
                # add Info to frame
                cv2.putText(image, model, (50, 50), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
                # Predicted steering angle
                pst = sa[len(sa)-1][0]
                pst *= conf.norm_const
                simst = "Predicted steering angle: {:.2f}".format(pst)
                cv2.putText(image, simst, (50, 115), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
                # create a preprocessed copy to compare what simulator generates to what network "sees"
                if (preproc == True):  # wide angle
                    image2 = preprocess(image_copy)
                    image2 = cv2.resize(image2, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
                    cv2.putText(image2, 'Network Image', (50, 50), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
                # concatenate
                if (preproc == True):  # wide angle
                    cimgs = np.concatenate((image, image2), axis=1)
                    image = cimgs
                # model name
                # model
                video.write(image);
                pred = ''
    except Exception as e:
        print("Exception raise: " + str(e))
    # file should be automatically closed but will close for good measure
    f.close()
    cv2.destroyAllWindows()
    video.release()


    return "DummyName.mp4"
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Make Video script')
    parser.add_argument('--filename', type=str, help='tcpflow log')
    parser.add_argument('--model', type=str, help='model name for video label')
    args = parser.parse_args()
    MakeVideo(args.filename, args.model, True)
    # example
    # python MakeVideo.py --filename=/tmp/tcpflow.log --model=20201120184912_sanity.h5
```

```
#####################################################
# 12. mech_turk.py
#####################################################


import os
import fnmatch
from shutil import copyfile

def MechTurkBatch(filemask, op):
    files = 4445 # we want 50 uniformly distributed files
    mod = 88 # 445 / 50
    count = 1
    filemask = os.path.expanduser(filemask)
    path, mask = os.path.split(filemask)

    #make path
    os.makedirs(path + op, exist_ok=True)
    for root, dirnames, filenames in os.walk(path):
        for filename in fnmatch.filter(filenames, mask):
            # matches.append(os.path.join(root, filename))

            if(count % mod == 0):
                src = os.path.join(root, filename)
                dst = root + '/out1/' + filename
                #print("cp " + source + "./mech_turk-v2-1")
                copyfile(src, dst)
            count += 1




if __name__ == "__main__":
    path = '../../dataset/ford/2017-08-04-V2-Log1-Center/*.png'
    op = 'out1'
    MechTurkBatch(path, op)

#####################################################
# 13. models.py
# Note: code based on
# https://github.com/tawnkramer/sdsandbox/blob/master/src/models.py
# Available for audit in audit_files/tawn from sharepoint link
# and
# https://github.com/naokishibuya/car-behavioral-cloning/blob/master/model.py
# Available for audit in audit_files/naoki from sharepoint link

#####################################################


'''
Models
Define the different NN models we will use
Author: Tawn Kramer
'''
```

```
from __future__ import print_function
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Input
from tensorflow.keras.layers import Dense, Lambda, ELU
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense, BatchNormalization
from tensorflow.keras.layers import Cropping2D
from tensorflow.keras.optimizers import Adadelta, Adam, SGD
from tensorflow.keras import initializers, regularizers
# Alexnet


import conf

def show_model_summary(model):
    model.summary()
    for layer in model.layers:
        print(layer.output_shape)

def nvidia_baseline(num_outputs):
    '''
    this model is approximately equal to:
    https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf
    Although nothing is said about dropout or activation, which is assumed to be RELU

    Hi Daniel,

    We used the following settings (we haven't documented them in any publication):

    loss function: MSE
    optimizer: adadelta
    learning rate: 1e-4 (but not really used in adadelta)
    dropout: 0.25

    Best regards,
    Urs
    '''
    # note row and col values are now from _NET
    # Adjust sizes accordingly in conf.py
    row, col, ch = conf.row, conf.col, conf.ch

    drop = 0.1 # spreading droupout
    # batch_init = initializers.glorot_uniform #  Original AlexNet initializers.RandomNormal(mean=0., stddev=0.01);
    img_in = Input(shape=(row, col, ch), name='img_in')
    x = img_in
    # RGB values assumed to be normalized and not centered i.e. x/127.5 - 1.
    x = Lambda(lambda x: x / 255.0)(x)
    x = Conv2D(24, (5, 5), strides=(2, 2), activation='relu', name="conv2d_1")(x)
    x = Dropout(drop)(x)
    x = Conv2D(36, (5, 5), strides=(2, 2), activation='relu', name="conv2d_2")(x) #2nd
    x = Dropout(drop)(x)
    x = Conv2D(48, (5, 5), strides=(2, 2), activation='relu', name="conv2d_3")(x)
    x = Dropout(drop)(x)
    x = Conv2D(64, (3, 3), strides=(1, 1), activation='relu', name="conv2d_4")(x) # default strides=(1,1) # 4th
    x = Dropout(drop)(x)
```

```
    x = Conv2D(64, (3, 3), strides=(1, 1), activation='relu', name="conv2d_5")(x) #5th
    x = Dropout(drop)(x)
    x = Flatten(name='flattened')(x)
    # x = Dense(1164, activation='relu', name="dense_1", kernel_initializer=batch_init, bias_initializer='ones')(x)
    #x = Dropout(drop)(x)
    x = Dense(100, activation='relu', name="dense_2")(x)
    #x = Dropout(drop)(x)
    x = Dense(50, activation='relu', name="dense_3")(x) # Added in Naoki's model
    #x = Dropout(drop)(x)
    # x = Dense(10, activation='relu', name="dense_4", kernel_initializer=batch_init, bias_initializer='zeros')(x)
    #x = Dropout(drop)(x)
    outputs = []
    # outputs.append(Dense(num_outputs, activation='linear', name='steering_throttle')(x))
    outputs.append(Dense(num_outputs, activation='linear', name='steering')(x))

    model = Model(inputs=[img_in], outputs=outputs)
    # opt = Adadelta(learning_rate=0.001, rho=0.95, epsilon=1e-07, name="Adadelta")
    opt = Adam(lr=conf.learning_rate)
    model.compile(optimizer=opt, loss="mse", metrics=['acc'])

    # add weight decay
    # https://stackoverflow.com/questions/41260042/global-weight-decay-in-keras
    #alpha = 0.0005  # weight decay coefficient
    #for layer in model.layers:
    #    if isinstance(layer, Conv2D) or isinstance(layer, Dense):
    #        layer.add_loss(lambda: regularizers.l2(alpha)(layer.kernel))
    #    if hasattr(layer, 'bias_regularizer') and layer.use_bias:
    #        layer.add_loss(lambda: regularizers.l2(alpha)(layer.bias))
    return model

def nvidia_model1(num_outputs):
    '''
    This model expects image input size 160hx120w
    this model is inspired by the NVIDIA paper
    https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf
    Activation is RELU
    '''
    # row, col, ch = conf.row, conf.col, conf.ch
    # better albeit less readable
    row, col, ch = conf.nvidia1_img_dims[conf.IMG_HEIGHT_NET_IDX], conf.nvidia1_img_dims[conf.IMG_WID
TH_NET_IDX], conf.nvidia1_img_dims[conf.IMG_DEPTH_IDX]
    drop = 0.1

    img_in = Input(shape=(row, col, ch), name='img_in')
    x = img_in
    # x = Cropping2D(cropping=((10,0), (0,0)))(x) #trim 10 pixels off top
    # x = Lambda(lambda x: x/127.5 - 1.)(x) # normalize and re-center
    x = Lambda(lambda x: x/255.0)(x)
    x = Conv2D(24, (5,5), strides=(2,2), activation='relu', name="conv2d_1")(x)
    x = Dropout(drop)(x)
    x = Conv2D(32, (5,5), strides=(2,2), activation='relu', name="conv2d_2")(x)
    x = Dropout(drop)(x)
    x = Conv2D(64, (5,5), strides=(2,2), activation='relu', name="conv2d_3")(x)
    x = Dropout(drop)(x)
```

```
    x = Conv2D(64, (3,3), strides=(1,1), activation='relu', name="conv2d_4")(x)
    x = Dropout(drop)(x)
    x = Conv2D(64, (3,3), strides=(1,1), activation='relu', name="conv2d_5")(x)
    x = Dropout(drop)(x)

    x = Flatten(name='flattened')(x)
    x = Dense(1064, activation='relu')(x)
    x = Dense(100, activation='relu')(x)
    #x = Dropout(drop)(x)
    x = Dense(50, activation='relu')(x)
    #x = Dropout(drop)(x)
    x = Dense(10, activation='relu')(x)

    outputs = []
    outputs.append(Dense(num_outputs, activation='linear', name='steering_throttle')(x))


    model = Model(inputs=[img_in], outputs=outputs)
    opt = Adam(lr=0.0001)
    model.compile(optimizer=opt, loss="mse", metrics=['acc'])
    # might want to try metrics=['acc', 'loss']  https://stackoverflow.com/questions/51047676/how-to-get-accuracy-of-
model-using-keras
    return model


def nvidia_model2(num_outputs):
    '''
    A.K.A. NaokiNet - https://github.com/naokishibuya/car-behavioral-cloning/blob/master/model.py
    This model expects images of size 66,200,3
    '''
    # row, col, ch = conf.row, conf.col, conf.ch
    row, col, ch = conf.nvidia2_img_dims[conf.IMG_HEIGHT_NET_IDX], conf.nvidia2_img_dims[conf.IMG_WID
TH_NET_IDX], \
                conf.nvidia2_img_dims[conf.IMG_DEPTH_IDX]

    drop = 0.1

    img_in = Input(shape=(row, col, ch), name='img_in')
    x = img_in
    # x = Cropping2D(cropping=((10,0), (0,0)))(x) #trim 10 pixels off top
    # x = Lambda(lambda x: x/127.5 - 1.0)(x) # normalize and re-center
    x = Lambda(lambda x: x / 255.0)(x)
    x = Conv2D(24, (5, 5), strides=(2, 2), activation='elu', name="conv2d_1")(x)
    x = Dropout(drop)(x)
    x = Conv2D(32, (5, 5), strides=(2, 2), activation='elu', name="conv2d_2")(x)
    x = Dropout(drop)(x)
    x = Conv2D(48, (5, 5), strides=(2, 2), activation='elu', name="conv2d_3")(x)
    x = Dropout(drop)(x)
    x = Conv2D(64, (3, 3), activation='elu', name="conv2d_4")(x) # default strides=(1,1)
    x = Dropout(drop)(x)
    x = Conv2D(64, (3, 3), activation='elu', name="conv2d_5")(x)
    x = Dropout(drop)(x)

    x = Flatten(name='flattened')(x)
```

```
    x = Dense(100, activation='elu')(x)
    # x = Dropout(drop)(x)
    x = Dense(50, activation='elu')(x)
    # x = Dropout(drop)(x)
    x = Dense(10, activation='elu')(x) # Added in Naoki's model

    outputs = []
    # outputs.append(Dense(num_outputs, activation='linear', name='steering_throttle')(x))
    outputs.append(Dense(num_outputs, name='steering_throttle')(x))

    model = Model(inputs=[img_in], outputs=outputs)
    opt = Adam(lr=0.0001)
    model.compile(optimizer=opt, loss="mse", metrics=['acc'])
    return model

def nvidia_model3(num_outputs):
    '''
    This model expects images of size 66,200,3
    '''
    row, col, ch = conf.row, conf.col, conf.ch

    drop = 0.1

    img_in = Input(shape=(row, col, ch), name='img_in')
    x = img_in
    # x = Cropping2D(cropping=((10,0), (0,0)))(x) #trim 10 pixels off top
    x = Lambda(lambda x: x/127.5 - 1.0)(x) # normalize and re-center
    # x = Lambda(lambda x: x / 255.0)(x)
    x = Conv2D(24, (5, 5), strides=(2, 2), activation='elu', name="conv2d_1")(x)
    # x = Dropout(drop)(x)
    x = Conv2D(36, (5, 5), strides=(2, 2), activation='elu', name="conv2d_2")(x)
    #x = Dropout(drop)(x)
    x = Conv2D(48, (5, 5), strides=(2, 2), activation='elu', name="conv2d_3")(x)
    #x = Dropout(drop)(x)
    x = Conv2D(64, (3, 3), activation='elu', name="conv2d_4")(x) # default strides=(1,1)
    #x = Dropout(drop)(x)
    x = Conv2D(64, (3, 3), activation='elu', name="conv2d_5")(x)
    x = Dropout(drop)(x)

    x = Flatten(name='flattened')(x)

    x = Dense(100, activation='elu')(x)
    # x = Dropout(drop)(x)
    x = Dense(50, activation='elu')(x)
    # x = Dropout(drop)(x)
    x = Dense(10, activation='elu')(x) # Added in Naoki's model

    outputs = []
    # outputs.append(Dense(num_outputs, activation='linear', name='steering_throttle')(x))
    outputs.append(Dense(num_outputs, name='steering_throttle')(x))

    model = Model(inputs=[img_in], outputs=outputs)
    opt = Adam(lr=0.0001)
```

```
    model.compile(optimizer=opt, loss="mse", metrics=['acc'])
    return model

