ASL Sign Language Translation

Beard, Bennion, Carlson, Napolitano

Data Science Capstone Project
Data Acquisition and Pre-Processing Report

Date:

[4/14/2022]

Team Members:

Name: Tyler Beard

Name: Adam Bennion

Name: Zach Carlson

Name: Andrew Napolitano

Identifying Data

Data Sources:

All 3 datasets contained incorrect sign language. Relying on in-house image generation.

- https://www.kaggle.com/datasets/danrasband/asl-alphabet-test
  - This data source has 30 images for each letter that are 200x200, total 870.
- https://www.kaggle.com/datasets/grassknoted/asl-alphabet
  - This data source has 3000 images for each letter that is 200x200.
- https://www.kaggle.com/datasets/datamunge/sign-language-mnist
  - 34627 grayscale 28x28 images.
  - Lacks J,Z,delete,space,nothing classes seen in the other kaggle datasets

Acquisition Process:

Search engine results for sign language alphabet was low quality. Too many of the results were drawings instead of photos, even with some filtering of results. Initially our team planned to utilize the above datasets. Closer examination revealed incorrect sign language was used, spurring our own sample generation via video recording and frame splitting. We produced our own video frame splitting script for this phase (see appendix: *video splitting file*). All images split from video were initially 1080x1920 pixels, portrait orientation..

Issues:

Incorrect datasets! Upon further inspection, the majority of our original resources had incorrect signs for the letter T or even other letters as well. The inaccuracies rendered the datasets useless and our own image generation became the primary data source. We chose not to augment the existing datasets with our own samples since the background, lighting, etc. would be too much of a difference for the selected classes, skewing training features for those classes.

If further acquisition is required, duplicating similar settings is necessary to retain a level of uniformity. Aspects to consider include resolution, background, where the hand lies in the image, distance from camera and camera aperture / zoom. Any future images split from video must be initially 1080x1920 pixels or at least the same aspect ratio, portrait orientation. The acquisition phase can be repeated quickly with our proven acquisition pipeline.

Data-Processing:

File Size Reduction

Initially the image sizes were far too large. At 1080x1920 pixels, the whole dataset was over 7.5 Gigabytes. Images were scaled down to 135x240 pixels using the Johnson-Lindenstrauss Lemma -- which calculates information loss given feature reduction. This down-scaling drastically reduced future processing requirements while retaining ~99% of sample feature distance. For more information on Johnson-Lindenstrauss Lemma, see the following link:

https://cs.stanford.edu/people/mmahoney/cs369m/Lectures/lecture1.pdf

The script produced by our team for down-scaling is available in the appendix (see appendix: *image scaling file*).

Making Samples Uniform

To make samples more uniform, grayscaling was applied via code to all down-scaled images. The idea is that if this preprocessing is applied to all samples including future test or real-world samples, then the information will be more homogeneous and reduced in dimension. Again, utilizing our own code we were able to automate grayscaling (see appendix: *conversion to grayscale file*).

Storing in a Consolidated, Machine Learning-friendly Format

Lastly, images were converted to rows in a CSV file, with each column of a row representing a pixel value from 0 to 255 due to grayscale. This format allows for consolidation and ease of use when pulling in data to our EDA and eventually our training programs. Automation was performed by our team (see appendix: *CSV conversion file*)

Potential Issues

If testing proves the provided samples were insufficient in variety prior to grayscaling, we will generate more variety. To support a wider range of skin color, image augmentation can be performed in the pre-processing phase to produce more images. Augmentation may include random noise, contrast, flipping images, brightening or darkening images, etc.

Appendix

Code involved in acquisition / pre-processing:

Video splitting file:

```
import cv2

import os
```

```
import sys


letters = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y']


for x in letters:


    current_letter = x


    dir_name = os.path.dirname(__file__)


    filename = os.path.join(dir_name, 'Videos', current_letter + '.mp4')


        frame_location  =  os.path.join(dir_name,  'framed_videos',  'letter_'  +
current_letter)
    CHECK_FOLDER = os.path.isdir(frame_location)


    # If folder doesn't exist, then create it.
    if not CHECK_FOLDER:
        os.makedirs(frame_location)


    capture = cv2.VideoCapture(filename)


    frameNr = 0
```

```
    while (True):


        success, frame = capture.read()


        if success:

            cv2.imwrite(f'{frame_location}/{current_letter}_frame_{frameNr}.jpg',
frame)


        else:

            break


        frameNr = frameNr + 1

        print(frameNr)


    capture.release()
```

