TECNOLÓGICO DE MONTERREY®

Maestría en Inteligencia Artificial Aplicada

Curso: Navegación autónoma

Tecnológico de Monterrey

Prof Titular y Tutor: Dr. David Antonio Torres

Prof Asistente: Maricarmen Vázquez Rojí

ALUMNO: Luis Alfonso Sabanero Esquivel

MATRICULA: A01273286

ALUMNO: Jose Mtanous

MATRICULA: A00169781

ALUMNO: Guillermo Alfonso Muñiz Hermosillo

MATRICULA: A01793101

ALUMNO: Jorge Mariles Estrada

MATRICULA: A01335663

Proyecto Final

Navegación Autónoma por Behavioral Cloning

Los videos del vehículo conducido por nuestra red neuronal se pueden encontrar aquí:

Explicación del código y proyecto

https://youtu.be/6QKBQJi1cKA

Pista1

https://www.youtube.com/watch?v=3EqZarqPXvg

Pista2

https://www.youtube.com/watch?v=RwEucryrof8

In [ ]:
```
# Importamos todas las librerías que necesitamos para el proyecto
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.optimizers import Adam
from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten, Dense
import cv2
import pandas as pd
import random
import os
import ntpath
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import matplotlib.image as mpimg
from imgaug import augmenters as iaa
import tensorflow as tf
```

In [ ]:
```
# Como el simulador corre nativo en windows pero el script de python corre sobre WS
# Usamos una librería que nos permita manejar las trayectorias de los archivos de m
# del sistema operativo.
def path_leaf(path):
    head, tail = ntpath.split(path)
    return tail
```

In [ ]:
```
# Para gestionar mejor los datos de entrenamiento, guardamos en directorios indepen
# los archivos de entrenamiento, de está manera podemos añadir o descartar corridas
# de mala calidad y solo quedarnos con los datos de calidad pristina.

# Hicimos varias vueltas de en sentido normal y varias en sentido contrario
# Además grabamos varias veces las partes más difíciles de la pista.
# En total son más de 1GB de imágenes de entrenamiento

columns = ['center', 'left', 'right', 'steering', 'throttle','reverse', 'speed']

datadirs = ['/mnt/c/Users/jmtan/OneDrive/Documentos/github/Parts/Pista2Adelante/',
            '/mnt/c/Users/jmtan/OneDrive/Documentos/github/Parts/Pista2Reversa/']

data = pd.DataFrame(columns=columns)

for trainingPath in datadirs:
    _data = pd.read_csv(os.path.join(trainingPath, 'driving_log.csv'), names = colu
```

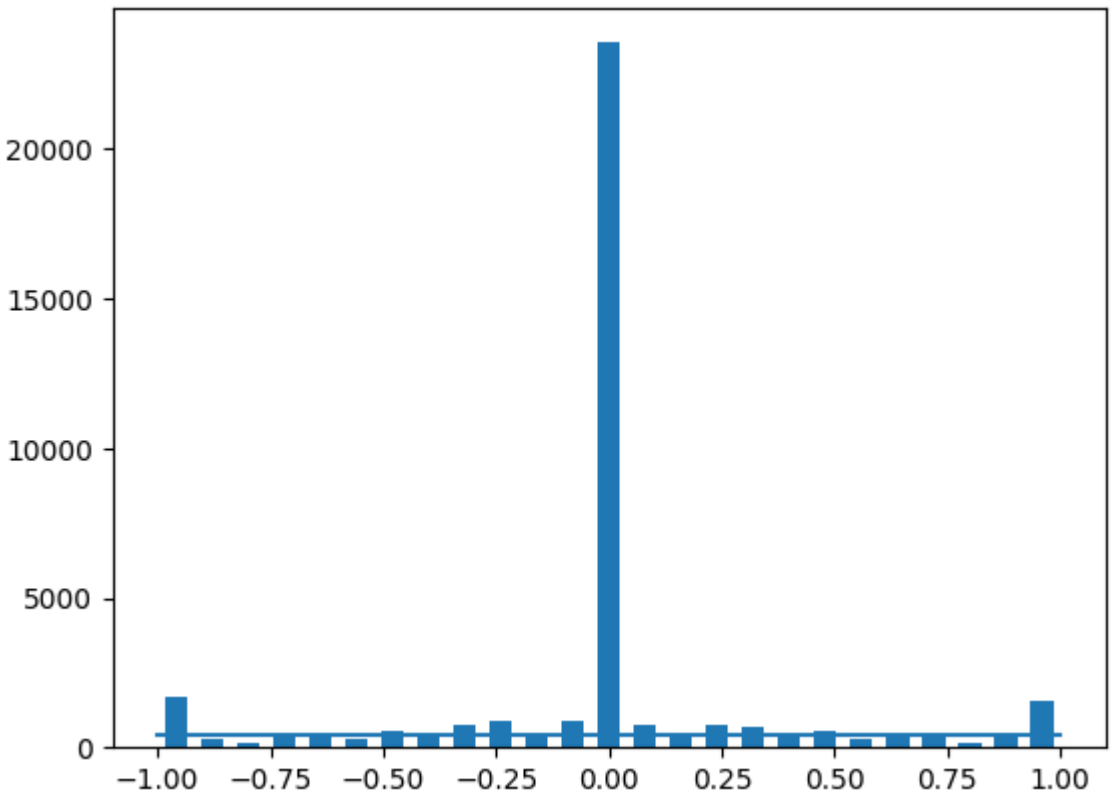
```
    _data['center'] = trainingPath + 'IMG/' + _data['center'].apply(path_leaf)
    _data['left'] = trainingPath + 'IMG/' + _data['left'].apply(path_leaf)
    _data['right'] = trainingPath + 'IMG/' + _data['right'].apply(path_leaf)
    data = pd.concat([data,_data],ignore_index=True)
data.head()
```