def get_alexnet(num_outputs):
    '''
    this model is also inspired by the NVIDIA paper
    https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf
    but taken from
    https://github.com/naokishibuya/car-behavioral-cloning/blob/master/model.py
    model = Sequential()
    model.add(Lambda(lambda x: x/127.5-1.0, input_shape=INPUT_SHAPE))
    model.add(Conv2D(24, 5, 5, activation='elu', strides=2))
    model.add(Conv2D(36, 5, 5, activation='elu', strides=2))
    model.add(Conv2D(48, 5, 5, activation='elu', strides=2))
    model.add(Conv2D(64, 3, 3, activation='elu'))
    model.add(Conv2D(64, 3, 3, activation='elu'))
    model.add(Dropout(args.keep_prob))
    model.add(Flatten())
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))
    model.summary()
    NB Tawn Kramer's model uses dropout = 0.1 on five layers, Naoki uses
    0.5 on a single layer
    '''
    #row, col, ch = conf.image_width_alexnet, conf.image_height_alexnet, conf.ch
    row, col, ch = conf.alexnet_img_dims[conf.IMG_HEIGHT_NET_IDX], conf.nvidia2_img_dims[conf.IMG_WID
TH_NET_IDX], \
    conf.nvidia2_img_dims[conf.IMG_DEPTH_IDX]
    drop = 0.5
    # read https://stackoverflow.com/questions/58636087/tensorflow-valueerror-failed-to-convert-a-numpy-array-to-a
-tensor-unsupporte
    # to work out shapes
    img_in = Input(shape=(row, col, ch), name='img_in')
    x = img_in
    # x = Cropping2D(cropping=((10,0), (0,0)))(x) #trim 10 pixels off top
    x = Lambda(lambda x: x/127.5 - 1.0)(x) # normalize and re-center
    # x = Lambda(lambda x: x / 255.0)(x)
    x = Conv2D(48, (8, 8), strides=(4, 4), padding='valid', activation='relu', name="conv2d_1")(x)
    # x = Dropout(drop)(x)
    x = MaxPooling2D(48, (1, 1), padding="same", name="maxpool2d_1")(x)

    x = Conv2D(128, (3, 3), strides=(2, 2), padding='valid', activation='relu', name="conv2d_2")(x)
    #x = Dropout(drop)(x)
    x = MaxPooling2D(128, (1, 1), padding="same", name="maxpool2d_2")(x)

    x = Conv2D(192, (3, 3), strides=(2, 2), padding='valid', activation='relu', name="conv2d_3")(x)
    #x = Dropout(drop)(x)
    x = Conv2D(192, (3, 3), strides=(1, 1), padding='same', activation='relu', name="conv2d_4")(x) # default strides=
(1,1)
    #x = Dropout(drop)(x)

    x = Conv2D(128, (3, 3), strides=(1, 1), padding='same', activation='relu', name="conv2d_5")(x)
```

```
    x = MaxPooling2D(128, (1, 1), padding="same", name="maxpool2d_3")(x)

    #x = Conv2D(64, (3, 3), activation='relu', name="conv2d_6")(x)
    # error Negative dimension size caused by subtracting 128 from 10 for '{{node max_pooling2d/MaxPool}} = Ma
xPool[T=DT_FLOAT,
    # data_format="NHWC", ksize=[1, 128, 128, 1], padding="VALID", strides=[1, 3, 3, 1]](conv2d_4/Identity)'
    # with input shapes: [?,10,10,192].
    # x = MaxPooling2D(128, (3, 3), padding="SAME")(x)
    # By commenting out line above, error is:
    #  Input to reshape is a tensor with 1843200 values, but the requested shape requires a multiple of 101568
    #   [[node model/flattened/Reshape (defined at /git/sdsandbox/src/train.py:272) ]] [Op:__inference_train_function
_1081]

    x = Dropout(drop)(x)

    """
        x = Conv2D(24, (5, 5), strides=(2, 2), activation='elu', name="conv2d_1")(x)
    # x = Dropout(drop)(x)
    x = Conv2D(36, (5, 5), strides=(2, 2), activation='elu', name="conv2d_2")(x)
    #x = Dropout(drop)(x)
    x = Conv2D(64, (5, 5), strides=(2, 2), activation='elu', name="conv2d_3")(x)
    #x = Dropout(drop)(x)
    x = Conv2D(64, (3, 3), activation='elu', name="conv2d_4")(x) # default strides=(1,1)
    #x = Dropout(drop)(x)
    x = MaxPooling2D(64, (3, 3), name="maxpool2d_5")(x)
    x = Dropout(drop)(x)

    x = Flatten(name='flattened')(x)

    x = Dense(2048, activation='elu')(x)
    # x = Dropout(drop)(x)
    x = Dense(2048, activation='elu')(x)
    # x = Dropout(drop)(x)
    # x = Dense(10, activation='elu')(x) # Added in Naoki's model
    """


    x = Flatten(name='flattened')(x) # error when followed by

    x = Dense(2048, name='Dense_1', activation='relu')(x)# 2048, 2048 ~  Input to reshape is a tensor with 442368 va
lues, but the requested shape requires a multiple of 21632
   # [[node model/flattened/Reshape (defined at /git/sdsandbox/src/train.py:272) ]] [Op:__inference_train_function_1
192]

    # x = Dropout(drop)(x)
    x = Dense(50, name='Dense_2', activation='elu')(x)
    # x = Dropout(drop)(x)
    x = Dense(10, activation='elu')(x) # Added in Naoki's model

    outputs = []
    # outputs.append(Dense(num_outputs, activation='linear', name='steering_throttle')(x))
    outputs.append(Dense(num_outputs, name='steering_throttle')(x))

    model = Model(inputs=[img_in], outputs=outputs)
```

```
    opt = Adam(lr=0.0001)
    model.compile(optimizer=opt, loss="mse", metrics=['acc'])
    return model
"""
def alexnet_model(img_shape=(224, 224, 3), n_classes=10, l2_reg=0.,
 weights=None):

 # Initialize model
 alexnet = Sequential()

 # Layer 1
 alexnet.add(Conv2D(96, (11, 11), input_shape=img_shape,
  padding='same', kernel_regularizer=l2(l2_reg)))
 alexnet.add(BatchNormalization())
 alexnet.add(Activation('relu'))
 alexnet.add(MaxPooling2D(pool_size=(2, 2)))

 # Layer 2
 alexnet.add(Conv2D(256, (5, 5), padding='same'))
 alexnet.add(BatchNormalization())
 alexnet.add(Activation('relu'))
 alexnet.add(MaxPooling2D(pool_size=(2, 2)))

 # Layer 3
 alexnet.add(ZeroPadding2D((1, 1)))
 alexnet.add(Conv2D(512, (3, 3), padding='same'))
 alexnet.add(BatchNormalization())
 alexnet.add(Activation('relu'))
 alexnet.add(MaxPooling2D(pool_size=(2, 2)))

 # Layer 4
 alexnet.add(ZeroPadding2D((1, 1)))
 alexnet.add(Conv2D(1024, (3, 3), padding='same'))
 alexnet.add(BatchNormalization())
 alexnet.add(Activation('relu'))

 # Layer 5
 alexnet.add(ZeroPadding2D((1, 1)))
 alexnet.add(Conv2D(1024, (3, 3), padding='same'))
 alexnet.add(BatchNormalization())
 alexnet.add(Activation('relu'))
 alexnet.add(MaxPooling2D(pool_size=(2, 2)))

 # Layer 6
 alexnet.add(Flatten())
 alexnet.add(Dense(3072))
 alexnet.add(BatchNormalization())
 alexnet.add(Activation('relu'))
 alexnet.add(Dropout(0.5))

 # Layer 7
 alexnet.add(Dense(4096))
 alexnet.add(BatchNormalization())
 alexnet.add(Activation('relu'))
```

```
 alexnet.add(Dropout(0.5))

 # Layer 8
 alexnet.add(Dense(n_classes))
 alexnet.add(BatchNormalization())
 alexnet.add(Activation('softmax'))

 if weights is not None:
  alexnet.load_weights(weights)

 return alexnet
"""

#####################################################
# 14. predict_client.py
# Note: code based on
# https://github.com/tawnkramer/sdsandbox/blob/master/src/predict_client.py
# Available for audit in audit_files/tawn from sharepoint link
#####################################################


'''
Predict Server
Create a server to accept image inputs and run them against a trained neural network.
This then sends the steering output back to the client.
Author: Tawn Kramer
'''
from __future__ import print_function
import os
import sys
import argparse
import time
import json
import base64
import datetime
from io import BytesIO
import signal
import tensorflow as tf
from tensorflow.python import keras
from tensorflow.python.keras.models import load_model
from PIL import Image
import numpy as np
from gym_donkeycar.core.fps import FPSTimer
from gym_donkeycar.core.message import IMesgHandler
from gym_donkeycar.core.sim_client import SimClient
# same preprocess as for training
from augmentation import augment, preprocess
import conf
from helper_functions import parse_bool
import utils.RecordVideo as RecordVideo
import Augmentation


if tf.__version__ == '1.13.1':
```

```
    from tensorflow import ConfigProto, Session

    # Override keras session to work around a bug in TF 1.13.1
    # Remove after we upgrade to TF 1.14 / TF 2.x.
    config = ConfigProto()
    config.gpu_options.allow_growth = True
    session = Session(config=config)
    keras.backend.set_session(session)

# need to import file TODO
import Automold as am
import Helpers as hp
import numpy as np
# helper function for prediction

def add_rain(image_arr, rt=None, st=0):
    """
    Add rain to image
    Inputs:
        image_arr: numpy array containing image
        rt: string, rain type "heavy" or "torrential"
        st: range to draw a random slant from
    Output
        image_arr: numpy array containing image with rain
    """
    # print("Adding rain...")
    if(st != 0):
        # draw a random number for slant
        st = np.random.randint(-1 * st, st)

    if(rt!='light'): # heavy or torrential
        image_arr = am.add_rain_single(image_arr, rain_type=rt, slant=st)
    else:
         # no slant
        image_arr = am.add_rain_single(image_arr)

    return image_arr

class DonkeySimMsgHandler(IMesgHandler):

    STEERING = 0
    THROTTLE = 1

    def __init__(self, model, constant_throttle, image_cb=None, rand_seed=0):
        self.model = model
        self.constant_throttle = constant_throttle
        self.client = None
        self.timer = FPSTimer()
        self.img_arr = None
        self.image_cb = image_cb
        self.steering_angle = 0.
        self.throttle = 0.
        self.rand_seed = rand_seed
        self.fns = {'telemetry' : self.on_telemetry,\
```

```
            'car_loaded' : self.on_car_created,\
            'on_disconnect' : self.on_disconnect,
            'aborted' : self.on_aborted}
    # images to record
    self.img_orig = None
    self.img_add_rain = None
    self.img_processed = None
    self.frame_count = 0

    # model name


def on_connect(self, client):
    self.client = client
    self.timer.reset()

def on_aborted(self, msg):
    self.stop()

def on_disconnect(self):
    pass

def on_recv_message(self, message):
    self.timer.on_frame()
    if not 'msg_type' in message:
        print('expected msg_type field')
        print("message:", message)
        return

    msg_type = message['msg_type']
    if msg_type in self.fns:
        self.fns[msg_type](message)
    else:
        print('unknown message type', msg_type)

def on_car_created(self, data):
    if self.rand_seed != 0:
        self.send_regen_road(0, self.rand_seed, 1.0)

def on_telemetry(self, data):
    imgString = data["image"]
    image = Image.open(BytesIO(base64.b64decode(imgString)))
    img_arr = np.asarray(image, dtype=np.float32)
    self.frame_count += 1
    self.img_orig = img_arr

    # set to same image size expected from acquisition process
    img_arr = ag.resize_expected(img_arr)

    # check for rain
    if(conf.rt != ''):
        img_arr = add_rain(img_arr, conf.rt, conf.st)
        self.img_add_rain = img_arr
```

```
    # same preprocessing as for training
    img_arr = ag.preprocess(img_arr)
    self.img_processed = img_arr

    #if(conf.record == True):
    #    text = (['Network Image', 'No Rain'])
    #    rv.add_image(img_arr, text)

    # if we are testing the network with rain
    self.img_arr = img_arr.reshape((1,) + img_arr.shape)

    if self.image_cb is not None:
        self.image_cb(img_arr, self.steering_angle )

def update(self):
    if self.img_arr is not None:
        self.predict(self.img_arr)
        self.img_arr = None

def predict(self, image_array):
    outputs = self.model.predict(image_array)
    # check if we are recording
    if (conf.record == True):

        # Add first image, with name of network and frame number
        # TODO, get network name from argument
        text = ([rv.modelname, 'Intensity Multiplie: 1', 'Acquired image', 'Frame:  ' + str(self.frame_count)])
        rv.add_image(self.img_orig, text)

        # Add second image, preprocessed with rain or without
        # text = (['Network image', 'No rain'])
        # rv.add_image(self.img_processed, text)
        # if rain added
        # check for rain
        if (conf.rt != ''):
            rtype = 'Type: ' + conf.rt
            s = 'Slant: -+' + str(conf.st)
            text = (['Added rain', rtype, s])
            rv.add_image(self.img_add_rain, text)

