Hieroglyphics

Data Science 440
User Manual

Team Six

Juliana(Jiayin) Hu, Sydney Wehn, Kangdong Yuan, Xueqing Zhang, Yuqi Gao

Table of Contents:

Clustering with Feature Descriptor

As our system provides the user to use a clustering algorithm with the feature descriptors to cluster their images, we spent many times developing the clustering algorithm and testing the algorithms on multiple datasets.

First, we developed two clustering algorithms with three different feature descriptors combined in order to cluster images. The three feature descriptors are Hog, Canny and Embedding. The two clustering algorithms are k-means and agglomerative clustering. These are the foundation of the system function. The first dataset we used was taken from the following GitHub about Egypatian Hieroglyphics.
https://github.com/yedkk/ds440-r/tree/main/DS440/Dataset/Manual/Preprocessed

As we worked through this dataset, we tested different numbers of clusterings and found that for both clustering algorithms, 30 is the best number of clusters. We had done 27 as one of clustering test numbers, and since it is closest to the 30 and equal to the number of alphabet of Egyptian Hieroglyphics we went further to investigate these specific clusters for both k means and agglomerative clusters. We manually calculated the percentage of the top 1 and top 2 appeared images in each of the 27 clusters. As the result, we found that more than half of the cluster had 80% of top 1 and a much smaller percentage for top 2, we think it means the cluster successfully clustered each image to its correct labels. By reviewing our results we found that most clusters include symbols that look similar. So we think our clustering models are performing well. Thus, we decide to use these two clustering algorithms and fit them to our next data, which relates to yoga poses. However, The results of clustering for yoga dataset aren't really well. We think it is better to use classification models for yoga dataset rather than using clustering models.

Clustering with Feature Descriptor Appendix

To smoothly run the clustering models, below are the packages we used.

```
#encoding=utf-8
import numpy as np
import cv2
from skimage import feature as ft
import matplotlib.pyplot as plt
import torchvision.models as models
import pretrainedmodels
import pretrainedmodels.utils as utils
import torch
import os, sys
import json
from sklearn.decomposition import PCA
```

K-Means Clustering and Visualization (contributed by Kangdong Yuan)

This K-Means clustering and Visualization use python's jupyter notebook as the environment for running the program. The python version requires a 3.8 version or below. In order to run this jupyter notebook, users need to install keras, PIL, sklearn, numpy, matplotlib, random, pandas, pickle, torch, plotly libraries.

For more convenient use and faster operation, it is recommended that users use Google's colab service to run this program.

I have stored the main parameters of the function in the form of variables. If the user needs to input different data, the user can change the location of the input and output. In addition, users can freely adjust the number of clusters of k-means. And the user can adjust the parameter of the PCA function to change the dimension of the data.

In general, if users use smaller matrix dimensions, the training speed of k-means will become faster. However, a smaller matrix dimension will cause the instability of the k-means model, so I suggest users maintain the matrix dimension between 10-20.

If the user wants to see the error value of each cluster number, I define a code block to draw a line graph of the error value of each cluster number. The user only needs to specify the range of the number of clusters they want to see.

For the visualization of k-means clustering, I defined two code blocks to execute. The user only needs to execute the code after specifying a specific number of clusters. Two-dimensional images and three-dimensional interactive images will be automatically generated.

ResNet34 and AlexNet Classification (contributed by Sydney)

For these classification models, the environment that we will code in will be Google Colab Pro with upgraded GPU (https://colab.research.google.com/). This environment was chosen for this aspect of the project due to its high computation speed and its ease of loading in data from Google Drive with local session folders and directories.

To be able to run the models used for classification, the following packages also must be installed and imported.

```
from __future__ import print_function, division
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import numpy as np
import time
import os
import copy
from keras.preprocessing import image
import os, shutil, glob, os.path
```

Once the packages are all imported, it is time to import the data. The dataset that we used in this project was taken from the following GitHub. (https://github.com/yedkk/ds440-r/tree/main/DS440/Dataset) I then took the dataset and manually dragged the images into their associated directories that indicated their Gardiner Sign, denoted in the title of the images. After the directories were set up and the images were loaded into Google Drive, I wrote code in Google Colabs that can be found here. https://colab.research.google.com/drive/1jqP8RmVEeXl0ATujQFmOWH8JQOImE5w7?usp=sharing.

I then transformed the data with random crops and flips to be able to then create data loaders that ensured our data were in tensors and a format that could be utilized by pretrained models. Then I loaded the AlexNet and ResNet34 pretrained models and indicated the number of classes that we will be classifying based on, which in this case was 27. These classes are taken from the Gardiner Sign list as seen in the appendix. They represent the 27 letters in the Egyptian hieroglyphic alphabet that all represent different types of symbols and types of categories. Then once we correctly set up the data pipeline architecture, with layers outlined in the appendix, the two models are evaluated with loss functions and train/ validation losses and accuracies that gave us insight into how our models perform with different parameters and train/ validation/ test splitting techniques. Here, we also evaluate our model with confusion matrices to examine recall, accuracy, and precision.

Cartouche (Contributed by Yuqi)

For the whole program, to make sure you had a working environment, you need to first make sure that you installed PyMuPDF, imageAI, Pillow, Tensorflow, matplotlib

Using:
pip install -r requirements.txt

And inside the text document defiance the version of them:
PyMuPDF==1.18.10
imageai==2.1.6
Pillow==7.0.0
tensorflow==2.4.0

You also need these to run the program:
import os
import ew
import shutil
import fitz
from PIL import Image
import tensorflow (as tf) *you can define it with anything you like
from imageai.Detection.Custom import CustomObjectDetection
from imageai.Detection.Custom import DetectionModelTrainer

Of course, if you already had the dataset of pictures, then you don't need to scrape from any PDFs, remember to change the path of the input images etc.

Also to access to the tool LabelImg, it is an open sourced tool published on Github at:
https://github.com/tzutalin/labelImg
If you want to label out your own dataset to train the model.

Also as a note, be careful with the path of the image in the labelled .xml files, if the path is wrong, the program will run into error and skip the file since it is nonsense junk.

For the pre-trained model used in the program, it is an open-sourced model published on the Github at:
https://github.com/OlafenwaMoses/ImageAI/releases/tag/essential-v4