```
#Accessing google drive where the video file which is to be reduces is stored
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# installing and importing necessary
pip install matplotlib==2.0.2
```

```
Requirement already satisfied: matplotlib==2.0.2 in /usr/local/lib/python3.6/dis
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: pyparsing!=2.0.0,!=2.0.4,!=2.1.2,!=2.1.6,>=1.5.6
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/dist-
Requirement already satisfied: numpy>=1.7.1 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: pytz in /usr/local/lib/python3.6/dist-packages (f
```

```
import numpy as np # linear algebra
import pandas as pd # data processing

# The below two are visualization libraires
import matplotlib.pyplot as plt
import seaborn as sns

# for calculating interval
from time import time

import glob
import random

plt.rcParams['figure.figsize'] = 10,8 # setting default figure size for the kernel

# for clustering image pixels
from sklearn.cluster import KMeans
# for simple image processing tasks
from skimage import io
from tqdm import tqdm
from tensorflow.keras.preprocessing import image
#cv2 to acces the images
import cv2
from PIL import Image


#Downloading an image from the internet for the experiment
url = 'https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcTzLzdD4MoAPP3I_JjrxndA
img_original = io.imread(url)
plt.axis('off')
plt.imshow(img_original)
plt.title('Our buddy for the experiment !')
plt.show()
```

Our buddy for the experiment !

```
#Converting image into numpy array
img = np.array(img_original,dtype=float) / 255

# Save the dimensions, we will be need them later
w, h, d = original_shape = img.shape
print('Original Shape'.center(20,'='))
print(img.shape)

# image_array size - w*h , d
image_array = img.reshape(-1,d)
print('ReShaped'.center(20,'='))
print(image_array.shape)
```

```
===Original Shape===
(462, 664, 3)
======ReShaped======
(306768, 3)
```

```
#Applying K-means clustering with 64 colours and 32 colours on the image to reduce th
n_colours = [64,32]

# 64 colour image
t0 = time()
kmeans64 = KMeans(n_clusters = n_colours[0],random_state=42,verbose=2,n_jobs=-1).fit(

print('Completed 64 clusters in'+ str({round(time()-t0,2)})+'seconds.')
```

```
# 32 colour image
t0 = time()
kmeans32 = KMeans(n_clusters = n_colours[1],random_state=42,verbose=2,n_jobs=-1)
kmeans32.fit(image_array)

print('Completed 32 clusters in' + str({round(time()-t0,2)})+  ' seconds.')

labels64 = kmeans64.labels_
labels32 = kmeans32.labels_
```

```
     Completed 64 clusters in{25.45}seconds.
     Completed 32 clusters in{13.45} seconds.
```

```
print('Within cluster sum of square error for'+str( {n_colours[0]})+' clusters = '+st
print('Within cluster sum of square error for'+str( {n_colours[1]})+' clusters = '+st
```

```
     Within cluster sum of square error for{64} clusters = {139.19}
     Within cluster sum of square error for{32} clusters = {274.54}
```

```
#Recreating the images after applying K-means clustering on the image
def recreate_image(centroids, labels, w, h):
    # centroids variable are calculated from the flattened image
    # centroids: w*h, d
    # so each row depicts the values per depth
    d = centroids.shape[1]
    image = np.zeros((w, h, d))
    label_idx = 0
    for i in range(w):
        for j in range(h):
            # filling values in new image with centroid values
            image[i][j] = centroids[labels[label_idx]]
            label_idx += 1
    return image


#All the three images with title.