            # add third image with prediction
        steering = outputs[0][0]
        steering *= conf.norm_const
        st_str = "{:.2f}".format(steering)
        st_str = "Predicted steering angle: " + st_str
        # st_str = "Predicted steering angle: 20"
        #rtype = 'Type: ' + conf.rt
        #s = 'Slant: -+' + str(conf.st)
        text = (["Network image", st_str])
        rv.add_image(image_array[0], text)
        rv.add_frame()
    self.parse_outputs(outputs)
```

```
    def parse_outputs(self, outputs):
        res = []

        # Expects the model with final Dense(2) with steering and throttle
        for i in range(outputs.shape[1]):
            res.append(outputs[0][i])

        self.on_parsed_outputs(res)

    def on_parsed_outputs(self, outputs):
        self.outputs = outputs
        self.steering_angle = 0.0
        self.throttle = 0.2

        if len(outputs) > 0:
            self.steering_angle = outputs[self.STEERING]

        if self.constant_throttle != 0.0:
            self.throttle = self.constant_throttle
        elif len(outputs) > 1:
            self.throttle = outputs[self.THROTTLE] * conf.throttle_out_scale

        self.send_control(self.steering_angle, self.throttle)

    def send_control(self, steer, throttle):
        # print("send st:", steer, "th:", throttle)
        msg = { 'msg_type' : 'control', 'steering': steer.__str__(), 'throttle':throttle.__str__(), 'brake': '0.0' }
        self.client.queue_message(msg)

    def send_regen_road(self, road_style=0, rand_seed=0, turn_increment=0.0):
        '''
        Regenerate the road, where available. For now only in level 0.
        In level 0 there are currently 5 road styles. This changes the texture on the road
        and also the road width.
        The rand_seed can be used to get some determinism in road generation.
        The turn_increment defaults to 1.0 internally. Provide a non zero positive float
        to affect the curviness of the road. Smaller numbers will provide more shallow curves.
        '''
        msg = { 'msg_type' : 'regen_road',
            'road_style': road_style.__str__(),
            'rand_seed': rand_seed.__str__(),
            'turn_increment': turn_increment.__str__() }

        self.client.queue_message(msg)

    def stop(self):
        self.client.stop()

    def __del__(self):
        self.stop()



def clients_connected(arr):
```

```
    for client in arr:
        if not client.is_connected():
            return False
    return True


def go(filename, address, constant_throttle=0, num_cars=1, image_cb=None, rand_seed=None):

    print("loading model", filename)
    model = load_model(filename)

    # In this mode, looks like we have to compile it
    model.compile("sgd", "mse")

    clients = []

    for _ in range(0, num_cars):
        # setup the clients
        handler = DonkeySimMsgHandler(model, constant_throttle, image_cb=image_cb, rand_seed=rand_seed)
        client = SimClient(address, handler)
        clients.append(client)

    while clients_connected(clients):
        try:
            time.sleep(0.02)
            for client in clients:
                client.msg_handler.update()
        except KeyboardInterrupt:
            # unless some hits Ctrl+C and then we get this interrupt
            print('stopping')
            break

def stop_exec(signum, frame):
    # restore the original signal handler as otherwise evil things will happen
    # in raw_input when CTRL+C is pressed, and our signal handler is not re-entrant
    signal.signal(signal.SIGINT, original_sigint)

    try:
        # changed raw_input to input
        if input("\nFinish recording video? (y/n)> ").lower().startswith('y'):
            print("*** CTRL+C to stop ***")
            rv.save_video()
            sys.exit(1)

    except KeyboardInterrupt:
        print("Ok ok, quitting")
        sys.exit(1)

    # restore the exit gracefully handler here
    signal.signal(signal.SIGINT, stop_exec)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='prediction server')
    parser.add_argument('--model', type=str, help='model filename')
```

```
    parser.add_argument('--modelname', type=str, default='nvidia1', help='model filename')
    parser.add_argument('--host', type=str, default='127.0.0.1', help='server sim host')
    parser.add_argument('--port', type=int, default=9091, help='bind to port')
    parser.add_argument('--num_cars', type=int, default=1, help='how many cars to spawn')
    parser.add_argument('--constant_throttle', type=float, default=0.0, help='apply constant throttle')
    parser.add_argument('--rand_seed', type=int, default=0, help='set road generation random seed')
    parser.add_argument('--rain', type=str, default='', help='type of rain [light|heavy|torrential')
    parser.add_argument('--slant', type=int, default=0, help='Rain slant deviation')
    parser.add_argument('--record', type=parse_bool, default="False", help='Record video of raw and processed imag
es')
    # parser.add_argument('--img_cnt', type=int, default=3, help='Number of side by side images to record')


    args = parser.parse_args()
    address = (args.host, args.port)

    conf.rt = args.rain
    conf.st = args.slant
    conf.record = args.record

    ag = Augmentation.Augmentation(args.modelname)

    if conf.record == True:
        print("*** When finished, press CTRL+C and y to finish recording, then CTRL+C to quit ***")
        original_sigint = signal.getsignal(signal.SIGINT)
        signal.signal(signal.SIGINT, stop_exec)
        rv = RecordVideo.RecordVideo(args.model, conf.rt)

    go(args.model, address, args.constant_throttle, num_cars=args.num_cars, rand_seed=args.rand_seed)
    # max value for slant is 20
    # Example
    # python3 predict_client.py --model=../trained_models/sanity/20201120171015_sanity.h5 --rain=light --slant=0



#####################################################
# 15. RecordVideo.py
#####################################################


# https://docs.python.org/3/tutorial/classes.html
import cv2
import conf
import numpy as np


class RecordVideo():
    """
    Record video class, used to record videos from tcpflow log or still images
    """

    def __init__(self, model, rt):
        """
        Record video while running predictions
```

```
        Inputs
            model: string, model name
            rt: type of rain
        """
        model = model.split('/')
        self.modelname = model[-1]
        videoname = self.modelname + '.avi'
        # 3 images side by side in the rain
        img_cnt = 3
        if(rt == ''): # 2 images in the dry
            img_cnt = 2
        self.img_cnt = img_cnt
        self.VIDEO_WIDTH, self.VIDEO_HEIGHT = conf.VIDEO_WIDTH, conf.VIDEO_HEIGHT  # 800, 600
        self.IMAGE_WIDTH, self.IMAGE_HEIGHT = conf.IMAGE_STILL_WIDTH, conf.IMAGE_STILL_HEIGH
T
        self.VIDEO_WIDTH = self.IMAGE_WIDTH * img_cnt
        self.video = cv2.VideoWriter(videoname, 0, 11,
                                     (self.VIDEO_WIDTH, self.VIDEO_HEIGHT))  # assumed 11fps approximately
        self.font = cv2.FONT_HERSHEY_SIMPLEX
        # video line spacing
        self.images = []

    def add_image(self, image, text):
        # self.img_arr_1 =
        # cv2.putText(image, model, (50, 50), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
        # Predicted steering angle
        # pst = sa[len(sa) - 1][0]
        # pst *= conf.norm_const
        # simst = "Predicted steering angle: {:.2f}".format(pst)
        # cv2.putText(image, simst, (50, 115), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
        self.images.append([image, text])

    def add_frame(self):
        y_offset = 50;
        x_offset = 50
        step = 65;
        img_cnt = len(self.images)
        # prepare images
        for i in range(0, img_cnt):
            # resize image
            image = self.images[i][0]
            image = cv2.resize(image, (self.IMAGE_WIDTH, self.IMAGE_HEIGHT), cv2.INTER_AREA)
            # add text
            lines = self.images[i][1]
            for line in lines:
                cv2.putText(image, line, (x_offset, y_offset), self.font, 1, (255, 255, 255), 2, cv2.LINE_AA)
                y_offset += step
                # print(line)
            # set start to top for printing lines in next frame
            y_offset = x_offset
            # store processed image
            self.images[i][0] = image
        # concatenate
        output_image = self.images[0][0]
```

```
        for i in range(1, img_cnt):
            output_image = np.concatenate((output_image, self.images[i][0]), axis=1)
        # append
        try:
            self.video.write(np.uint8(output_image))  # catch error Assertion failed) image.depth() == CV_8U
        except Exception as e:
            print("Exception raise: " + str(e))
        # blank images
        self.images = []

    def save_video(self):
        # save video as videoname
        cv2.destroyAllWindows()
        self.video.release()


##################################################
# 16. result_plots.py
##################################################


import numpy as np
from utils.steerlib import GetSteeringFromtcpflow, plotMultipleSteeringAngles

def plot1():
    """
    Intensity multiplier 1, 20201207091932_nvidia1.h5
    """

    # intensity multiplier = 1
    p = []
    # No rain mult 1 (shaw we have a "no rain" for every multiplier ?)
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_no_rain_tcpflow.lo
g')
    sarr = np.asarray(sa)
    pa = sarr[:,0]
    p.append([pa, 'no rain'])
    # light mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_light_rain_mult_1_
tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:,0]
    p.append([pa, 'light rain'])
    # heavy mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_heavy_10_mult_1_
tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:,0]
    p.append([pa, 'heavy rain slant +-10'])
    # torrential mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_torrential_20_mult
_1_tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:,0]
```

```
    p.append([pa, 'torrential rain slant +-20'])

    plotMultipleSteeringAngles(p, 25, True, "Generated Track intensity multiplier 1", "20201207091932_nvidia1.h5"
, 'tcpflow log predicted')

def plot2():
    """
    Intensity multiplier 4, 20201207091932_nvidia1.h5
    """
    # intensity multiplier = 4
    p = []
    # No rain mult 1 (shaw we have a "no rain" for every multiplier ?)
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_no_rain_tcpflow.lo
g')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'no rain'])
    # light mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_light_rain_mult_4_
tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'light rain'])
    # heavy mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_heavy_10_mult_4_
tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'heavy rain slant +-10'])
    # torrential mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_torrential_20_mult
_4_tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'torrential rain slant +-20'])

    plotMultipleSteeringAngles(p, 25, True, "Generated Track intensity multiplier 4", "20201207091932_nvidia1.h5"
,
                               'tcpflow log predicted')

def plot3():
    """
    Intensity multiplier 8, 20201207091932_nvidia1.h5
    """

    # intensity multiplier = 8
    p = []
    # No rain mult 1 (shaw we have a "no rain" for every multiplier ?)
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_no_rain_tcpflow.lo
g')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'no rain'])
    # light mult 1
```

```
    sa = GetSteeringFromtcpflow(
        '../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_light_rain_mult_8_tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'light rain'])
    # heavy mult 1
    sa = GetSteeringFromtcpflow(
        '../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_heavy_10_mult_8_tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'heavy rain slant +-10'])
    # torrential mult 1
    sa = GetSteeringFromtcpflow(
        '../../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_torrential_20_mult_8_tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'torrential rain slant +-20'])

    plotMultipleSteeringAngles(p, 25, True, "Generated Track intensity multiplier 8", "20201207091932_nvidia1.h5"
,
                               'tcpflow log predicted')

def plot4():
    """
    Intensity multiplier 1, 20201207192948_nvidia2.h5
    """

    # intensity multiplier = 1
    p = []
    # No rain mult 1 (shaw we have a "no rain" for every multiplier ?)
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_no_rain_tcpflow.lo
g')
    sarr = np.asarray(sa)
    pa = sarr[:,0]
    p.append([pa, 'no rain'])
    # light mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_light_rain_mult_1_
tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:,0]
    p.append([pa, 'light rain'])
    # heavy mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_heavy_10_mult_1_
tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:,0]
    p.append([pa, 'heavy rain slant +-10'])
    # torrential mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_torrential_20_mult
_1_tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:,0]
    p.append([pa, 'torrential rain slant +-20'])
```

```
    plotMultipleSteeringAngles(p, 25, True, "Generated Track intensity multiplier 1", "20201207192948_nvidia2.h5"
, 'tcpflow log predicted')

def plot5():
    """
    Intensity multiplier 4, 20201207192948_nvidia2.h5
    """
    # intensity multiplier = 4
    p = []
    # No rain mult 1 (shaw we have a "no rain" for every multiplier ?)
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_no_rain_tcpflow.lo
g')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'no rain'])
    # light mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_light_rain_mult_4_
tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'light rain'])
    # heavy mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_heavy_10_mult_4_
tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'heavy rain slant +-10'])
    # torrential mult 1
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_torrential_20_mult
_4_tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'torrential rain slant +-20'])

    plotMultipleSteeringAngles(p, 25, True, "Generated Track intensity multiplier 4", "20201207192948_nvidia2.h5"
,
                               'tcpflow log predicted')

def plot6():
    """
    Intensity multiplier 8, 20201207192948_nvidia2.h5
    """

    # intensity multiplier = 8
    p = []
    # No rain mult 1 (shaw we have a "no rain" for every multiplier ?)
    sa = GetSteeringFromtcpflow('../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_no_rain_tcpflow.lo
g')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'no rain'])
    # light mult 1
    sa = GetSteeringFromtcpflow(
        '../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_light_rain_mult_8_tcpflow.log')
```