Out[ ]:

	center	
0	/mnt/c/Users/jmtan/OneDrive/Documentos/github/...	/mnt/c/Users/jmtan/OneDrive/Documentos/
1	/mnt/c/Users/jmtan/OneDrive/Documentos/github/...	/mnt/c/Users/jmtan/OneDrive/Documentos/
2	/mnt/c/Users/jmtan/OneDrive/Documentos/github/...	/mnt/c/Users/jmtan/OneDrive/Documentos/
3	/mnt/c/Users/jmtan/OneDrive/Documentos/github/...	/mnt/c/Users/jmtan/OneDrive/Documentos/
4	/mnt/c/Users/jmtan/OneDrive/Documentos/github/...	/mnt/c/Users/jmtan/OneDrive/Documentos/

In [ ]:
```
# Como podemos ver la mayoría de los datos son imágenes en donde el vhículo no esta
# girando e iba recto.

num_bins = 25
samples_per_bin = 400
hist, bins = np.histogram(data['steering'], num_bins)
center = (bins[:-1]+ bins[1:]) * 0.5
plt.bar(center, hist, width=0.05)
plt.plot((np.min(data['steering']), np.max(data['steering'])),(samples_per_bin, sam
```

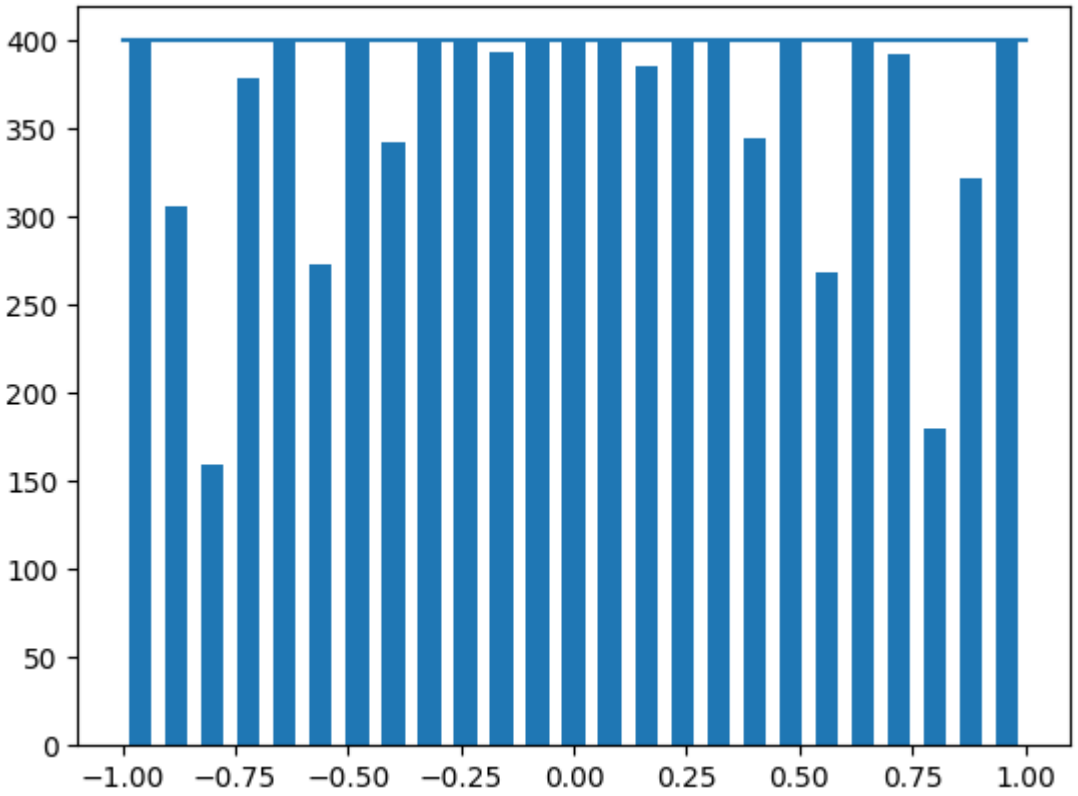
Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7f0cecbb43d0>]
```



In [ ]:
```
# Para balancear mejor el entrenamiento, descartamos la mayoría de las imágenes con

print('total data:', len(data))
remove_list = []
for j in range(num_bins):
    list_ = []
    for i in range(len(data['steering'])):
        if data['steering'][i] >= bins[j] and data['steering'][i] <= bins[j+1]:
            list_.append(i)
    list_ = shuffle(list_)
    list_ = list_[samples_per_bin:]
    remove_list.extend(list_)
print('removed:', len(remove_list))
data.drop(data.index[remove_list], inplace=True)
print('remaining:', len(data))
hist, _ = np.histogram(data['steering'], (num_bins))
plt.bar(center, hist, width=0.05)
plt.plot((np.min(data['steering']), np.max(data['steering'])),(samples_per_bin, sam
```

