COMPRESSED IMAGE QUALITY ASSESSMENT BASED ON SAAK FEATURES

Project Term Paper

by
Sakshi Sharma

JACOBS
UNIVERSITY

Prof. Sergey Kosov
Jacobs University Bremen

1 Introduction

The objective of this project is to implement an algorithm in python to extract SAAK (Subspace approximation with augmented kernels) features which is used in measuring the quality of the compressed image . For this purpose I have referred to paper written by Xinfeng Zhang, Sam Kwong and C.-C. Jay Kuo. I have implemented the method of extracting the SAAK feautures presented in the paper. But i was not able to implement the compresses image quality assessment method being used in the refereed paper but computed the MSE between the original image and reconstructed image.Further, the MSE is used to find the PSNR (Peak signal to noise ratio) for measuring the quality of compressed image. Since this project was meant to completed in group of 3 so I did not get enough time to implement exact quality assessment method presented in the paper. This project is being presented as final exam for masters in data engineering under the course Data Visualization and Image Processing. The remainder of this paper is organized as follows. Section 2 introduces the data-driven SAAK transform.

2 SAAK TRANSFORM

Similar to the Convolution Neural Network(CNN), SAAK tranform is used to extract the features in order to better understand the image. The Saak transform consists of two main element similar with traditional CNNs, i.e., subspace approximation and kernel augmentation. Different from CNN, subspace approximation in SAAK transform utilizes the orthonormal eigenvectors of the covariance matrix of training samples as transform kernels, which are the well-known Karhunen-Loe‘ve transform (KLT) kernels. In the case of considering discrete data, the KLT method is similar to the PCA method as it needs to find the covariance matrix and find the set of orthonormal eigenvectors that maximize the variance of our data along them. Therefore, for the subsbace apprroximation and building the KLT kerrnels I implement the PCA method from sklearn package in python . Following paragraphs explains the python functions I built for extracting the SAAK features. These functions written in python can be be found in the appendix.

The first function I built, *extract_patch_data* is first used to extract a numpy array of data patches(size 2*2) for each image.The input for this function should be a numpy array of shape (n,d,h,h) where n is any number of image, d is the numbers of channels in the image and h is the height and width of the squared image. The output of this function is then taken by the second function, *pca_aug* and outputs an array of KLT kernels. The function takes the numpy array of the data patches and first reshapes it into a simpler data-matrix format. Then for this data cloud mean is computed and substracted. This is equivalent to the common step we do to normalize data before using the PCA method. PCA is done on the normalized data and we retain all PCA components. These PCA components are then used to create the augmented KLT kernels. First we make a matrix with all the negative values for the computed PCA component vectors and then, stack these over the positive component matrix to make the positive negative kernel pair. This arrray of the KLT kernels next need to be reshaped to use for convolution which is done by the third function $conv\_filt$. The output of the $conv\_filt$

function which gives us the KLT kernels, is then passed to the last function *conv_relu* along with the np array of the original images. Both the passed array in the function are first converted into torch data types and subsequently to py-torch Tensors. Converting them to tensors allows us to easily find the convolution between the filters and images stored in the tensors using the neural network functionalities of py-torch and also pass them through the relu function of the same package. The Output of this function is two Tensors, first which contains the relu output(i.e the SAAK coefficients that represent the dominant structures of the images. The second tensor contains all the filters which we will need in order to reconstruct the image from the SAAK coefficients. All the steps outlined in this paragraph, only represents a single stage SAAK Transformation. The function *single_saak_trans* wraps all the smaller functions discussed above to preform a single stage SAAK transformation. To implement a multistage SAAK transform the wrapper function *single_saak_trans* can be called in an iterative loop where we iterate over the desired number of stages.

3 Image compression based on SAAK Features

As mentioned in the introduction for image quality assessment I computed the MSE between the original image and reconstructed image.Further, the MSE is used to find the PSNR (Peak signal to noise ratio) for measuring the quality of compressed image. I first extract a subset of images from the MINST dataset. For theses extracted dataset I compute the SAAK coefficient by implementing a 2 stage SAAK Transformation. Further, I used Saak coefficient found after the second stage to reconstruct the image. Then, compute PSNR as mentioned above. The computed PSNR values can be found in the appendix with the codes and output. The below image show the psnr values calculated for 1000 image taken from minst dataset.