```
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'light rain'])
    # heavy mult 1
    sa = GetSteeringFromtcpflow(
        '../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_heavy_10_mult_8_tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'heavy rain slant +-10'])
    # torrential mult 1
    sa = GetSteeringFromtcpflow(
        '../../trained_models/nvidia2/tcpflow/20201207192948_nvidia2_torrential_20_mult_8_tcpflow.log')
    sarr = np.asarray(sa)
    pa = sarr[:, 0]
    p.append([pa, 'torrential rain slant +-20'])

    plotMultipleSteeringAngles(p, 25, True, "Generated Track intensity multiplier 8", "20201207192948_nvidia2.h5"
,
                               'tcpflow log predicted')
if __name__ == "__main__":
    # 20201207091932_nvidia1.h5
    # mult_1
    # plot1()
    # mult_4
    # plot2()
    # mult_8
    # plot3()
    # 20201207192948_nvidia2.h5
    # mult_1
    # plot4()
    # mult_4
    # plot5()
    # mult_8
    # plot6()

#####################################################
# 17. SteeringAngleBins.ipynb.py
#####################################################



#!/usr/bin/env python
# coding: utf-8

# In[1]:


import fnmatch
import json
import seaborn as sns
import os
import numpy as np
import matplotlib.pyplot as plt
import statistics
```

```
def load_json(filepath):
    """
    Load a json file
    Inputs
        filepath: string, path to file
    Outputs:
        data: dictionary, json key, value pairs
    """
    with open(filepath, "rt") as fp:
        data = json.load(fp)
    return data

def GetJSONSteeringAngles(filemask):
    """
    Get steering angles stored as 'user/angle' attributes in .json files
    Inputs:
        filemask: string, path and mask
    Outputs
        svals: list, steering values
    """
    filemask = os.path.expanduser(filemask)
    path, mask = os.path.split(filemask)

    matches = []
    for root, dirnames, filenames in os.walk(path):
        for filename in fnmatch.filter(filenames, mask):
            matches.append(os.path.join(root, filename))

    # matches = sorted(matches, key=os.path.getmtime)

    # steering values
    svals = []
    for fullpath in matches:
            frame_number = os.path.basename(fullpath).split("_")[0]
            json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
            jobj = load_json(json_filename)
            svals.append(jobj['user/angle'])
    return svals


# In[5]:


#als = GetJSONSteeringAngles('~/git/msc-data/unity/genRoad/*.jpg')
#es = len(svals)
# Unity steering angle
#nityMaxSteering = 25#
#valscp = [element * UnityMaxSteering for element in svals]
# NB Plotted as normalized histogram
#ns.distplot(svalscp, bins=50, kde=False, norm_hist=True,
#           axlabel='Steering Angles (degrees) norm. hist. ' + str(values) + " data points \n \
#           mean = 0.00 std = 0.00")

# sns_plot.savefig("output.png")
```

```
def jsonSteeringBins(filemask, pname="output", save=True, nc=25):
    """
    Plot a steering values' histogram
    Inputs
        filemask: string, where to search for images, and corresponding .json files
        pname: string, output plot name
        save: boolean, save plot to disk
        nc: int, normalization constant, used in the simulator to put angles in range
        -1, 1. Default is 25.
    Outputs
        svals: list containing non-normalized steering angles
    """
    svals = GetJSONSteeringAngles(filemask)
    svalscp = [element * nc for element in svals]
    values = len(svalscp)
    mean = ("%.2f" % statistics.mean(svals))
    std = ("%.2f" % statistics.stdev(svals))
    plt.title=(pname)
    # NB Plotted as normalized histogram
    sns.distplot(svalscp, bins=nc*2, kde=False, norm_hist=True,
    axlabel= pname + ' steer. degs. norm. hist. ' + str(values) + ' values, mean = ' + mean + ' std = ' + std)
    #if(save):
    #    sns.save("output.png")
    plt.savefig(pname + '.png')

    # return for downstream processing if required
    return svals

def listSteeringBins(svals, pname="output", save=True, nc=25):
    """
    Plot a steering values' histogram
    Inputs
        svals: list, array of normalized steering values
        pname: string, output plot name
        save: boolean, save plot to disk
        nc: int, normalization constant, used in the simulator to put angles in range
        -1, 1. Default is 25.
    Outputs
        none
    """
    svalscp = [element * nc for element in svals]
    values = len(svalscp)
    mean = ("%.2f" % statistics.mean(svals))
    std = ("%.2f" % statistics.stdev(svals))
    plt.title=(pname)
    # NB Plotted as normalized histogram
    sns.distplot(svalscp, bins=nc*2, kde=False, norm_hist=True,
    axlabel= pname + ' steer. degs. norm. hist. ' + str(values) + ' values, mean = ' + mean + ' std = ' + std)
    #if(save):
    #    sns.save("output.png")
    plt.savefig(pname + '.png')

    # return for downstream processing if required
    return svals
```

```
svals = jsonSteeringBins('~/git/msc-data/unity/genRoad/*.jpg', 'genRoad')


# In[ ]:




# In[52]:


def printTimeMS(nf, fr):
    """
    Print time in minutes and seconds.
    Inputs
        nf: integer, number of frames
        fr: float, frame rate
    Outputs
        None
    """
    total_secs = nf / fr
    minutes = int(nf / fr / 60)
    seconds = round(nf / fr / 60 % minutes * 60)
    print("{}m{}s".format(minutes, seconds))

printTimeMS(45410, 24)
printTimeMS(280727, 24)



# In[54]:




# In[87]:


# simpler
import datetime
str(datetime.timedelta(seconds=280727/24))


# In[4]:


svals = GetJSONSteeringAngles('~/git/msc-data/unity/genRoad/*.jpg')
values = len(svals)
# Unity steering angle
UnityMaxSteering = 25
svalscp = [element * UnityMaxSteering for element in svals]
```

```
# NB Plotted as normalized histogram
sns.distplot(svalscp, bins=50, kde=False, norm_hist=True,
             axlabel='Steering Angles (degrees) norm. hist. ' + str(values) + " data points")


# In[99]:


def printJSONSteeringAngles(filemask):
    """
    Get steering angles stored as 'user/angle' attributes in .json files
    Inputs:
        filemask: string, path and mask
    Outputs
        svals: list, steering values
    """
    filemask = os.path.expanduser(filemask)
    path, mask = os.path.split(filemask)

    matches = []
    for root, dirnames, filenames in os.walk(path):
        for filename in fnmatch.filter(filenames, mask):
            matches.append(os.path.join(root, filename))

    # steering values
    svals = []
    for fullpath in matches:
            frame_number = os.path.basename(fullpath).split("_")[0]
            json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
            jobj = load_json(json_filename)
            svals.append(jobj['user/angle'])
    return svals
    # return svals
svals = printJSONSteeringAngles('~/git/sdsandbox/sdsim/log/*.jpg')
cnt = 0
sec = 0
for vals in svals:
    cnt += 1
    if(cnt % 24 == 0):
        sec += 1
        print("Frame {} ========================".format(sec))
    print(vals)


# In[116]:


UnityMaxSteering = 25
svalscp = [element * UnityMaxSteering for element in svals]
my_iterator = filter(lambda svalscp: svalscp <= 20 and svalscp >= -20, svalscp)

svals2020 = list(my_iterator)

rm = len(svals) - len(svals2020)
```

```
pctrm =  rm * 100  / len(svals)
print("Removed entries outside -20 + 20 range: ", len(svals) - len(svals2020))
print("%.2f" % pctrm)


# In[121]:


# mean and std
import statistics
print("%.2f" % statistics.mean(svals2020))
print("%.2f" % statistics.stdev(svals2020))


# In[6]:


########################
# SkewClenaup.png
########################


# genRoad outlier cleanup
# NB Name is last one plotted
# plot everything
# used in SkewCleanup.png
import os
# plot all - nice effect but breaks kernel
# Now let's do one plot per folder to find out where these outliers are
path = '~/git/msc-data/unity/genRoad' # both work
path = '../../dataset/unity/genRoad/'
dirs = os.walk(path)
for mydir in dirs:
    if (len(mydir[1]) == 0):
        fn = mydir[0].split('/')
        path = mydir[0] + '/*.jpg'
        svals = jsonSteeringBins(path, fn[-1])


# deleted empties logs_Thu_Jul__9_16_12_28_2020
# both ~/git ... and ../.. work ok
# \../../dataset/unity/genRoad/logs_Thu_Jul__9_12_25_38_2020/*.jpg', 'test')


# In[20]:


import os
# plot all - nice effect but breaks kernel
# Now let's do one plot per folder to find out where these outliers are
path = '~/git/msc-data/unity/genRoad' # both work
path = '../../dataset/unity/genRoad/'
dirs = os.walk(path)
for mydir in dirs:
```

```
    if (len(mydir[1]) == 0):
        fn = mydir[0].split('/')
        path = mydir[0] + '/*.jpg'
        print("svals = jsonSteeringBins('{}', '{}')".format(path, fn[-1]))
        # svals = jsonSteeringBins(path, fn[-1])


# In[34]:


# find skewer
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Fri_Jul_10_09_29_13_2020/*.jpg', 'logs_Fri_Jul_10_09
_29_13_2020')
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Thu_Jul__9_15_47_22_2020/*.jpg', 'logs_Thu_Jul__9_1
5_47_22_2020')
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Thu_Jul__9_16_06_19_2020/*.jpg', 'logs_Thu_Jul__9_1
6_06_19_2020')
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Thu_Jul__9_15_53_22_2020/*.jpg', 'logs_Thu_Jul__9_1
5_53_22_2020')
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Fri_Jul_10_09_32_12_2020/*.jpg', 'logs_Fri_Jul_10_09
_32_12_2020')
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Thu_Jul__9_12_57_15_2020/*.jpg', 'logs_Thu_Jul__9_1
2_57_15_2020')
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Thu_Jul__9_13_03_17_2020/*.jpg', 'logs_Thu_Jul__9_1
3_03_17_2020')
# svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Thu_Jul__9_12_33_37_2020/*.jpg', 'logs_Thu_Jul__9_
12_33_37_2020')
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Thu_Jul__9_14_59_57_2020/*.jpg', 'logs_Thu_Jul__9_1
4_59_57_2020')
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Fri_Jul_10_09_22_57_2020/*.jpg', 'logs_Fri_Jul_10_09
_22_57_2020')
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Thu_Jul__9_16_08_45_2020/*.jpg', 'logs_Thu_Jul__9_1
6_08_45_2020')
svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Thu_Jul__9_16_00_15_2020/*.jpg', 'logs_Thu_Jul__9_16
_00_15_2020')
#svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020/*.jpg', 'logs_Fri_Jul_10_09
_16_18_2020')
# svals = jsonSteeringBins('../../dataset/unity/genRoad/logs_Thu_Jul__9_12_25_38_2020/*.jpg', 'logs_Thu_Jul__9_
12_25_38_2020')


# In[35]:


# moved skewing logs
# mv genRoad/logs_Thu_Jul__9_16_00_15_2020 quarantine/
svals = jsonSteeringBins('../../dataset/unity/genRoad/*.jpg', 'logs_Fri_Jul_10_09_29_13_2020')


# In[ ]:
```

```
##################################################
# 18. steerlib.ipynb.py
##################################################


#!/usr/bin/env python
# coding: utf-8

# In[10]:


# steerlib - Helper library to create videos and plots

# modules
import fnmatch
import json
import seaborn as sns
import os
import numpy as np
import argparse
import matplotlib.pyplot as plt
import statistics
import seaborn as sns
import pickle


# In[15]:


def load_json(filepath):
    """
    Load a json file
    Inputs
        filepath: string, path to file
    Outputs
        data: dictionary, json key, value pairs
    Example
    path = "~/git/msc-data/unity/roboRacingLeague/log/logs_Sat_Nov_14_12_36_16_2020/record_11640.json"
    js = load_json(path)
    """
    with open(filepath, "rt") as fp:
        data = json.load(fp)
    return data


# In[16]:


def GetJSONSteeringAngles(filemask):
    """
    Get steering angles stored as 'user/angle' attributes in .json files
    Inputs:
```

```
    filemask: string, path and mask
  Outputs
    svals: list, steering values
  """
  filemask = os.path.expanduser(filemask)
  path, mask = os.path.split(filemask)

  matches = []
  for root, dirnames, filenames in os.walk(path):
    for filename in fnmatch.filter(filenames, mask):
      matches.append(os.path.join(root, filename))

  # steering values
  svals = []
  for fullpath in matches:
      frame_number = os.path.basename(fullpath).split("_")[0]
      json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
      jobj = load_json(json_filename)
      svals.append(jobj['user/angle'])
  return svals