```
total data: 37212
removed: 28268
remaining: 8944
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7f0cec04b9d0>]
```



In [ ]:
```
indexed_data = data.iloc[1]
center, left, right = indexed_data[0], indexed_data[1],indexed_data[2]
center.strip()
```

Out[ ]:
```
'/mnt/c/Users/jmtan/OneDrive/Documentos/github/Parts/Pista1Adelante/IMG/center_202
3_06_22_12_50_48_231.jpg'
```

In [ ]:
```
print(data.iloc[1])

def load_img_steering(df):
    image_path = []
    steering = []
    for i in range(len(data)):
        indexed_data = data.iloc[i]
        center, left, right = indexed_data[0], indexed_data[1],indexed_data[2]
        image_path.append(center.strip())
        steering.append(float(indexed_data[3]))
        # left image append
        image_path.append(left.strip())
        steering.append(float(indexed_data[3])+0.15)
        # right image append
        image_path.append(right.strip())
        steering.append(float(indexed_data[3])-0.15)
    image_paths = np.asarray(image_path)
    steerings = np.asarray(steering)
    return image_paths, steerings
image_paths, steerings = load_img_steering(data)
```

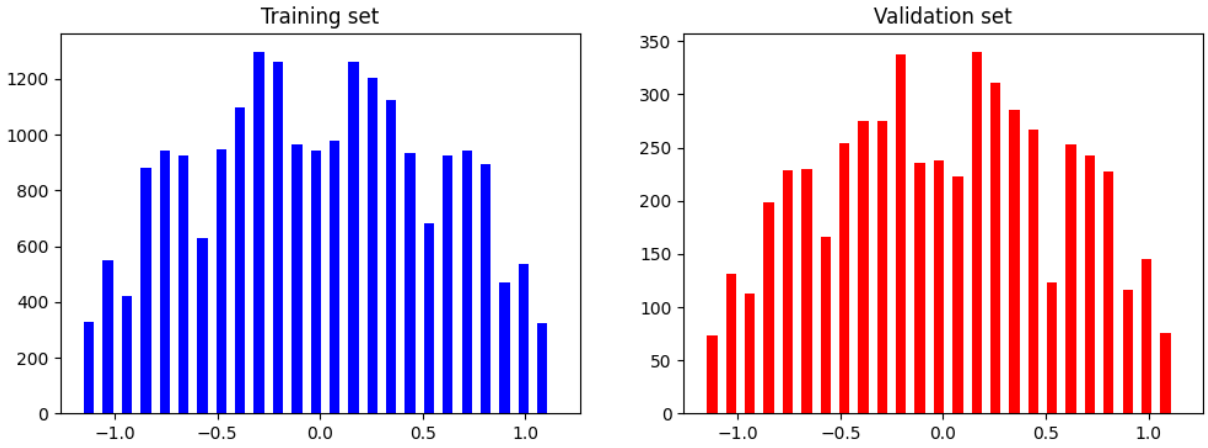
```
center      /mnt/c/Users/jmtan/OneDrive/Documentos/github/...
left        /mnt/c/Users/jmtan/OneDrive/Documentos/github/...
right       /mnt/c/Users/jmtan/OneDrive/Documentos/github/...
steering                                                 -0.2
throttle                                                  1.0
reverse                                                     0
speed                                                12.03734
Name: 21, dtype: object
```

In [ ]:
```
# Usamos el 20% de las imágenes para validación y el 80 para entrenamiento

X_train, X_valid, y_train, y_valid = train_test_split(image_paths, steerings, test_
print('Training Samples: {}\nValid Samples:{}'.format(len(X_train), len(X_valid)))
```

```
Training Samples: 21465
Valid Samples:5367
```

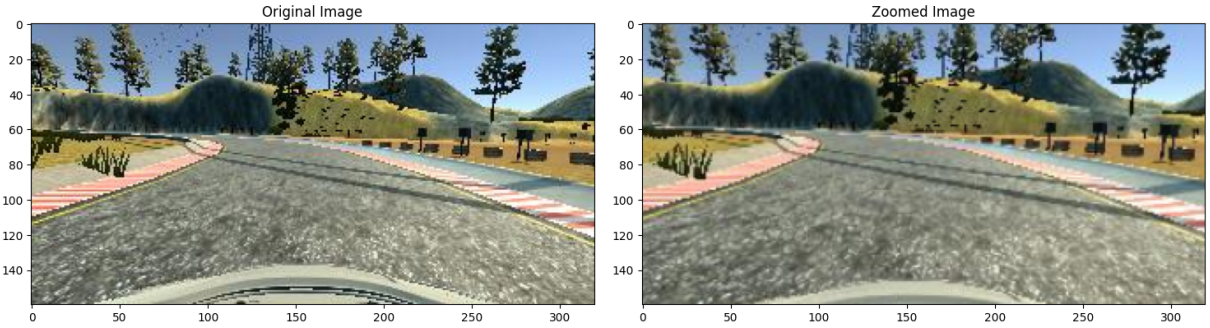
In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
axes[0].hist(y_train, bins=num_bins, width=0.05, color='blue')
axes[0].set_title('Training set')
axes[1].hist(y_valid, bins=num_bins, width=0.05, color='red')
axes[1].set_title('Validation set')
```

Out[ ]:
```
Text(0.5, 1.0, 'Validation set')
```



In [ ]:
```
# Generamos imágnes sintéticas para aumentar nuestro juego de entrenamiento
# Cambios de intensidad
# Proyección (mirror)
# Recorte

def zoom(image):
    zoom = iaa.Affine(scale=(1, 1.3))
    image = zoom.augment_image(image)
    return image

image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
zoomed_image = zoom(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(zoomed_image)
axs[1].set_title('Zoomed Image')
```