Figure 1: PSNR values for 1000 random images

From the above image it can be seen that most of the images have low psnr value which shows some loss of information.

4 Additional Information

To access this algorithm ,you can download the Jupiter notebook from GitHub link. https://github.com/Sakshi9147/Data-visualisation-exam

The only requirement to test new images is to have input image structured as Numpy array as discussed in section-2.

5 Appendix: Code and Output

```
import torch
import tensorflow_datasets as tfds
import tensorflow as tf
import argparse
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import scipy.stats as stats
import numpy as np
import torch.utils.data as data_utils
from sklearn.decomposition import PCA
import torch.nn.functional as F
from torch.autograd import Variable
from itertools import product
from sklearn.decomposition import PCA
import seaborn as sns
```

```
def extract_patch_data(datasets,depth):
    #make index for the data patches
    factor=np.power(2,depth)
    length=datasets.shape[2]/factor
    idx1=range(0,int(length),2)
    idx2=[i+2 for i in idx1]
    #iterate and generate list over the indexes
    data_=[datasets[:,:,i:j,k:l] for ((i,j),(k,l)) in product(zip(idx1,idx2),\
                                                              zip(idx1,idx2))]
    data_=np.array(data_)
    data=np.reshape(data_,(data_.shape[0]*data_.shape[1],data_.shape[2],2,2))
    return data
```

```
def pca_aug(data_in):
    # data reshape for PCA
    data=np.reshape(data_in,(data_in.shape[0],-1))
    # mean removal
    mean = np.mean(data, axis=0)
    datas_mean_remov = data - mean
    # PCA, retain all components
    pca=PCA(n_components=2)
    pca.fit(datas_mean_remov)
    comps=pca.components_
    # augment
    comps_aug=[vec*(-1) for vec in comps[:-1]]
    comps_complete=np.vstack((comps,comps_aug))
    return comps_complete

def conv_filt(aug_filt_array):
    shape=aug_filt_array.shape[1]/4
    num=aug_filt_array.shape[0]
    filt=np.reshape(aug_filt_array,(num,int(shape),4))
    filters=np.reshape(filt,(num,int(shape),2,2))
    return filters
```

```
def conv_relu(filters,datasets,stride=2):
    # torch data change
    filters_t=torch.from_numpy(filters)
    datasets_t=torch.from_numpy(datasets)

    # Variables
    filt=Variable(filters_t).type(torch.FloatTensor)
    data=Variable(datasets_t).type(torch.FloatTensor)

    # Convolution
    output=F.conv2d(data,filt,stride=stride)

    # Relu
    relu_output=F.relu(output)

    return relu_output,filt


def single_saak_trans(datasets=None,depth=0):




    data_flatten = extract_patch_data(datasets,depth)

    #PCA and augmentation
    comps_complete = pca_aug(data_flatten)


    # get filters
    filters = conv_filt(comps_complete)


    # output Saak coefficients and filters
    relu_output,filt = conv_relu(filters,datasets,stride=2)

    data = relu_output.data.numpy()


    print('saak transformation Complete!{}')

    return data,filt,relu_output


ds_train = tfds.as_numpy(tfds.load('mnist', split='train', batch_size=-1,\
                                   as_supervised=False))

'''EXTRRACT DATA AS NUMPY'''

def extract_dataset():

    ds_t = ds_train['image'][:1000]
    ds_t = ds_t.reshape(ds_t.shape[0],1,28,28)
    return ds_t
```

```
Downloading and preparing dataset mnist/3.0.1 (download: 11.06 MiB, generated:
WARNING:absl:Dataset mnist is hosted on GCS. It will automatically be downloac
local data directory. If you'd instead prefer to read directly from our public
GCS bucket (recommended if you're running on GCP), you can instead pass
`try_gcs=True` to `tfds.load` or set `data_dir=gs://tfds-data/datasets`.

DI Completed...: 100%                    4/4 [00:01<00:00, 3.59 file/s]


Dataset mnist downloaded and prepared to /root/tensorflow_datasets/mnist/3.0.1
```