# In[64]:


def jsonSteeringBins(filemask, pname="output", save=True, nc=25, rmout=0):
  """
  Plot a steering values' histogram
  Inputs
    filemask: string, where to search for images, and corresponding .json files
    pname: string, output plot name
    save: boolean, save plot to disk
    nc: int, normalization constant, used in the simulator to put angles in range
    -1, 1. Default is 25.
    rmout: integer, outlier range to remove
  Outputs
    svals: list containing non-normalized steering angles
  Example:
  # svals = jsonSteeringBins('~/git/msc-data/unity/genRoad/*.jpg', 'genRoad', save=True, nc=25, rmout=20)
  """
  svals = GetJSONSteeringAngles(filemask)
  values = len(svals)
  svalscp = [element * nc for element in svals]
  if(rmout>0):
    my_iterator = filter(lambda svalscp: svalscp <= rmout and svalscp >= (-1 * rmout), svalscp)
    svalsrmout = list(my_iterator)
    svalscp = svalsrmout
    values = len(svalsrmout)
    print("Removed {} records".format(len(svals) - len(svalsrmout)))
    svals = svalsrmout
  mean = ("%.2f" % statistics.mean(svalscp))
  std = ("%.2f" % statistics.stdev(svalscp))
  plt.title=(pname)
  # NB Plotted as normalized histogram
```

```
    sns.distplot(svalscp, bins=nc*2, kde=False, norm_hist=True,
    axlabel= pname + ' st. degs. norm. hist. ' + str(values) + ' values, mean = ' + mean + ' std = ' + std)
    #if(save):
    #    sns.save("output.png")
    if(save):
        plt.savefig(pname + '.png')
    plt.show()
    # return for downstream processing if required
    return svals


# In[63]:


def removeOutliers(svals, rmout, nc):
    """
    Remove outliers from a list
    Inputs
        svals: double, steering values
        rmout: integer, -+ range tp remove
        nc: steering normalization constant - same as used in simulator (max steering)
    Output
        svals: list, list with values excluded
    """
    svalscp = [element * nc for element in svals]
    my_iterator = filter(lambda svalscp: svalscp <= rmout and svalscp >= (-1 * rmout), svalscp)
    svalsrmout = list(my_iterator)
    svalscp = svalsrmout
    values = len(svalsrmout)
    print("Removed {} records".format(len(svalscp) - len(svalsrmout)))
    svals = svalsrmout
    return svals


# In[65]:


def listSteeringBins(svals, pname="output", save=True, nc=25, rmout=0):
    """
    Plot a steering values' histogram
    Inputs
        svals: list, array of normalized steering values
        pname: string, output plot name
        save: boolean, save plot to disk
        nc: int, normalization constant, used in the simulator to put angles in range
        -1, 1. Default is 25.
        rmout: integer, outlier range to remove
    Outputs
        none
    """
    svalscp = [element * nc for element in svals]
    values = len(svals)

    # remove outliers
```

```
    if(rmout>0):
        #my_iterator = filter(lambda svalscp: svalscp <= rmout and svalscp >= (-1 * rmout), svalscp)
        #svalsrmout = list(my_iterator)
        #svalscp = svalsrmout
        #values = len(svalsrmout)
        #print("Removed {} records".format(len(svals) - len(svalsrmout)))
        #svals = svalsrmout
        svals = removeOutliers(svalscp, rmout, nc)
        values = len(svals)
    mean = ("%.2f" % statistics.mean(svalscp))
    std = ("%.2f" % statistics.stdev(svalscp))
    plt.title=(pname)
    # NB Plotted as normalized histogram
    sns.distplot(svalscp, bins=nc*2, kde=False, norm_hist=True,
    axlabel= pname + ' steer. degs. norm. hist. ' + str(values) + ' values, mean = ' + mean + ' std = ' + std)
    #if(save):
    #    sns.save("output.png")
    if(save):
        plt.savefig(pname + '.png')
    plt.show()


# In[66]:


filemask = '~/git/msc-data/unity/genRoad/*.jpg'
svals = GetJSONSteeringAngles(filemask)
listSteeringBins(svals, pname="outputExOut20", save=True, nc=25, rmout=20)


# In[58]:


filemask = '~/git/msc-data/unity/genRoad/*.jpg'
svals = GetJSONSteeringAngles(filemask)
listSteeringBins(svals, pname="output", save=True, nc=25, rmout=0)


# In[68]:


# generate training dataset latex tables for report
for folder in ['genRoad','log_sample', 'roboRacingLeague','smallLoop', 'smallLoopingCourse','warehouse']:
    print(folder)

"""
SDSandbox   unity/smallLoopingCourse/log/* 34443 from small\_looping\_course
SDSandbox   unity/warehouse/*   41126 From Warehouse course
SDSandbox   unity/smallLoop/*   45422   From small\_looping\_course
SDSandbox   unity/roboRacingLeague/* 12778 From "Robot Racing League" course
SDSandbox   unity/log\_sample   25791   From small\_looping\_course
SDSandbox   unity/genRoad 280727 From "Generated Road" course
"""
```

In[ ]:

\begin{table}[]
\begin{center}
\begin{tabular}{|l|l|l|l|}
\hline
\multicolumn{4}{|c|}{Deliverables - Datasets} \\ \hline

get_ipython().run_line_magic('ID', 'Task Deliverable Description')
get_ipython().run_line_magic('1', 'Download D1 Udacity real world dataset')
get_ipython().run_line_magic('2', 'Generate   D2 Unity3D simulator data')
get_ipython().run_line_magic('3', 'Combine D3 Udacity real and simulator data')
get_ipython().run_line_magic('4', 'Mechanical Turk dry/rainy Ford dataset')

ID & Task & Deliverable & Description \\ \hline\hline
1 & Download & D1 & Udacity real world dataset \\ \hline
2 & Generate & D2 & Udacity simulator data \\ \hline
3 & Combine & D3 & Udacity real and simulator data \\ \hline
4 & Gather & D4 & Mechanical Turk dry/rainy Ford dataset \\ \hline

\end{tabular}
\end{center}
\caption{Datasets used to train models}
\label{Deliverables-Datasets}
\end{table}

##
19. steerlib.py
##

Helper library to create videos and plots

import fnmatch
import json
import os
import numpy as np
import matplotlib.pyplot as plt
import statistics
import seaborn as sns
import pickle
from PIL import Image
# prediction
import tensorflow as tf
from tensorflow.python import keras
from tensorflow.python.keras.models import load_model
# rain
from predict_client import add_rain
# Augmentation library
import Augmentation

```
def load_json(filepath):
    """
    Load a json file
    Inputs
        filepath: string, path to file
    Outputs
        data: dictionary, json key, value pairs
    Example
    path = "~/git/msc-data/unity/roboRacingLeague/log/logs_Sat_Nov_14_12_36_16_2020/record_11640.json"
    js = load_json(path)
    """
    with open(filepath, "rt") as fp:
        data = json.load(fp)
    return data

def GetSteeringFromtcpflow(filename):
    """
    Get a tcpflow log and extract steering values obtained from network communication between.
    Note, we only plot the predicted steering angle jsondict['steering']
    and the value of jsondict['steering_angle'] is ignored. Assumed to be the steering angle
    calculated by PID given the current course.
    sim and prediction engine (predict_client.py)
    Inputs
        filename: string, name of tcpflow log
    Returns
        sa: list of arrays, steering angle predicton and actual value tuple.
    Example


    """
    # open file
    sa = []
    # initialize prediction
    pred = ''
    f = open(filename, "r")
    file = f.read()
    try:
        # readline = f.read()
        lines = file.splitlines()
        for line in lines:
            # print(line)
            start = line.find('{')
            if (start == -1):
                continue
            jsonstr = line[start:]
            # print(jsonstr)
            jsondict = json.loads(jsonstr)
            if "steering" in jsondict:
                # predicted
                pred = jsondict['steering']
                # jsondict['steering_angle']
                # sa.append([float(pred), act])
                sa.append([float(pred), float(pred)])  # append twice to keep code from breaking
```

```
            # if "steering_angle" in jsondict:
            # actual
            #   act = jsondict['steering_angle']
            # save pair, only keep last pred in case two were send as it does happen i.e.:
            # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.071960375", "thr
ottle": "0.08249988406896591", "brake": "0.0"}
            # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.079734944", "thr
ottle": "0.08631626516580582", "brake": "0.0"}
            # 127.000.000.001.09091-127.000.000.001.59460: {"msg_type":"telemetry","steering_angle":-0.07196037,(.
..)
            #   if(len(pred) > 0):
            #      sa.append([float(pred), act])
            #      pred = '' # need to save this image
            # deal with image later, sort out plot first
            # imgString = jsondict["image"]
            # image = Image.open(BytesIO(base64.b64decode(imgString)))
            # img_arr = np.asarray(image, dtype=np.float32)
    except Exception as e:
        print("Exception raise: " + str(e))
    # file should be automatically closed but will close for good measure
    f.close()
    return sa

def GetJSONSteeringAngles(filemask):
    """
    Get steering angles stored as 'user/angle' attributes in .json files
    Inputs:
        filemask: string, path and mask
    Outputs
        svals: list, steering values
    """
    filemask = os.path.expanduser(filemask)
    path, mask = os.path.split(filemask)

    matches = []
    for root, dirnames, filenames in os.walk(path):
        for filename in fnmatch.filter(filenames, mask):
            matches.append(os.path.join(root, filename))
    # sort by create date
    matches = sorted(matches, key=os.path.getmtime)
    # steering values
    svals = []
    for fullpath in matches:
            frame_number = os.path.basename(fullpath).split("_")[0]
            json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
            jobj = load_json(json_filename)
            svals.append(jobj['user/angle'])
    return svals

def jsonSteeringBins(filemask, pname="output", save=True, nc=25):
    """
    Plot a steering values' histogram
    Inputs
        filemask: string, where to search for images, and corresponding .json files
```

```
        pname: string, output plot name
        save: boolean, save plot to disk
        nc: int, normalization constant, used in the simulator to put angles in range
        -1, 1. Default is 25.
    Outputs
        svals: list containing non-normalized steering angles
    Example:
    # svals = jsonSteeringBins('~/git/msc-data/unity/genRoad/*.jpg', 'genRoad')
    """
    svals = GetJSONSteeringAngles(filemask)
    values = len(svals)
    svalscp = [element * nc for element in svals]
    mean = ("%.2f" % statistics.mean(svals))
    std = ("%.2f" % statistics.stdev(svals))
    plt.title=(pname)
    # NB Plotted as normalized histogram
    sns.distplot(svalscp, bins=nc*2, kde=False, norm_hist=True,
    axlabel= pname + ' steer. degs. norm. hist. ' + str(values) + ' values, mean = ' + mean + ' std = ' + std)
    #if(save):
    #    sns.save("output.png")
    plt.savefig(pname + '.png')

    # return for downstream processing if required
    return svals

def plot_hist(path):
    """
    Create loss/accuracy plot for training and save to disk
    Inputs
        path: file to history pickle file
    Outputs
        none
    Example:
    path = "/home/simbox/git/sdsandbox/trained_models/nvidia1/20201107144927_nvidia1.history"
    plot_hist(path)
    or
    $ python steerlib.py
    $ python
    $ >>> import steerlib
    $ >>> path = "/home/simbox/git/sdsandbox/trained_models/nvidia1/20201107144927_nvidia1.history"
    $ >>> plot_hist(path)
    """
    history = pickle.load(open(path, "rb"))
    # type(history)
    # <class 'dict'>
    # history.keys()
    # dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
    do_plot = True
    try:
        if do_plot:
            fig = plt.figure()  # when loading dictionary keys, we omit .history (see train.py)
            plot_name = path.split('/')[-1]
            sp = plot_name \
                + ' - ' + '(l,vl,a,va)' + '{0:.3f}'.format(history['loss'][-1]) \
```

```
            + ',' + '{0:.3f}'.format(history['val_loss'][-1]) \
            + ',' + '{0:.3f}'.format(history['acc'][-1]) \
            + ',' + '{0:.3f}'.format(history['val_acc'][-1])

        fig.suptitle(sp, fontsize=9)
        ax = fig.add_subplot(111)
        ax.plot(history['loss'], 'r-', label='Training Loss', )
        ax.plot(history['val_loss'], 'm-', label='Validation Loss')
        ax2 = ax.twinx()
        ax2.plot(history['acc'], '-', label='Training Accuracy')
        ax2.plot(history['val_acc'], '-', label='Validation Accuracy')
        ax.legend(loc=2) # https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.legend.html
        ax.grid()
        ax.set_xlabel("Epoch")
        ax.set_ylabel(r"Loss")
        ax2.set_ylabel(r"Accurary")
        ax.set_ylim(0, 0.2)
        ax2.set_ylim(0.5, 1)
        ax2.legend(loc=1)
        aimg = plot_name.split('.')[0]
        aimg = str(aimg)  + '_accuracy.png'
        plt.savefig(aimg)
    except Exception as e:
        print("Failed to save accuracy/loss graph: " + str(e))

def gos(p, g, n):
    """
    Calculate the goodness-of-steer between a prediction and a ground truth array.
    Inputs
        p: array of floats, steering angle prediction
        g: array of floats, steering angle ground truth.
        n: float, normalization constant
    Output
        gos: float, average of absolute difference between ground truth and prediction arrays
    """
    # todo add type assertion
    assert len(p) == len(g), "Arrays must be of equal length"
    return sum(abs(p - g)) / len(p) * n
    # print("Goodness of steer: {:.2f}".format(steer))

def plotSteeringAngles(p, g=None, n=1, save=False, track= "Track Name", mname="model name", title='title'):
    """
    Plot predicted and (TODO) optionally ground truth steering angles
    Inputs
        p, g: prediction and ground truth float arrays
        n: float, steering normalization constant
        save: boolean, save plot flag
        track, mname, title: string, track (data), trained model and title strings for plot
    Outputs
        plt: pyplot, plot
    Example
    # set argument variables (see data_predict.py)
    plotSteeringAngles(p, g, nc, True, datapath[-2], modelpath[-1], 'Gs ' + gss)
    """
```