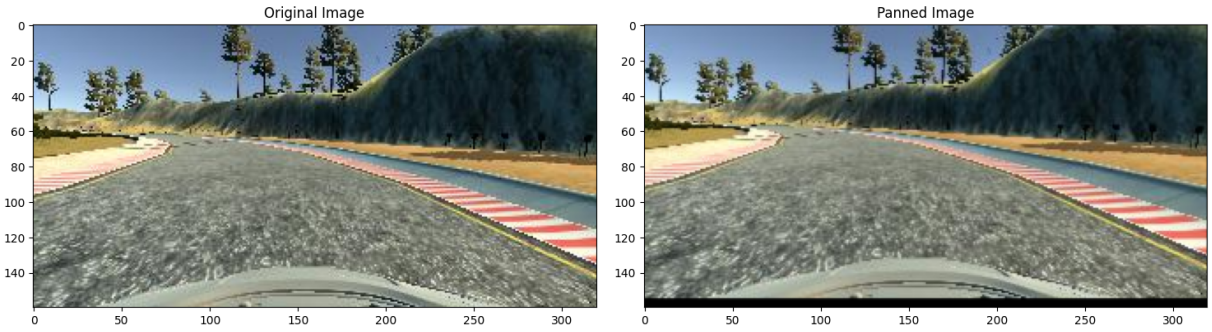
Out[ ]:
```
Text(0.5, 1.0, 'Zoomed Image')
```



In [ ]:
```
def pan(image):
    pan = iaa.Affine(translate_percent= {"x" : (-0.1, 0.1), "y":(-0.1, 0.1)})
    image = pan.augment_image(image)
    return image

image = image_paths[random.randint(0, 1000)]
```

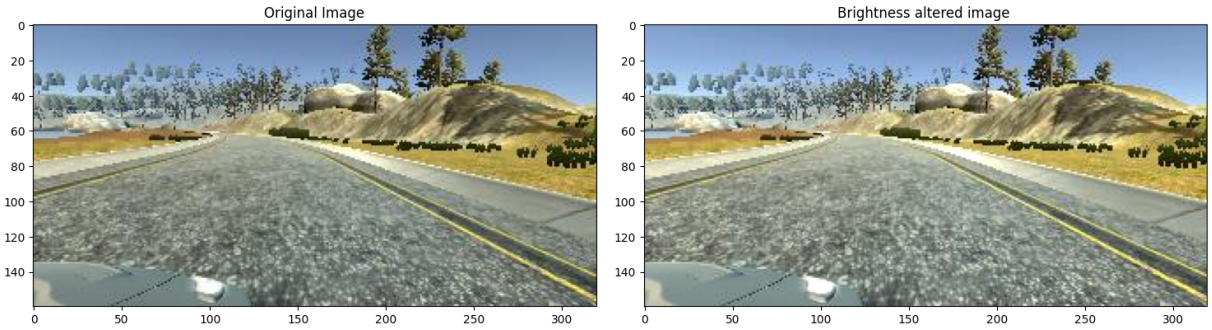
```
original_image = mpimg.imread(image)
panned_image = pan(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(panned_image)
axs[1].set_title('Panned Image')
```

Out[ ]: Text(0.5, 1.0, 'Panned Image')



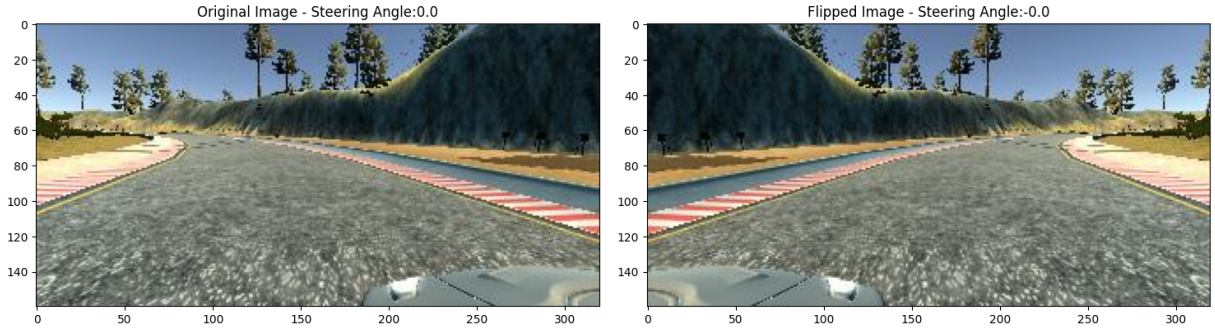
In [ ]:
```
def img_random_brightness(image):
    brightness = iaa.Multiply((0.2, 1.2))
    image = brightness.augment_image(image)
    return image

image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
brightness_altered_image = img_random_brightness(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(brightness_altered_image)
axs[1].set_title('Brightness altered image ')
```

Out[ ]: Text(0.5, 1.0, 'Brightness altered image ')