plt.figure(figsize=(20,10))
plt.subplot(132)
plt.axis('off')
plt.title('Original image')
plt.imshow(img)

plt.subplot(131)
plt.axis('off')
plt.title('Compressed image (64 colors, K-Means)')
plt.imshow(recreate_image(kmeans64.cluster_centers_, labels64, w, h))

plt.subplot(133)
plt.axis('off')
plt.title('Compressed image (32 colors, K-Means)')
plt.imshow(recreate_image(kmeans32.cluster_centers_, labels32, w, h))
```

```
plt.show()
```

Dividing the video into frames to apply K-means clustering on each frame which will result in a reduced size of the video

```
# Program To Read video and Extract Frames
import cv2

# Function to extract frames
def FrameCapture(path):

    # Path to video file
    vidObj = cv2.VideoCapture(path)

    # Used as counter variable
    count = 0

    # checks whether frames were extracted
    success = 1

    while success:

        # vidObj object calls read
        # function extract frames
        success, image = vidObj.read()

        # Saves the frames with frame-count
        cv2.imwrite("/content/drive/MyDrive/Video_frames/frames/frame%d.jpg" % count,

        count += 1


# Calling the function
FrameCapture('/content/drive/MyDrive/Video_frames/PEOPLE_ARE_AWESOME_2014__BEST_VIDE
```

```
FrameCapture('/content/drive/MyDrive/Video_frames/PEOPLE ARE AWESOME 2014 _ BEST VIDE


#Importing all the frames of the video
images = glob.glob('/content/drive/MyDrive/Video_frames/frames/*')
len(images)
```

```
4604
```

```
images = []
for i in range(0,4604):
  images.append('/content/drive/MyDrive/Video_frames/frames/frame'+str(i)+'.jpg')
len(images)
```

```
4604
```

```
# Preprocessing the frames/Images to apply K-Means clustering on each frame/Image
img_width = 350
img_height = 350

X = []

for img_name in tqdm(images):
    # reading the image
    img = image.load_img(img_name,target_size=(img_width, img_height))
    img = image.img_to_array(img)
    img = img/255.0
    w, h, d = original_shape = img.shape
    img = img.reshape(-1,d)
    # appending the image into the list
    X.append(img)
# converting the list to numpy array
Xtrain = np.array(X)

print(Xtrain.shape)     # Xtrain is the dataset with frames
```

```
100%|██████████| 4604/4604 [01:26<00:00, 53.14it/s]
(4604, 122500, 3)
```

```
# Applying K-means clustering with 32 colour on each frame in Xtrain dataset and savi
count = 1332
for i in tqdm(range(0,len(Xtrain))):
  kmeans32 = KMeans(n_clusters = 32,random_state=42,verbose=1,n_jobs=-1)
  kmeans32.fit(Xtrain[i])
  labels32 = kmeans32.labels_
  plt.axis('off')
  plt.imsave('/content/drive/MyDrive/Video_frames/Compressed_imagesframes/frame%d.jpg
  plt.close()
  count +=1
```

```
100%|██████████| 3272/3272 [4:58:59<00:00,  5.48s/it]
```

```
#Importinf the compressed Images/frames in the workspace to recombine
compressedImages = []
for i in range(0,4604):
  compressedImages.append('/content/drive/MyDrive/Video frames/Compressed imagesframe
print(len(compressedImages))    # 'CompressedImages' is the list of all the compresse
```

```
4604
```

```
#Getting the height and width of the video which based on the compressed frames/Image
mean_height = 0
mean_width = 0

num_of_images = len(compressedImages)
# print(num_of_images)

for file in compressedImages:
    im = Image.open(file)
    width, height = im.size
    mean_width += width
    mean_height += height
    # im.show()   # uncomment this for displaying the image

# Finding the mean height and width of all images.
# This is required because the video frame needs
# to be set with same width and height. Otherwise
# images not equal to that width height will not get
# embedded into the video
mean_width = int(mean_width / num_of_images)
mean_height = int(mean_height / num_of_images)

print(mean_height)  # The height of the video
print(mean_width)   # The width of the video
```

```
350
350
```

```
# Resizing of the images/frames to give them same width and height
for file in tqdm(compressedImages):
    if file.endswith(".jpg") or file.endswith(".jpeg") or file.endswith("png"):
        # opening image using PIL Image
        im = Image.open(file)

        # im.size includes the height and width of image
        width, height = im.size

        # resizing
        imResize = im.resize((mean_width, mean_height), Image.ANTIALIAS)
        imResize.save( file, 'JPEG', quality = 95) # setting quality
```

```
# Video Generating function
def generate_video():

    video_name = 'PeopleAreAmazing.mp4' # Name of the compressed video

    frame = cv2.imread(compressedImages[0])

    height, width, layers = frame.shape

    video = cv2.VideoWriter(video_name, 0, 30, (width, height))  # writing the video

    # Appending the images to the video one by one
    for image in compressedImages:
        video.write(cv2.imread(image))

    # Deallocating memories taken for window creation
    cv2.destroyAllWindows()
    video.release()  # releasing the video generated


# Calling the generate_video function to generate the compressed video
generate_video()
```