**PROCEDURE**

1. I have built this model based on the concept of deep convolutional embedded clustering(DCEC) in which the features of the images are filtered using deep convolutional neural networks and the filtered data is passed through the clustering algorithm.
2. The first step towards building the model was to load the dataset which consists of images into numpy arrays in order to process it.
3. The step step was to preprocess it. In which the given dataset was divided into training ,testing and validation data.next each image is divided by 255 to speed up the training process.
4. The preprocessed images are then sent to Autoencoders to extract the features of images in order for classification of images and then restoring the original structure of the image.
5. Autoencoders are convolutional neural networks which consist of two parts encoders and decoders . The encoder is part of the neural network which filters the features of the image ,the decoder is the mirror of the encoder which uses the same features filtered by the encoder to reconstruct the image .
6. The next step is to cluster the unlabelled data of images this is achieved with the help of kmeans algorithm in which the data is labeled to a cluster with a series of iterations until each data is labeled to its respective cluster .
7. The features of the image which are extracted through autoencoder is used by kmeans algorithm to cluster images.
8. Then a confusion matrix is created to match the labels by the kmeans algorithm with the true labels of the image i.e the testing data.
9. The input image is given which is processed through the encoder and its cluster is predicted by the kmeans algorithm.
10. The cluster number to which the input image belongs is returned and is used to plot the images belonging to the respective cluster.

#importing libraries

%tensorflow\_version 2.x

import numpy as np

import pandas as pd

import os

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

import keras

from keras.models import Sequential

from keras.layers import Dense, Dropout, Conv2D, MaxPool2D, UpSampling2D, Activation

from keras import backend as K

from sklearn.cluster import KMeans

from sklearn.metrics import confusion\_matrix

import seaborn as sns

from keras.datasets import mnist

import zipfile

import tensorflow as tf

import glob

from skimage.io import imread

from skimage.transform import resize

local\_zip = '/tmp/dataset.zip' #loading data set

zip\_ref = zipfile.ZipFile(local\_zip,'r')

zip\_ref.extractall('/tmp')

zip\_ref.close()

img\_list = sorted(glob.glob('/tmp/dataset/\*.jpg')) #converting images to numpy arrays

IMG\_SIZE = 152

x\_data=np.empty((len(img\_list),IMG\_SIZE,IMG\_SIZE,3),dtype=np.float32)

for i,img\_path in enumerate(img\_list):

img = imread(img\_path)

img = resize(img, output\_shape=(IMG\_SIZE,IMG\_SIZE,3),preserve\_range=True)

x\_data[i] = img

X\_train, X\_test = train\_test\_split(x\_data,test\_size=0.2, random\_state=123) #splitting the data to training,testing and validation

X\_train, X\_validate= train\_test\_split(X\_train, test\_size=0.25, random\_state=123)

X\_test=X\_test/255 #preprocessing the data

X\_train=X\_train/255

X\_validate=X\_validate/255

#using the autoencoder which consist of the first part as encoder and the second part as the decoder

#encoders are used to extract the features of the images as they are passed down from one layer to another

#and the features are extracted from the embedded layer i.e 7th layer.

model = Sequential()

model.add(Conv2D(8, kernel\_size=3, padding='same', activation='relu', input\_shape=(152,152,3)))

model.add(MaxPool2D((2,2), padding='same'))

model.add(Dropout(0.2))

model.add(Conv2D(8, kernel\_size=3, padding='same', activation='relu'))

model.add(MaxPool2D((2,2), padding='same'))

model.add(Dropout(0.2))

model.add(Conv2D(8, kernel\_size=3, padding='same', activation='relu'))

model.add(MaxPool2D((2,2), padding='same'))#embedded layer

model.add(Dropout(0.2))#decoder starts from here

model.add(Conv2D(8, kernel\_size=3, padding='same', activation='relu'))

model.add(UpSampling2D((2,2)))

model.add(Dropout(0.2))

model.add(Conv2D(8, kernel\_size=3, padding='same', activation='relu'))

model.add(UpSampling2D((2,2)))

model.add(Dropout(0.2))

model.add(Conv2D(8, kernel\_size=3, padding='same', activation='relu'))

model.add(UpSampling2D((2,2)))

model.add(Dropout(0.2))

model.add(Conv2D(1, kernel\_size=3, padding='same', activation='relu'))

model.summary()#summary of the model

#compiling the model using adam optimizer with a learning rate=0.001 and the loss function is mean squared error which is optimal for classifying.