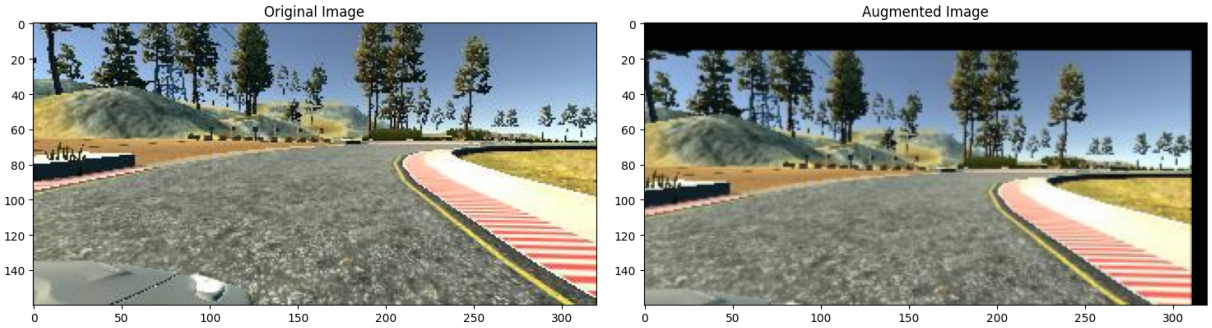
In [ ]:
```
def img_random_flip(image, steering_angle):
    image = cv2.flip(image,1)
    steering_angle = -steering_angle
    return image, steering_angle

random_index = random.randint(0, 1000)
image = image_paths[random_index]
steering_angle = steerings[random_index]

original_image = mpimg.imread(image)
flipped_image, flipped_steering_angle = img_random_flip(original_image, steering_an

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image - ' + 'Steering Angle:' + str(steering_angle))

axs[1].imshow(flipped_image)
axs[1].set_title('Flipped Image - ' + 'Steering Angle:' +   str(flipped_steering_an
```

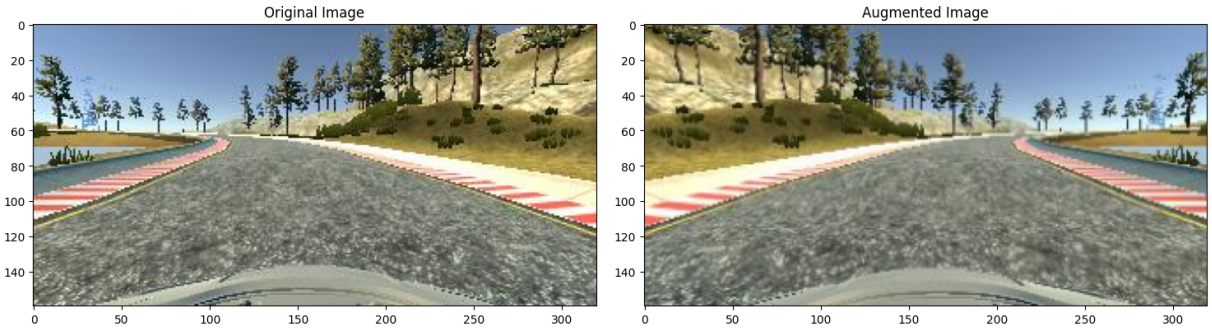
Out[ ]:
```
Text(0.5, 1.0, 'Flipped Image - Steering Angle:-0.0')
```

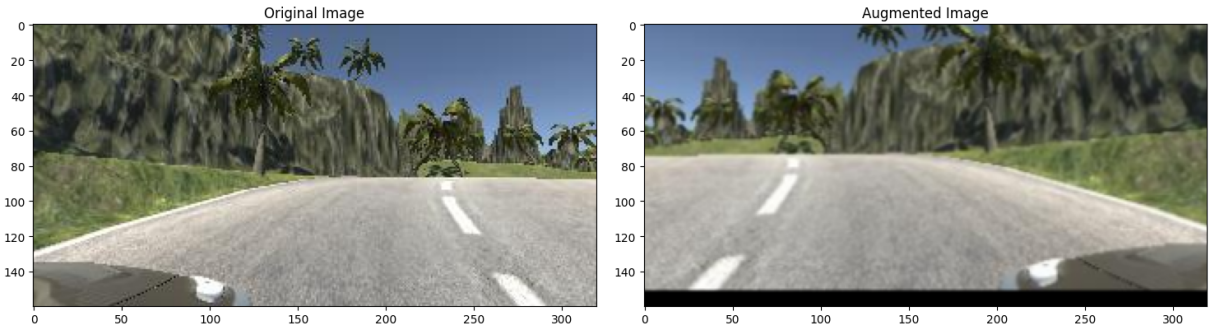


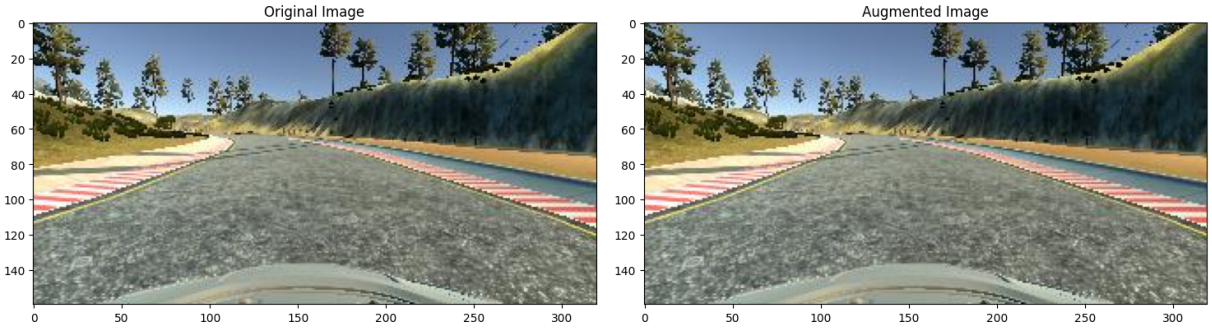
In [ ]:
```
def random_augment(image, steering_angle):
    image = mpimg.imread(image)
    if np.random.rand() < 0.5:
        image = pan(image)
    if np.random.rand() < 0.5:
        image = zoom(image)
    if np.random.rand() < 0.5:
        image = img_random_brightness(image)
    if np.random.rand() < 0.5:
        image, steering_angle = img_random_flip(image, steering_angle)
    return image, steering_angle

ncol = 2
nrow = 10

fig, axs = plt.subplots(nrow, ncol, figsize=(15, 50))
fig.tight_layout()

for i in range(10):
    randnum = random.randint(0, len(image_paths) - 1)
    random_image = image_paths[randnum]
    random_steering = steerings[randnum]
```