```
    plt.rcParams["figure.figsize"] = (18,3)

    plt.plot(p*n, label="predicted")
    try:
        if (g is not None):
            plt.plot(g*n, label="ground truth")
    except Exception as e:
        print("problems plotting: " + str(e))

    plt.ylabel('Steering angle')
    plt.xlabel('Frame number')
    # Set a title of the current axes.
    # plt.title('tcpflow log predicted steering angles: track ' + track + ' model ' + mname)
    plt.title(title + ' Steering angles: track ' + track + ', model ' + mname)
    # show a legend on the plot
    plt.legend()
    # Display a figure.
    # horizontal grid only
    plt.grid(axis='y')
    # set limit
    plt.xlim([-5,len(p)+5])
    plt.gca().invert_yaxis()
    # plt.show()
    if(save==True):
        plt.savefig('sa_' + track + '_' + mname + '.png')
    # if need be
    return plt

def plotMultipleSteeringAngles(p, n=25, save=False, track="Track Name", mname="model name", title='title', w=1
8, h=3):
    """
    Plot multiple predicted and (TODO) optionally ground truth steering angles
    Inputs
        p: list of tuples, prediction and labels
        n: float, steering normalization constant
        save: boolean, save plot flag
        track, mname, title: string, track (data), trained model and title strings for plot
        w: integer, plot width
        h: integer, plot height
    Outputs
        plt: pyplot, plot
    Example
    # get some steering angles
    sa = GetSteeringFromtcpflow('../trained_models/nvidia1/tcpflow/20201207091932_nvidia1_no_rain_tcpflow.log'
)
    sarr = np.asarray(sa)
    pa = sarr[:,0]
    p.append([pa, 'no rain'])
    plotSteeringAngles(p, g, nc, True, datapath[-2], modelpath[-1], 'Gs ' + gss)
    """
    import matplotlib.pyplot as plt # local copy
    plt.rcParams["figure.figsize"] = (w,h)
```

```
    for i in range (0, len(p)):
        plt.plot(p[i][0]*n, label=p[i][1])
    #try:
      #  if (g is not None):
     #       plt.plot(g*n, label="ground truth")
    #except Exception as e:
    #    print("problems plotting: " + str(e))

    plt.ylabel('Steering angle')
    plt.xlabel('Frame number')
    # Set a title of the current axes.
    # plt.title('tcpflow log predicted steering angles: track ' + track + ' model ' + mname)
    plt.title(title + ' Steering angles: track ' + track + ', model ' + mname)
    # show a legend on the plot
    plt.legend()
    # Display a figure.
    # horizontal grid only
    plt.grid(axis='y')
    # set limit
    plt.xlim([-5,len(p[0][0])+5])
    plt.gca().invert_yaxis()
    # plt.show()
    if(save==True):
        plt.savefig('sa_' + track + '_' + mname + '.png')
    # if need be
    return plt

def getSteeringFromtcpflow(filename):
    """
    Get a tcpflow log and extract steering values obtained from network communication between sim and predict_clie
nt.py.
    Note, we only plot the predicted steering angle jsondict['steering']
    and the value of jsondict['steering_angle'] is ignored. Assumed to be the steering angle
    calculated by PID given the current course.
    sim and prediction engine (predict_client.py)
    Inputs
        filename: string, name of tcpflow log
    Returns
        sa: list of arrays, steering angle predicton and actual value tuple.
    Example


    """
    # open file
    sa = []
    # initialize prediction
    pred = ''
    f = open(filename, "r")
    file = f.read()
    try:
        # readline = f.read()
        lines = file.splitlines()
        for line in lines:
            # print(line)
```

```
            start = line.find('{')
            if (start == -1):
                continue
            jsonstr = line[start:]
            # print(jsonstr)
            jsondict = json.loads(jsonstr)
            if "steering" in jsondict:
                # predicted
                pred = jsondict['steering']
                # jsondict['steering_angle']
                # sa.append([float(pred), act])
                sa.append([float(pred), float(pred)])  # append twice to keep code from breaking
            # if "steering_angle" in jsondict:
            # actual
            #   act = jsondict['steering_angle']
            # save pair, only keep last pred in case two were send as it does happen i.e.:
            # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.071960375", "thr
ottle": "0.08249988406896591", "brake": "0.0"}
            # 127.000.000.001.59460-127.000.000.001.09091: {"msg_type": "control", "steering": "-0.079734944", "thr
ottle": "0.08631626516580582", "brake": "0.0"}
            # 127.000.000.001.09091-127.000.000.001.59460: {"msg_type":"telemetry","steering_angle":-0.07196037,(.
..)
            #   if(len(pred) > 0):
            #      sa.append([float(pred), act])
            #      pred = '' # need to save this image
            # deal with image later, sort out plot first
            # imgString = jsondict["image"]
            # image = Image.open(BytesIO(base64.b64decode(imgString)))
            # img_arr = np.asarray(image, dtype=np.float32)
    except Exception as e:
        print("Exception raise: " + str(e))
    # file should be automatically closed but will close for good measure
    f.close()
    return sa

"""

# get image

# get steering

# add / don't add rain

outputs = self.model.predict(image_array)

Append groundtruth, predicted to list.
"""
def PrintLatexRowModelGOS(filemask, modelpath, modelname, rt='', st=0):
    """
    Generate a "goodness of fit value" for a model on a given track
    Inputs
        filemask: string, path and mask
        modelpath: string, path to keras model
        modelname: string, canonical model name e.g. nvidia1, nvidia2, nvidia_baseline
```

```
    rt: string, rain type e.g. drizzle/light, heavy torrential
    st: integer, -+20 degree rain slant
Outputs
    svals: list, ground truth steering values and predictions
"""
# load augmentation library for correct model geometry
ag = Augmentation.Augmentation(modelname)

# load model
print("loading model", modelpath)
model = load_model(modelpath)

# In this mode, looks like we have to compile it
model.compile("sgd", "mse")
filemask = os.path.expanduser(filemask)
path, mask = os.path.split(filemask)

matches = []
for root, dirnames, filenames in os.walk(path):
    for filename in fnmatch.filter(filenames, mask):
        matches.append(os.path.join(root, filename))

# steering values
svals = []
for fullpath in matches:
    frame_number = os.path.basename(fullpath).split("_")[0]
    json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
    jobj = load_json(json_filename)
    # steering ground truth
    steer_gt = jobj['user/angle']
    # open the image
    img_arr = Image.open(fullpath)
    # Convert PIL Image to numpy array
    img_arr = np.array(img_arr, dtype=np.float32)
    # add rain if need be
    if rt != '':
        img_arr = add_rain(img_arr, rt, st)
    # apply same preprocessing
    # same preprocessing as for training
    img_arr = ag.preprocess(img_arr)

    # put in correct format
    img_arr = img_arr.reshape((1,) + img_arr.shape)
    # generate prediction
    outputs = model.predict(img_arr)
    # store predictions
    steer_pred = outputs[0][0]
    # store ground truth and prediction
    svals.append([steer_pred, steer_gt])
# get goodness of fit
sarr = np.asarray(svals)
p = sarr[:, 0]
g = sarr[:, 1]
nc = 25 # unity maximum steering angle / normalization constant - should hold in conf.py and managed with Aug
```

```
mentation
    mygos = gos(p, g, nc)
    # format to human readable/friendlier 2 decimal places
    gos_str = "{:.2f}".format(round(mygos, 2))
    # strip path from modelpath
    modelfile = modelpath.split('/')
    modelfile = modelfile[-1]
    # print latex formated data
    # header
    hd_str = 'Filename & Model & Rain Type & Slant & gos \\\\ \hline'
    # log file
    print('Log: ', path, '\\\\ \hline')
    print(hd_str)
    # results
    res_str = f'{modelfile} & {modelname} & {rt} & {st} & {gos_str} \\\\ \hline'
    print(res_str)

def GetPredictedSteeringAngles(filemask, model, modelname, rt='', st=0):
    """
    Generate a "goodness of fit value" for a model on a given track
    Inputs
        filemask: string, path and mask
        modelpath: string, path to keras model
        modelname: string, canonical model name e.g. nvidia1, nvidia2, nvidia_baseline
        rt: string, rain type e.g. drizzle/light, heavy torrential
        st: integer, -+20 degree rain slant
    Outputs
        svals: list, ground truth steering values and predictions
    """
    # load augmentation library for correct model geometry
    ag = Augmentation.Augmentation(modelname)

    # load model
    # print("loading model", modelpath)
    # assume model is loaded and compiled
    # model = load_model(modelpath)

    # In this mode, looks like we have to compile it
    # model.compile("sgd", "mse")
    filemask = os.path.expanduser(filemask)
    path, mask = os.path.split(filemask)

    matches = []
    for root, dirnames, filenames in os.walk(path):
        for filename in fnmatch.filter(filenames, mask):
            matches.append(os.path.join(root, filename))

    # sort by create date
    matches = sorted(matches, key=os.path.getmtime)
    # steering values
    svals = []
    for fullpath in matches:
        frame_number = os.path.basename(fullpath).split("_")[0]
        json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
```

```
        jobj = load_json(json_filename)
        # steering ground truth
        steer_gt = jobj['user/angle']
        # open the image
        img_arr = Image.open(fullpath)
        # Convert PIL Image to numpy array
        img_arr = np.array(img_arr, dtype=np.float32)
        # add rain if need be
        if rt != '':
            img_arr = add_rain(img_arr, rt, st)
        # apply same preprocessing
        # same preprocessing as for training
        img_arr = ag.preprocess(img_arr)

        # put in correct format
        img_arr = img_arr.reshape((1,) + img_arr.shape)
        # generate prediction
        outputs = model.predict(img_arr)
        # store predictions
        steer_pred = outputs[0][0]
        # store ground truth and prediction
        svals.append([steer_pred, steer_gt])
    return svals

def printGOSRows():
    """
    Print GOS rows for results report.
    We are comparing mainly the two best models for nvidia1 and nvidia2, would have been nice to also test
    the driveable nvidia_baseline, and sanity models
    """
    # models
    # nvidia2 - ../../trained_models/nvidia2/20201207192948_nvidia2.h5
    # nvidia1 - ../../trained_models/nvidia1/20201207091932_nvidia1.h5
    # 20201120124421\_nvidia\_baseline.h5
    # log logs_Wed_Nov_25_23_39_22_2020
    ##############################################
    # 1. nvidia2 Generated track
    ##############################################
    # log = '../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020/*.jpg'
    #modelpath = '../../trained_models/nvidia2/20201207192948_nvidia2.h5'
    #modelname = 'nvidia2'
    #rt = 'torrential'
    #st = 20
    #PrintLatexRowModelGOS(log, modelpath, modelname, rt, st)
    # Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207192948_nvidia2.h5 & nvidia2 &  & 0 & 1.68 \\ \hline
    # Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207192948_nvidia2.h5 & nvidia2 & light & 0 & 2.12 \\ \hline
    # Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207192948_nvidia2.h5 & nvidia2 & heavy & 10 & 2.17 \\ \hline
    # Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
```

```
# Filename & Model & Rain Type & Slant & gos \\ \hline
# 20201207192948_nvidia2.h5 & nvidia2 & torrential & 20 & 2.30 \\ \hline
#########################################################
# 2. nvidia1 Generated track
#########################################################
# modelpath = '../../trained_models/nvidia1/20201207091932_nvidia1.h5'
#modelname = 'nvidia1'
#rt = 'torrential'
#st = 20
#PrintLatexRowModelGOS(log, modelpath, modelname, rt, st)
# Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
# Filename & Model & Rain Type & Slant & gos \\ \hline
# 20201207091932_nvidia1.h5 & nvidia1 &  & 0 & 1.82 \\ \hline
# Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
# Filename & Model & Rain Type & Slant & gos \\ \hline
# 20201207091932_nvidia1.h5 & nvidia1 & light & 0 & 2.11 \\ \hline
# Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
# Filename & Model & Rain Type & Slant & gos \\ \hline
# 20201207091932_nvidia1.h5 & nvidia1 & heavy & 10 & 2.13 \\ \hline
# Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
# Filename & Model & Rain Type & Slant & gos \\ \hline
# 20201207091932_nvidia1.h5 & nvidia1 & torrential & 20 & 2.28 \\ \hline
#########################################################
# 3. nvidia_baseline Generated track
#########################################################
#modelpath = '../../trained_models/nvidia_baseline/20201207201157_nvidia_baseline.h5'
#modelname = 'nvidia2_baseline'
#rt = 'torrential'
#st = 20
#PrintLatexRowModelGOS(log, modelpath, modelname, rt, st)
# Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
# Filename & Model & Rain Type & Slant & gos \\ \hline
# 20201207201157_nvidia_baseline.h5 & nvidia2_baseline &  & 0 & 2.32 \\ \hline
# Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
# Filename & Model & Rain Type & Slant & gos \\ \hline
# 20201207201157_nvidia_baseline.h5 & nvidia2_baseline & light & 0 & 3.12 \\ \hline
# Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
# Filename & Model & Rain Type & Slant & gos \\ \hline
# 20201207201157_nvidia_baseline.h5 & nvidia2_baseline & heavy & 10 & 3.17 \\ \hline
# Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
# Filename & Model & Rain Type & Slant & gos \\ \hline
# 20201207201157_nvidia_baseline.h5 & nvidia2_baseline & torrential & 20 & 3.39 \\ \hline
#########################################################
# 4. sanity Generated track 20201120171015\_ sanity.h5
#########################################################
#modelpath = '../../trained_models/sanity/20201120171015_sanity.h5'
#modelname = 'nvidia1'
#rt = 'torrential'
#st = 20
#PrintLatexRowModelGOS(log, modelpath, modelname, rt, st)
# Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
# Filename & Model & Rain Type & Slant & gos \\ \hline
# 20201120171015_sanity.h5 & nvidia1 &  & 0 & 5.03 \\ \hline
# Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
```