model.compile(optimizer='adam', loss="mse",metrics=['accuracy'])

#training the model

model.fit(X\_train, X\_train, epochs=20, batch\_size=40, validation\_data=(X\_validate, X\_validate), verbose=1)

#restoring the original image in order to maintain the structure of the images.

restored\_testing\_dataset = model.predict(X\_test)

#extracting the features from embedded layer

encoder = K.function([model.layers[0].input], [model.layers[8].output])

#encoding the x\_test images in order to be trained in kmeans algorithm

encoded\_images = encoder([X\_test])[0].reshape(-1,19\*19\*8)

#training the extracted features of image in the kmeans algorithm

kmeans = KMeans(n\_clusters=7,max\_iter=2000)

clustered\_training\_set = kmeans.fit\_predict(encoded\_images)

#plotting the confusion matrix to identify the number of images classified as correct

cm = confusion\_matrix(clustered\_training\_set, clustered\_training\_set)

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

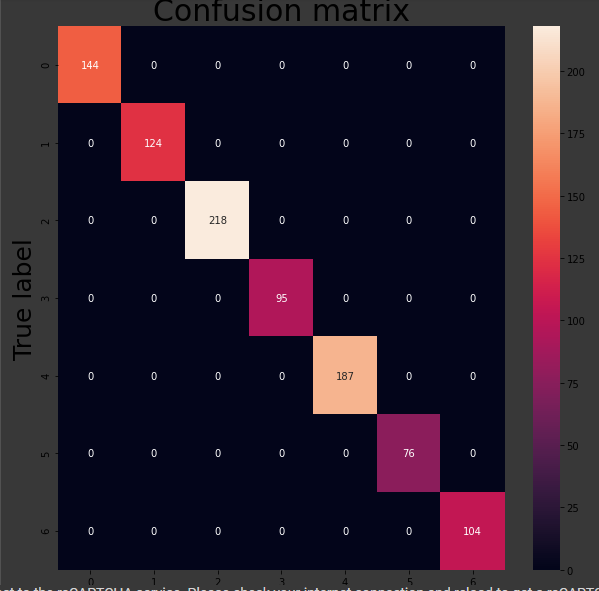
sns.heatmap(cm, annot=True, fmt="d")

plt.title("Confusion matrix", fontsize=30)

plt.ylabel('True label', fontsize=25)

plt.xlabel('Clustering label', fontsize=25)

plt.show()



#importing the image as input in order to identify the images belonging to the same cluster as outout

from google.colab import files

from keras.preprocessing import image

uploaded=files.upload()

for fn in uploaded.keys():

# predicting images

path='/content/' + fn

img=image.load\_img(path, target\_size=(152, 152,3))

x=image.img\_to\_array(img)

images = np.vstack([x])

images=images.reshape(-1,152,152,3)/255

enco = encoder([images])[0].reshape(-1,19\*19\*8)

r=kmeans.predict(enco)

#plotting the images belonging to the same cluster as input

fig = plt.figure(figsize=(20,20))

cluster = cm[r].argmax()

for c, val in enumerate(X\_test[clustered\_training\_set == cluster][0:10]):

fig.add\_subplot(10, 10,10+c+1)

plt.imshow(val.reshape((152,152,3)))

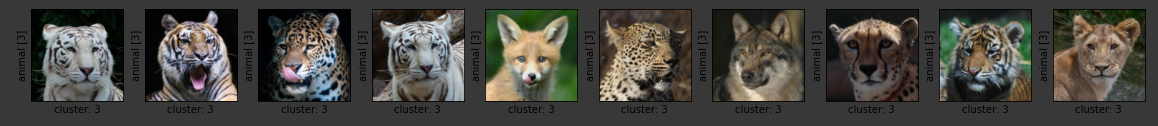
plt.gray()

plt.xticks([])

plt.yticks([])

plt.xlabel('cluster: '+str(cluster))

plt.ylabel('animal '+str(r))



Thank you.