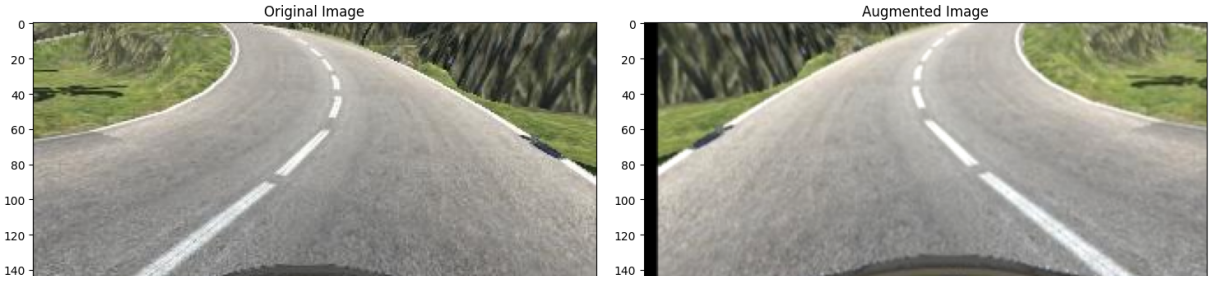
```
    original_image = mpimg.imread(random_image)
    augmented_image, steering = random_augment(random_image,random_steering)
    axs[i][0].imshow(original_image)
    axs[i][0].set_title("Original Image")
    axs[i][1].imshow(augmented_image)
    axs[i][1].set_title("Augmented Image")
```

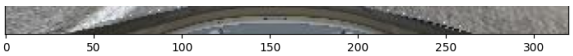
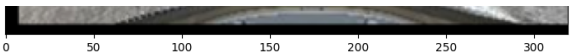

Original Image
Augmented Image


Original Image
Augmented Image


Original Image
Augmented Image


Original Image
Augmented Image


Original Image
Augmented Image

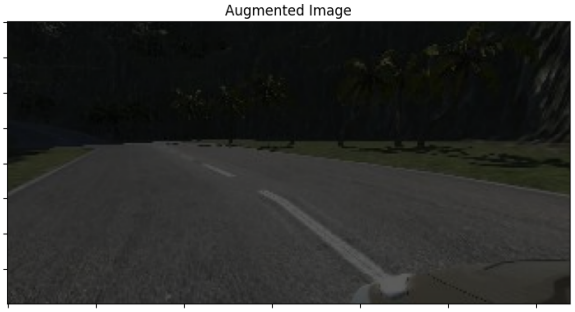



Original Image

Augmented Image

Original Image

Augmented Image

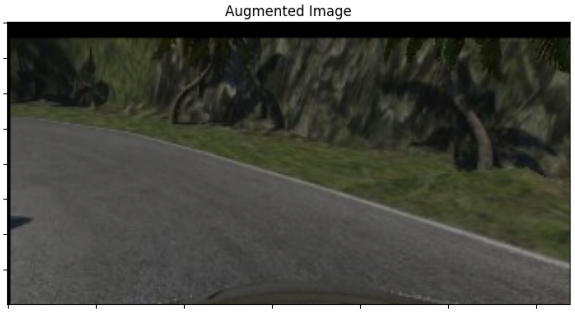


Original Image

Augmented Image

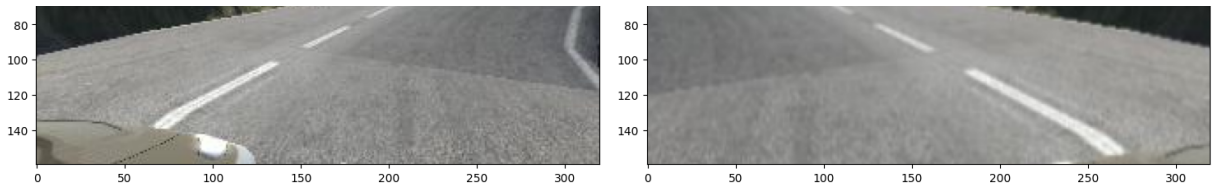
Original Image

Augmented Image



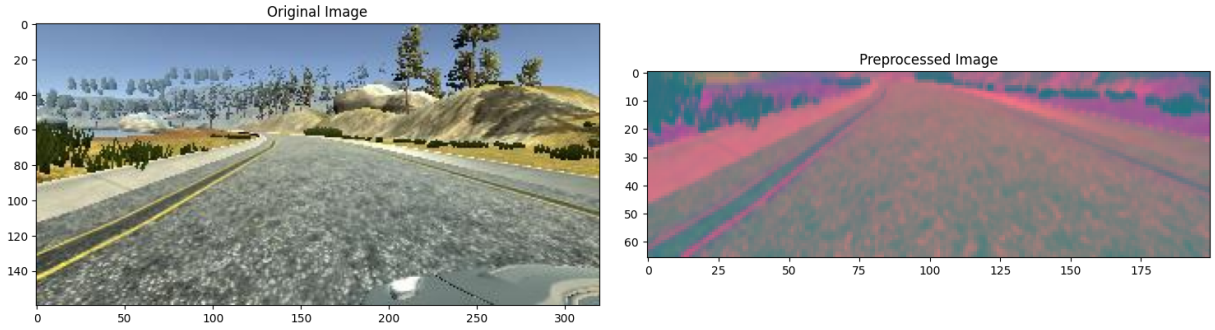
Original Image

Augmented Image



In [ ]:
```
# Cambiamos el espacio de colores a uno más robusto y recortamos la parte central d

def img_preprocess(img):
    img = img[60:135,:,:]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img, (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img/255
    return img

image = image_paths[100]
original_image = mpimg.imread(image)
preprocessed_image = img_preprocess(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()
axs[0].imshow(original_image)
axs[0].set_title('Original Image')
axs[1].imshow(preprocessed_image)
axs[1].set_title('Preprocessed Image')
```