```
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201120171015_sanity.h5 & nvidia1 & light & 0 & 3.11 \\ \hline
    # Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201120171015_sanity.h5 & nvidia1 & heavy & 10 & 3.07 \\ \hline
    # Log:  ../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201120171015_sanity.h5 & nvidia1 & torrential & 20 & 3.00 \\ \hline

    ################################################
    # 5. nvidia2 Generated Road
    ################################################
    log = '../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020/*.jpg'
    #modelpath = '../../trained_models/nvidia2/20201207192948_nvidia2.h5'
    #modelname = 'nvidia2'
    #rt = 'torrential'
    #st = 20
    #PrintLatexRowModelGOS(log, modelpath, modelname, rt, st)
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207192948_nvidia2.h5 & nvidia2 &  & 0 & 2.99 \\ \hline # drove 16 minutes on this road https://youtu.be
/z9nILq9dQfI
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207192948_nvidia2.h5 & nvidia2 & light & 0 & 3.20 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207192948_nvidia2.h5 & nvidia2 & heavy & 10 & 3.22 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207192948_nvidia2.h5 & nvidia2 & torrential & 20 & 3.27 \\ \hline
    ########################################################
    # 6. nvidia1 Generated Road
    ########################################################
    #modelpath = '../../trained_models/nvidia1/20201207091932_nvidia1.h5'
    #modelname = 'nvidia1'
    # rt = 'torrential'
    # st = 20
    # PrintLatexRowModelGOS(log, modelpath, modelname, rt, st)
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207091932_nvidia1.h5 & nvidia1 &  & 0 & 3.87 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207091932_nvidia1.h5 & nvidia1 & light & 0 & 3.75 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207091932_nvidia1.h5 & nvidia1 & heavy & 10 & 3.70 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207091932_nvidia1.h5 & nvidia1 & torrential & 20 & 3.57 \\ \hline

    ########################################################
    # 7. nvidia_baseline Generated Road
```

```
    ########################################################
    #modelpath = '../../trained_models/nvidia_baseline/20201207201157_nvidia_baseline.h5'
    #modelname = 'nvidia2_baseline'
    #rt = 'torrential'
    #st = 20
    #PrintLatexRowModelGOS(log, modelpath, modelname, rt, st)
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207201157_nvidia_baseline.h5 & nvidia2_baseline &  & 0 & 5.51 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207201157_nvidia_baseline.h5 & nvidia2_baseline & light & 0 & 4.97 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207201157_nvidia_baseline.h5 & nvidia2_baseline & heavy & 10 & 4.98 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201207201157_nvidia_baseline.h5 & nvidia2_baseline & torrential & 20 & 5.05 \\ \hline
    ########################################################
    # 8. sanity Generated track 20201120171015\_ sanity.h5
    ########################################################
    modelpath = '../../trained_models/sanity/20201120171015_sanity.h5'
    modelname = 'nvidia1'
    rt = 'torrential'
    st = 20
    PrintLatexRowModelGOS(log, modelpath, modelname, rt, st)
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201120171015_sanity.h5 & nvidia1 &  & 0 & 3.85 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201120171015_sanity.h5 & nvidia1 & light & 0 & 3.06 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201120171015_sanity.h5 & nvidia1 & heavy & 10 & 3.05 \\ \hline
    # Log:  ../../dataset/unity/genRoad/logs_Fri_Jul_10_09_16_18_2020 \\ \hline
    # Filename & Model & Rain Type & Slant & gos \\ \hline
    # 20201120171015_sanity.h5 & nvidia1 & torrential & 20 & 3.02 \\ \hline

def printMultiPlots(model1_nvidia2, model2_nvidia1, model3_nvidia_baseline, model4_sanity):
    """
    Print multiple plots to reference first 16 results in Goodness of steer tables
    Inputs
     model1_nvidia2, model2_nvidia1, model3_nvidia_baseline, model4_sanity: the required keras models
    """
    # init plot list
    plot_list = []
    # define log
    log = '../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020/*.jpg'
    # Get ground truth values
    gt = GetJSONSteeringAngles(log)
    gt = np.asarray(gt)
    # get predictions
    ############################################
```

```
    # 1. nvidia2 Generated track
    ############################################
    # 1.1 dry
    """
    print('Predicting nvidia2 dry...')
    sa = GetPredictedSteeringAngles(log, model1_nvidia2, 'nvidia2', rt='', st=0)
    sarr = np.asarray(sa)
    p = sarr[:,0]
    g = sarr[:, 1]
    plot_list.append([g, 'ground truth'])
    plot_list.append([p, 'prediction - no rain'])
    # 1.2 light rain
    print('Predicting nvidia2 light rain...')
    sa = GetPredictedSteeringAngles(log, model1_nvidia2, 'nvidia2', rt='light', st=0)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    plot_list.append([p, 'prediction - light rain'])
    # 1.3 heavy rain, slant = +-10
    print('Predicting nvidia2 heavy rain...')
    sa = GetPredictedSteeringAngles(log, model1_nvidia2, 'nvidia2', rt='heavy', st=10)
    p = sarr[:,0]
    plot_list.append([p, 'prediction - heavy rain +-10'])
    # 1.4 torrential rain, slant = +-20
    print('Predicting nvidia2 torrential rain...')
    sa = GetPredictedSteeringAngles(log, model1_nvidia2, 'nvidia2', rt='torrential', st=20)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    plot_list.append([p, 'prediction - torrential rain +-20'])
    print('Plotting...')
    plotMultipleSteeringAngles(plot_list, 25, True, "Generated_Track", "20201207192948_nvidia2.h5", title='SDSan
dbox log genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020', w=18, h=4)
    """
    ############################################
    # 2. nvidia1 Generated track
    ############################################
    # 2.1 dry
    print('Predicting nvidia1 dry...')
    sa = GetPredictedSteeringAngles(log, model2_nvidia1, 'nvidia1', rt='', st=0)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    g = sarr[:, 1]
    plot_list.append([g, 'ground truth'])
    plot_list.append([p, 'prediction - no rain'])
    # 2.2 light rain
    print('Predicting nvidia1 light rain...')
    sa = GetPredictedSteeringAngles(log, model2_nvidia1, 'nvidia1', rt='light', st=0)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    plot_list.append([p, 'prediction - light rain'])
    # 2.3 heavy rain, slant = +-10
    print('Predicting nvidia1 heavy rain...')
    sa = GetPredictedSteeringAngles(log, model2_nvidia1, 'nvidia1', rt='heavy', st=10)
    sarr = np.asarray(sa);
    p = sarr[:,0]
```

```
    plot_list.append([p, 'prediction - heavy rain +-10'])
    # 2.4 torrential rain, slant = +-20
    print('Predicting nvidia1 torrential rain...')
    sa = GetPredictedSteeringAngles(log, model2_nvidia1, 'nvidia1', rt='torrential', st=20)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    plot_list.append([p, 'prediction - torrential rain +-20'])
    print('Plotting...')
    plotMultipleSteeringAngles(plot_list, 25, True, "Generated_Track", "20201207091932_nvidia1.h5", title='SDSan
dbox log genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020', w=18, h=4)

    ##############################################
    # 3. nvidia_baseline Generated track
    ##############################################
    # 3.1 dry
    """
    print('Predicting model3_nvidia_baseline dry...')
    sa = GetPredictedSteeringAngles(log, model3_nvidia_baseline, 'nvidia1', rt='', st=0)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    g = sarr[:, 1]
    plot_list.append([g, 'ground truth'])
    plot_list.append([p, 'prediction - no rain'])
    # 3.2 light rain
    print('Predicting model3_nvidia_baseline light rain...')
    sa = GetPredictedSteeringAngles(log, model3_nvidia_baseline, 'nvidia1', rt='light', st=0)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    plot_list.append([p, 'prediction - light rain'])
    # 3.3 heavy rain, slant = +-10
    print('Predicting model3_nvidia_baseline heavy rain...')
    sa = GetPredictedSteeringAngles(log, model3_nvidia_baseline, 'nvidia1', rt='heavy', st=10)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    plot_list.append([p, 'prediction - heavy rain +-10'])
    # 3.4 torrential rain, slant = +-20
    print('Predicting model3_nvidia_baseline torrential rain...')
    sa = GetPredictedSteeringAngles(log, model3_nvidia_baseline, 'nvidia1', rt='torrential', st=20)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    plot_list.append([p, 'prediction - torrential rain +-20'])
    print('Plotting...')
    plotMultipleSteeringAngles(plot_list, 25, True, "Generated_Track", "20201207201157_nvidia_baseline.h5", title
='SDSandbox log genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020', w=18, h=3)
    """

    """
    ##############################################
    # 4. sanity Generated track
    ##############################################
    # 4.1 dry
    print('Predicting model4_sanity dry...')
    sa = GetPredictedSteeringAngles(log, model4_sanity, 'nvidia1', rt='', st=0)
    sarr = np.asarray(sa);
```

```
    p = sarr[:,0]
    g = sarr[:, 1]
    plot_list.append([g, 'ground truth'])
    plot_list.append([p, 'prediction - no rain'])
    # 4.2 light rain
    print('Predicting model4_sanity light rain...')
    sa = GetPredictedSteeringAngles(log, model4_sanity, 'nvidia1', rt='light', st=0)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    plot_list.append([p, 'prediction - light rain'])
    # 4.3 heavy rain, slant = +-10
    print('Predicting model4_sanity heavy rain...')
    sa = GetPredictedSteeringAngles(log, model4_sanity, 'nvidia1', rt='heavy', st=10)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    plot_list.append([p, 'prediction - heavy rain +-10'])
    # 4.4 torrential rain, slant = +-20
    print('Predicting model4_sanity torrential rain...')
    sa = GetPredictedSteeringAngles(log, model4_sanity, 'nvidia1', rt='torrential', st=20)
    sarr = np.asarray(sa);
    p = sarr[:,0]
    plot_list.append([p, 'prediction - torrential rain +-20'])
    print('Plotting...')
    plotMultipleSteeringAngles(plot_list, 25, True, "Generated_Track", "20201120171015_sanity.h5", title='SDSand
box log genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020', w=18, h=3)
    """
if __name__ == "__main__":
    # plot_hist("/home/simbox/git/sdsandbox/trained_models/nvidia1/20201107144927_nvidia1.history")
#if __name__ == "__main__":
#    parser = argparse.ArgumentParser(description='Plot Steering Utils')
#    parser.add_argument('--inputs', type=str, help='file path')

    # args = parser.parse_args()
    #svals = jsonSteeringBins('~/git/msc-data/unity/genRoad/*.jpg', 'genRoad')

    # PrintLatexRowModelGOS('../../dataset/unity/genTrack/genTrackOneLap/logs_Wed_Nov_25_23_39_22_2020/
*.jpg', '../../trained_models/nvidia2/20201207192948_nvidia2.h5', 'nvidia2')
    # printGOSRows()
    # load models
    model1_nvidia2 = ''
    model2_nvidia1 = ''
    model3_nvidia_baseline = ''
    model4_sanity = ''
    # modelpath = '../../trained_models/sanity/20201120171015_sanity.h5'
    # modelpath = '../../trained_models/nvidia_baseline/20201207201157_nvidia_baseline.h5'
    #modelpath = '../../trained_models/nvidia1/20201207091932_nvidia1.h5'
    # modelpath = '../../trained_models/nvidia2/20201207192948_nvidia2.h5'
    # assume model is loaded and compiled
    # nvidia2
    modelpath = '../../trained_models/nvidia2/20201207192948_nvidia2.h5'
    model1_nvidia2 = load_model(modelpath)
    model1_nvidia2.compile("sgd", "mse")
    # nvidia1
    modelpath = '../../trained_models/nvidia1/20201207091932_nvidia1.h5'
```