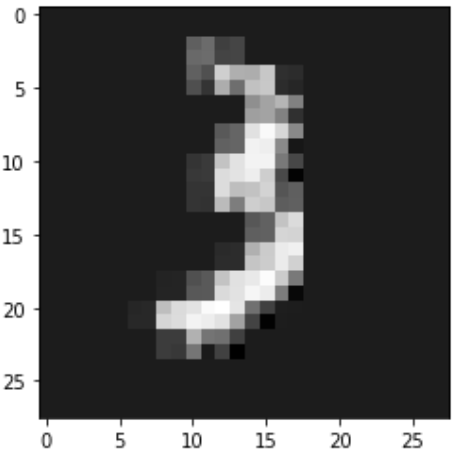
```
def milti_saak_(num_stages):
  filters = []
  outputs = []

  data = extract_dataset()
  dataset = data
  for i in num_stages:
      print ('{} stage of saak transform: '.format(i))
      data,filt,output=single_saak_trans(data,depth=i)
      filters.append(filt)
      outputs.append(output)
      print ('')


  return dataset,filters,outputs


dataset,filters,outputs=milti_saak_([1,2])
```

```
1 stage of saak transform:
saak transformation Complete!{}

2 stage of saak transform:
saak transformation Complete!{}
```

I write the function toy_recon for reconstructing the image from the SAAK Features, got from the last stage in a multistage SAAK transform. I first take reversed arrray of the output and filter of the particular stage of SAAK transform and do a transposed convolution (Deconvolution) to construct the reconstructed image.

```
def toy_recon(outputs,filters):
    # reverse the output start reconstruction from second last stage

    outputs=outputs[::-1][1:]
    filters=filters[::-1][1:]
    num=len(outputs)
    data=outputs[0]
    for i in range(num):
        data = F.conv_transpose2d(data, filters[i], stride=2)

    return data
```

```
# Making numpy array of reconstructed images
data_rec = toy_recon(outputs,filters)



# function to display image from numpy array of reconstructed image

def show_reconstructed(inv,n):
    inv_img=inv.data.numpy()[n][0]
    plt.imshow(inv_img)
    plt.gray()
    plt.show()


# show actual:
def show_actual(n):
    inv_img= np.squeeze(ds_train['image'][n], axis=-1)
    plt.imshow(inv_img)
    plt.gray()

    plt.show()


show_reconstructed(data_rec,55)
```




```
show_actual(55)
```

0 [redacted]

```
#calculating PSNR for 1000 random images in mnist dataset
import math
def PSNR():
  psnr = []
  for i in range(len(ds_train['image'][:1000])):
    m = np.mean((data_rec.data.numpy()[i] - np.squeeze(ds_train['image'][i], axis=-
    psnr.append(20 * math.log10(225 / math.sqrt(m)))

  return psnr



psnr_list = PSNR()


psnr_list
```

```
     22.206834290171585,
     19.453751068966543,
     17.571203765758003,
     18.08319646115231,
     21.361239113015714,
     17.306203971033625,
     20.066336238429543,
     17.7610205303216,
     20.100498000147997,
     18.48403468477762,
     20.926136340510368,
     18.93224467110943,
     17.11299338393953,
     18.513117183202624,
     18.444420203328608,
     20.232802749224884,
     16.731529918167052,
     17.490500441621094,
     18.030201420084587,
     19.652522003295555,
     18.204057134620292,
     18.53293921180684,
     22.0204600083696,
     16.910218439789155,
     19.088101404092548,
     18.812589200073937,
     21.788249451714307,
     19.0227260082959,
     18.402313764076762,
     17.48885518863629,
     18.00190238270432,
     17.389849721290325,
     17.709831104073082,
     17.30261713056028,
     18.396495861507777,
     16.702094145814392,
     18.067233829757885,
     18.97510893897227,
     17.272425284945978,
     21.395775757065735,
     18.713110793882826,
```

```
18.866805820054495,
17.820948100398596,
17.177047559373307,
18.904910197126426,
17.36868220831533,
18.57948334172429,
20.1923612882919,
19.88209098915456,
17.66744341372743,
18.3324438416044,
19.391959046848136,
17.84444320862383,
18.413738609915008,
21.36567269580987,
19.443381785017245,
18.323670885358347,
17.316829584087717,
18.4921714043872,
18.082618284742654,
```

```
ax = sns.boxplot(psnr_list)
ax.set(xlabel='PSNR score')
ax.set_title('PSNR score for 1000 images from minst dataset')
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarnir
  FutureWarning
Text(0.5, 1.0, 'PSNR score for 1000 images from minst dataset')
```