Out[ ]:
```
Text(0.5, 1.0, 'Preprocessed Image')
```

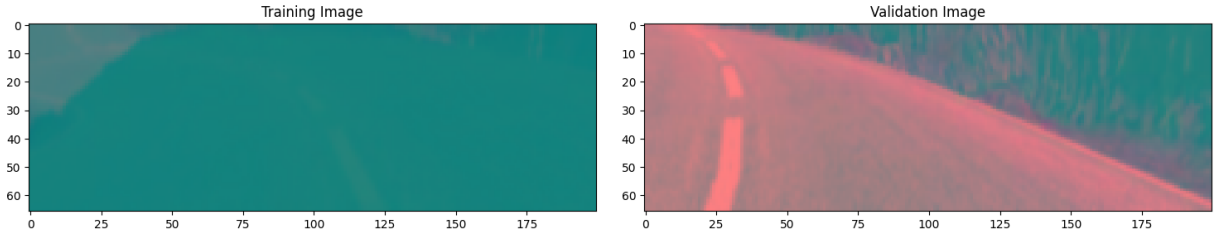


In [ ]:
```
# Función auxiliar para generar batches de imágenes para alimentar nuestro entremie

def batch_generator(image_paths, steering_ang, batch_size,istraining):
    while True:
        batch_img = []
        batch_steering = []
        for i in range(batch_size):
            random_index = random.randint(0, len(image_paths) - 1)
            if istraining:
                im, steering = random_augment(image_paths[random_index],steering_an
            else:
                im = mpimg.imread(image_paths[random_index])
                steering = steering_ang[random_index]
            im = img_preprocess(im)
            batch_img.append(im)
```

```
            batch_steering.append(steering)
        yield (np.asarray(batch_img), np.asarray(batch_steering))
```

In [ ]:
```
x_train_gen, y_train_gen = next(batch_generator(X_train, y_train,1, 1))
x_valid_gen, y_valid_gen = next(batch_generator(X_valid, y_valid,1, 0))

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(x_train_gen[0])
axs[0].set_title('Training Image')

axs[1].imshow(x_valid_gen[0])
axs[1].set_title('Validation Image')
```

Out[ ]:
```
Text(0.5, 1.0, 'Validation Image')
```



In [ ]:
```
# Definimos el modelo de envidia para entrenarlo.

def nvidia_model():
  model = Sequential()
  model.add(tf.keras.layers.Conv2D(24,(5,5), strides=(2, 2), input_shape=(66, 200,
  model.add(tf.keras.layers.Conv2D(36, (5,5), strides=(2, 2), activation='elu'))
  model.add(tf.keras.layers.Conv2D(48, (5,5), strides=(2, 2), activation='elu'))
  model.add(tf.keras.layers.Conv2D(64, (3,3), activation='elu'))

  model.add(tf.keras.layers.Conv2D(64, (3,3), activation='elu'))
#   model.add(Dropout(0.5))


  model.add(Flatten())

  model.add(Dense(100, activation = 'elu'))
#   model.add(Dropout(0.5))

  model.add(Dense(50, activation = 'elu'))
#   model.add(Dropout(0.5))

  model.add(Dense(10, activation = 'elu'))
#   model.add(Dropout(0.5))

  model.add(Dense(1))

  optimizer = Adam(lr=1e-3)
  model.compile(loss='mse', optimizer=optimizer)
  return model
```

In [ ]:
```
model = nvidia_model()
print(model.summary())
```

```
2023-06-23 13:08:09.578966: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_
executor.cc:982] could not open file to read NUMA node: /sys/bus/pci/devices/0000:0
9:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-06-23 13:08:09.770653: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_
executor.cc:982] could not open file to read NUMA node: /sys/bus/pci/devices/0000:0
9:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-06-23 13:08:09.770723: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_
executor.cc:982] could not open file to read NUMA node: /sys/bus/pci/devices/0000:0
9:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-06-23 13:08:09.772365: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_
executor.cc:982] could not open file to read NUMA node: /sys/bus/pci/devices/0000:0
9:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-06-23 13:08:09.772421: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_
executor.cc:982] could not open file to read NUMA node: /sys/bus/pci/devices/0000:0
9:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-06-23 13:08:09.772458: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_
executor.cc:982] could not open file to read NUMA node: /sys/bus/pci/devices/0000:0
9:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-06-23 13:08:11.283316: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_
executor.cc:982] could not open file to read NUMA node: /sys/bus/pci/devices/0000:0
9:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-06-23 13:08:11.283399: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_
executor.cc:982] could not open file to read NUMA node: /sys/bus/pci/devices/0000:0
9:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-06-23 13:08:11.283406: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1722]
Could not identify NUMA node of platform GPU id 0, defaulting to 0.  Your kernel may
not have been built with NUMA support.
2023-06-23 13:08:11.283459: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_
executor.cc:982] could not open file to read NUMA node: /sys/bus/pci/devices/0000:0
9:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-06-23 13:08:11.283490: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635]
Created device /job:localhost/replica:0/task:0/device:GPU:0 with 9509 MB memory:  ->
device: 0, name: NVIDIA GeForce RTX 3060, pci bus id: 0000:09:00.0, compute capabili
ty: 8.6
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 31, 98, 24)        1824

 conv2d_1 (Conv2D)           (None, 14, 47, 36)        21636

 conv2d_2 (Conv2D)           (None, 5, 22, 48)         43248

 conv2d_3 (Conv2D)           (None, 3, 20, 64)         27712

 conv2d_4 (Conv2D)           (None, 1, 18, 64)         36928

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 100)               115300

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 10)                510

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
_________________________________________________________________
None
/home/jmtanous/miniconda3/envs/tf/lib/python3.9/site-packages/keras/optimizers/legac
y/adam.py:117: UserWarning: The `lr` argument is deprecated, use `learning_rate` ins
tead.
  super().__init__(name, **kwargs)
```