```
    model2_nvidia1 = load_model(modelpath)
    model2_nvidia1.compile("sgd", "mse")
    # nvidia_baseline
    modelpath = '../../trained_models/nvidia_baseline/20201207201157_nvidia_baseline.h5'
    model3_nvidia_baseline = load_model(modelpath)
    model3_nvidia_baseline.compile("sgd", "mse")
    # sanity
    modelpath = '../../trained_models/sanity/20201120171015_sanity.h5'
    model4_sanity = load_model(modelpath)
    model4_sanity.compile("sgd", "mse")

    printMultiPlots(model1_nvidia2, model2_nvidia1, model3_nvidia_baseline, model4_sanity)

    #path = 'record_11640.json'
    #js = load_json(path)
    #print(js)
    # plotSteeringAngles(p, None, 25, True, "Generated Track", "20201120171015_sanity.h5", 'tcpflow log predicted'
)
    # plots1()


#######################################################
# 20. train.py
# Note: code based on
# https://github.com/tawnkramer/sdsandbox/blob/master/src/train.py
# Available for audit in audit_files/tawn from sharepoint link
#######################################################


'''
Train
Train your nerual network
Author: Tawn Kramer
To fix missing .jpeg files, see utils/
'''
from __future__ import print_function

import argparse
import fnmatch
import json
import os
import pickle
import random
from datetime import datetime
from time import strftime
import numpy as np
from PIL import Image
from tensorflow import keras
# import tensorflow as tf
import conf
import models
from helper_functions import hf_mkdir, parse_bool
from augmentation import augment, preprocess
# import cv2
```

```
import Augmentation

'''
matplotlib can be a pain to setup. So handle the case where it is absent. When present,
use it to generate a plot of training results.
'''
try:
    import matplotlib
    # Force matplotlib to not use any Xwindows backend.
    matplotlib.use('Agg')

    import matplotlib.pyplot as plt
    do_plot = True
except:
    do_plot = False


def shuffle(samples):
    '''
    randomly mix a list and return a new list
    '''
    ret_arr = []
    len_samples = len(samples)
    while len_samples > 0:
        iSample = random.randrange(0, len_samples)
        ret_arr.append(samples[iSample])
        del samples[iSample]
        len_samples -= 1
    return ret_arr

def load_json(filename):
    with open(filename, "rt") as fp:
        data = json.load(fp)
    return data

def generator(samples, is_training, batch_size=64):
    '''
    Rather than keep all data in memory, we will make a function that keeps
    it's state and returns just the latest batch required via the yield command.

    As we load images, we can optionally augment them in some manner that doesn't
    change their underlying meaning or features. This is a combination of
    brightness, contrast, sharpness, and color PIL image filters applied with random
    settings. Optionally a shadow image may be overlayed with some random rotation and
    opacity.
    We flip each image horizontally and supply it as a another sample with the steering
    negated.
    '''
    num_samples = len(samples)

    while 1: # Loop forever so the generator never terminates
        samples = shuffle(samples)
        #divide batch_size in half, because we double each output by flipping image.
        for offset in range(0, num_samples, batch_size):
```

```
batch_samples = samples[offset:offset+batch_size]

images = []
controls = []
for fullpath in batch_samples: # not sure this is doing anything, as images are not being flipped
    try:
        frame_number = os.path.basename(fullpath).split("_")[0]
        json_filename = os.path.join(os.path.dirname(fullpath), "record_" + frame_number + ".json")
        data = load_json(json_filename)
        steering = float(data["user/angle"])
        throttle = float(data["user/throttle"])

        try:
            image = Image.open(fullpath)
        except:
            print('failed to open', fullpath)
            continue

        #PIL Image as a numpy array
        image = np.array(image, dtype=np.float32)
        # image_cp = image
        # resize for nvidia
        # nvidia 2
        # image = cv2.resize(image, (200, 66), cv2.INTER_AREA)
        # augmentation only for training
        if(conf.aug):
            if is_training and np.random.rand() < 0.6:
                image, steering = ag.augment(image, steering)

        # This provides this actual size network is expecting, so must run
        if (conf.preproc):
            image = ag.preprocess(image) # preprocess(image)
        # assert (preprocess(image)==ag.preprocess(image))

        # for nvidia2 model
        # 224 224 Alexnet
        # image = cv2.resize(image, (224, 224), cv2.INTER_AREA)
        # for NVIDIA should be 200x66
        images.append(image)

        if conf.num_outputs == 2:
            controls.append([steering, throttle])
        elif conf.num_outputs == 1:
            controls.append([steering])
        else:
            print("expected 1 or 2 outputs")

    except Exception as e:
        print(e)
        print("we threw an exception on:", fullpath)
        yield [], []


# final np array to submit to training
```

```
            X_train = np.array(images)
            y_train = np.array(controls)
            yield X_train, y_train


def get_files(filemask, s=False):
    '''
    Use a filemask and search a path recursively for matches
    Inputs
        filemask: string passed as command line option, must not be enclosed in quotes
        s: boolean, sort by create date flag
    '''

    filemask = os.path.expanduser(filemask)
    path, mask = os.path.split(filemask)

    matches = []
    for root, dirnames, filenames in os.walk(path):
        for filename in fnmatch.filter(filenames, mask):
            matches.append(os.path.join(root, filename))
    if(s == True):
        matches = sorted(matches, key=os.path.getmtime)
    return matches


def train_test_split(lines, test_perc):
    '''
    split a list into two parts, percentage of test used to seperate
    '''
    train = []
    test = []

    for line in lines:
        if random.uniform(0.0, 1.0) < test_perc:
            test.append(line)
        else:
            train.append(line)

    return train, test

def make_generators(inputs, limit=None, batch_size=conf.batch_size):
    '''
    load the job spec from the csv and create some generator for training
    '''

    #get the image/steering pairs from the csv files
    lines = get_files(inputs)
    print("found %d files" % len(lines))

    if limit is not None:
        lines = lines[:limit]
        print("limiting to %d files" % len(lines))

    train_samples, validation_samples = train_test_split(lines, test_perc=0.2)
```

```
    print("num train/val", len(train_samples), len(validation_samples))

    # compile and train the model using the generator function
    train_generator = generator(train_samples, True, batch_size=batch_size)
    validation_generator = generator(validation_samples, False, batch_size=batch_size)

    n_train = len(train_samples)
    n_val = len(validation_samples)

    return train_generator, validation_generator, n_train, n_val


def go(model_name, outdir, epochs=50, inputs='./log/*.jpg', limit=None):

    print('working on model', model_name)

    hf_mkdir(outdir)
    outdir += '/' + model_name
    hf_mkdir(outdir)

    # https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior
    dt = strftime("%Y%m%d%H%M%S")
    fp = outdir + '/' + dt + '_' + model_name;
    model_name = fp + '.h5'

    '''
    modify config.json to select the model to train.
    '''
    # model = models.get_nvidia_model_naoki(conf.num_outputs)
    # interpreter seems to be playing up, dummy assignment to appease
    if(conf.model_name=='nvidia1'):
        model = models.nvidia_model1(conf.num_outputs)
    elif(conf.model_name=='nvidia2'):
        model = models.nvidia_model2(conf.num_outputs)
    elif(conf.model_name == 'nvidia_baseline'):
        model = models.nvidia_baseline(conf.num_outputs)
    elif(conf.model_name == 'alexnet'):
        try:
            model = models.get_alexnet(conf.num_outputs)
        except Exception as e:
            print("Failed to save accuracy/loss graph: " + str(e))
        # adjust image size
        # conf.row = conf.image_width_net = conf.image_width_alexnet
        # conf.col = conf.image_height_net = conf.image_height_alexnet

    else:
        try:
            raise ValueError
        except ValueError:
            print('No valid model name given. Please check command line arguments and model.py')

    callbacks = [
        # running with naoki's model
```

```
        keras.callbacks.EarlyStopping(monitor='val_loss', patience=conf.training_patience, verbose=0),
        keras.callbacks.ModelCheckpoint(model_name, monitor='val_loss', save_best_only=True, verbose=0),
        # keras.callbacks.ModelCheckpoint(('model-{epoch:03d}' +'_' + model_name), monitor='val_loss', save_best_o
nly=True, verbose=0),
    ]

    batch_size = conf.batch_size


    #Train on session images
    train_generator, validation_generator, n_train, n_val = make_generators(inputs, limit=limit, batch_size=batch_siz
e)

    if n_train == 0:
        print('no training data found')
        return

    steps_per_epoch = n_train // batch_size
    validation_steps = n_val // batch_size

    print("steps_per_epoch", steps_per_epoch, "validation_steps", validation_steps)
    s1 = strftime("%Y%m%d%H%M%S")

    #history = model.fit_generator(train_generator,
    #    steps_per_epoch = steps_per_epoch,
    #    validation_data = validation_generator,
    #    validation_steps = validation_steps,
    #    epochs=epochs,
    #    verbose=1,
    #    callbacks=callbacks)
    history = []
    try:
        history = model.fit(train_generator,
            steps_per_epoch = steps_per_epoch,
            validation_data = validation_generator,
            validation_steps = validation_steps,
            epochs=epochs,
            verbose=1,
            callbacks=callbacks)
    except Exception as e:
        print("Failed to run model: " + str(e))

    # e =  "Input to reshape is a tensor with 147456 values, but the requested shape requires a multiple of 27456". errp
r rao with jungle1 dataset
    #   [[node model/flattened/Reshape (defined at /git/sdsandbox/src/train.py:250) ]] [Op:__inference_train_function
_2398]
    #
    # Function call stack:
    # train_function
    s2 = strftime("%Y%m%d%H%M%S")
    FMT = "%Y%m%d%H%M%S"
    tdelta = datetime.strptime(s2, FMT) - datetime.strptime(s1, FMT)
    tdelta = "Total training time: " + str(tdelta)
```

```
    # save info
    log = fp + '.log'
    logfile = open(log, 'w')
    logfile.write("Model name: " + model_name + "\r\n")
    logfile.write(tdelta)
    logfile.write('\r\n');
    logfile.write("Training loss: " + '{0:.3f}'.format(history.history['loss'][-1]) + "\r\n")
    logfile.write("Validation loss: " + '{0:.3f}'.format(history.history['val_loss'][-1]) + "\r\n")
    logfile.write("Training accuracy: " + '{0:.3f}'.format(history.history['acc'][-1]) + "\r\n")
    logfile.write("Validation accuracy: " + '{0:.3f}'.format(history.history['val_acc'][-1]) + "\r\n")
    logfile.close()


    # save history
    histfile = fp + '.history'
    with open(histfile, 'wb') as file_pi:
        pickle.dump(history.history, file_pi)
    try:
        if do_plot:
            fig = plt.figure()
            sp = '(l,vl,a,va)' + '{0:.3f}'.format(history.history['loss'][-1]) \
                + ',' + '{0:.3f}'.format(history.history['val_loss'][-1]) \
                + ',' + '{0:.3f}'.format(history.history['acc'][-1]) \
                + ',' + '{0:.3f}'.format(history.history['val_acc'][-1]) \
                + ' - ' + model_name.split('/')[-1]
            fig.suptitle(sp, fontsize=9)

            ax = fig.add_subplot(111)
            #ax.plot(time, Swdown, '-', label='Swdown')
            ax.plot(history.history['loss'], 'r-', label='Training Loss', )
            ax.plot(history.history['val_loss'], 'c-', label='Validation Loss')
            ax2 = ax.twinx()
            ax2.plot(history.history['acc'], 'm-', label='Training Accuracy')
            ax2.plot(history.history['val_acc'], 'y-', label='Validation Accuracy')
            ax.legend(loc=2) # https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.legend.html
            ax.grid()
            ax.set_xlabel("Epoch")
            ax.set_ylabel(r"Loss")
            ax2.set_ylabel(r"Accurary")
            ax.set_ylim(0, 0.2)
            ax2.set_ylim(0.5, 1)
            ax2.legend(loc=1)
            aimg = fp + '_accuracy.png'
            plt.savefig(aimg)
    except Exception as e:
        print("Failed to save accuracy/loss graph: " + str(e))

# moved to helper functions
#def parse_bool(b):
#    return b == "True"

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='train script')
    parser.add_argument('--model', type=str, help='model name')
```

```
    parser.add_argument('--outdir', type=str, help='output directory')
    parser.add_argument('--epochs', type=int, default=conf.training_default_epochs, help='number of epochs')
    parser.add_argument('--inputs', default='../dataset/unity/genRoad/*.jpg', help='input mask to gather images')
    parser.add_argument('--limit', type=int, default=None, help='max number of images to train with')
    parser.add_argument('--aug', type=parse_bool, default=False, help='image augmentation flag')
    parser.add_argument('--preproc', type=parse_bool, default=True, help='image preprocessing flag')

    args = parser.parse_args()

    conf.aug = args.aug
    conf.preproc = args.preproc
    conf.model_name = args.model
    #print(tf.__version__) 2.2.0
    ag = Augmentation.Augmentation(args.model)
    go(args.model, args.outdir, epochs=args.epochs, limit=args.limit, inputs=args.inputs)

#python train.py ..\outputs\mymodel_aug_90_x4_e200 --epochs=200
```