In [ ]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

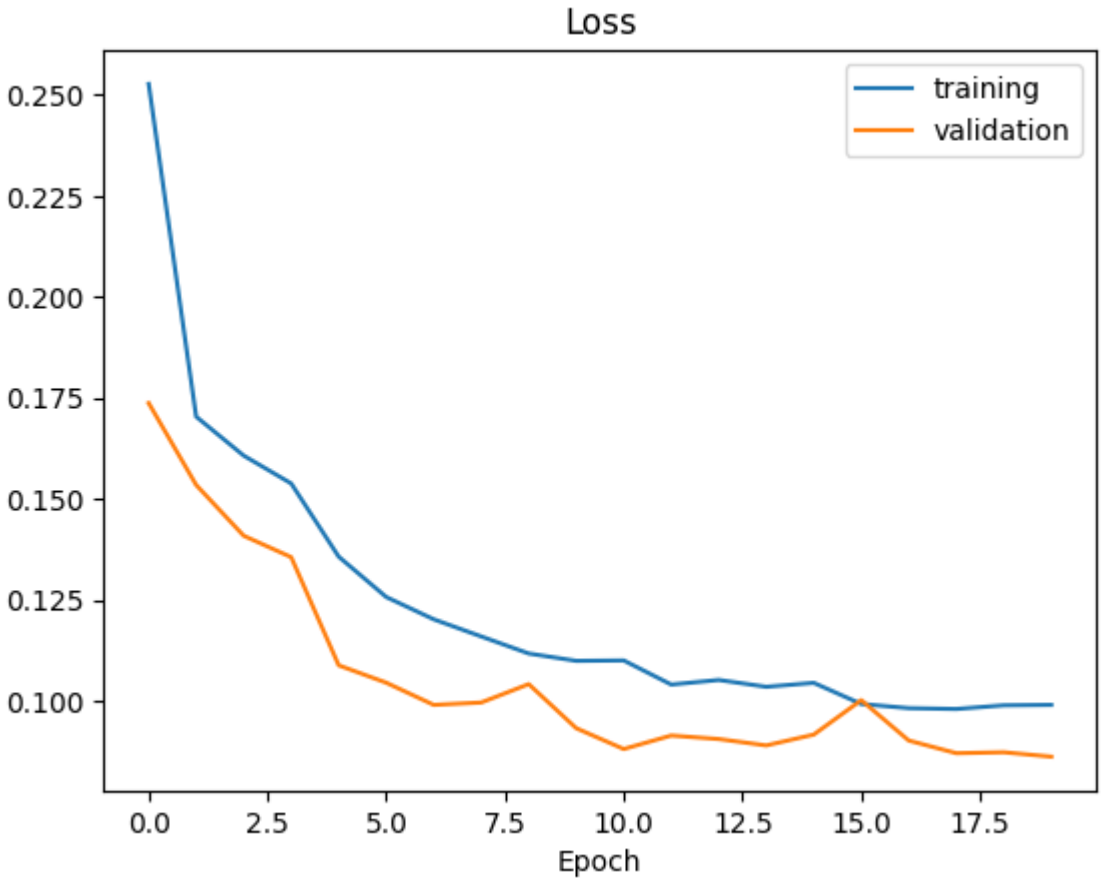
In [ ]:
```
# Entrenamos nuestro modelo en 20 épocas

history = model.fit(batch_generator(X_train, y_train,100, 1),
                    steps_per_epoch=300,
                    epochs=20,
                    validation_data=batch_generator(X_valid, y_valid, 100, 0),
                    validation_steps=200,
                    verbose=1,
                    shuffle = 1)
```

```
300/300 [==============================] - 433s 1s/step - loss: 0.0989 - val_loss:
0.0873
Epoch 20/20
300/300 [==============================] - 448s 1s/step - loss: 0.0990 - val_loss:
0.0862
```

In [ ]:
```
model.save('modelPista2Pista1AdelanteAtras20epochs.h5')
```

In [ ]:
```
# Como podemos ver el modelo no está sobre entrenado
# incluso mayor entrenamiento podría mejorar nuestro desempeño
# sin embargo el tiempo de entrenamiento es largo y con 20 épocas fue
# suficiente para que el coche manejara ambas pistas de manera autónoma.
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training', 'validation'])
plt.title('Loss')
plt.xlabel('Epoch')
```

Out[ ]:
```
Text(0.5, 0, 'Epoch')
```



El código está basado en capítulo 10 del libro de texto: Ranjan, S. y Senthamilarasu S.(2020). Applied Deep Learning and Computer Vision for Self-Driving Cars. Packt Publishing Ltd.