Image scaling file:

```
#reference:
https://stackoverflow.com/questions/273946/how-do-i-resize-an-image-using-pil-and
-maintain-its-aspect-ratio


import os

import sys

import PIL.Image as Image
```

```
def CropImages(uncroppedDir="", croppedDir="",dimension = (135,240)):


    if not os.path.exists(uncroppedDir):
        print('No uncropped directory found.')
        return
    os.makedirs(os.path.dirname(croppedDir), exist_ok=True)


    numUncropped = len(os.listdir(uncroppedDir))
    if numUncropped < 1:
        print('No images in uncropped directory.')
        return
    for i,infile in enumerate(os.listdir(uncroppedDir)):
        print(int(i/numUncropped*100), '%', end='\r')
        outfile = croppedDir + infile
        try:
            im = Image.open(uncroppedDir+infile)
            width, height = im.size
            if dimension[0] == dimension[1]:
                min_dim = min(width,height)
                                box_dim  =  [width/2-min_dim/2,  height/2-min_dim/2,
width/2+min_dim/2, height/2+min_dim/2]
                im = im.resize(dimension, Image.ANTIALIAS,box=box_dim)
            else:
                im = im.resize(dimension, Image.ANTIALIAS)
            im.save(outfile, "png")
```

```
        except IOError as e:
            print(e,"ERROR - failed to crop '%s'" % infile)
    print("100%")


def LaunchCrop(subdirectories = True, inpath = "uncropped/",outpath = "cropped/", dimensionX = 135, dimensionY = 240):
    print("Cropping...")
    size = dimensionX, dimensionY

    if subdirectories:
        dirs = os.listdir(inpath)
        for d in dirs:
            print("Cropping files in directory:", inpath+d+'/')
            CropImages(inpath+d+'/', outpath+d+'/', size)
    else:
        CropImages(inpath, outpath, size)



args = sys.argv
if len(args) < 2:
    LaunchCrop(subdirectories = True, inpath = "uncropped/",outpath = "cropped/", dimensionX = 135, dimensionY = 240)
elif len(args) < 6:
      print("Not enough arguments. Please provide: 1)subdirectories (boolean), 2)inpath (relative to working directory, ending in /), 3)outpath (ending in /), 4)dimensionX (integer), 5)dimensionY (integer)")
```

```
else:
    LaunchCrop(subdirectories = True if args[1]=="True" else False,
                    inpath = args[2], outpath = args[3], dimensionX = int(args[4]),
dimensionY = int(args[5]))
```

Conversion to grayscale file:

```
#reference: https://bit.ly/3vku3bt


import os
import sys
from PIL import Image, ImageOps


def ConvertImagesToGrayscale(colorDir="", grayDir=""):
    if not os.path.exists(colorDir):
        print('No RBG image directory found.')
        return
    os.makedirs(os.path.dirname(grayDir), exist_ok=True)

    numColored = len(os.listdir(colorDir))
```

```
    if numColored < 1:
        print('No images in cropped_frames (i.e. RBG image) directory.')
        return

    for i,infile in enumerate(os.listdir(colorDir)):
        print(int(i/numColored*100), '%', end='\r')
        outfile = grayDir + infile

        try:
            im = Image.open(colorDir+infile)
            gray_im = ImageOps.grayscale(im)
            gray_im.save(outfile, "png")

        except IOError as e:
            print(e,"ERROR - failed to convert '%s'" % infile)
    print("100%")

def LaunchConvert(subdirectories = True, inpath = "cropped_frames/",outpath =
"gray_frames/"):
    print("Converting...")

    if subdirectories:
        dirs = os.listdir(inpath)
        for d in dirs:
            print("Converting image files in directory:", inpath+d+'/')
```

```
            ConvertImagesToGrayscale(inpath+d+'/', outpath+d+'/')

    else:

        ConvertImagesToGrayscale(inpath, outpath)



args = sys.argv

if len(args) < 2:

     LaunchConvert(subdirectories = True, inpath = "cropped_frames/",outpath =
"gray_frames/")

elif len(args) < 4:

     print("Not enough arguments. Please provide: 1)subdirectories (boolean),
2)inpath (relative to working directory, ending in /), 3)outpath (ending in /)")

else:

    LaunchConvert(subdirectories = True if args[1]=="True" else False,

                inpath = args[2], outpath = args[3])
```

Sample images from our data set. After splitting video, scaling down, and converting to grayscale. Prior to CSV conversion.

CSV conversion file:

```
import os

import sys

from PIL import Image, ImageOps

import numpy as np

import csv




def ConvertImagesToCSV(imageDir="", csvDir=""):

    if not os.path.exists(imageDir):

        print('No image directory found.')

        return

    os.makedirs(os.path.dirname(csvDir), exist_ok=True)


    numImg = len(os.listdir(imageDir))

    if numImg < 1:

        print('No images in cropped_frames (i.e. RBG image) directory.')

        return


    for i, infile in enumerate(os.listdir(imageDir)):


        try:

            img_file = Image.open(imageDir + infile)


            #get the letter
```

```
            letter = infile[0]
            print(letter)

                                        value    =    np.asarray(img_file.getdata(),
dtype=int).reshape((img_file.size[1], img_file.size[0]))
            value = value.flatten()
            value = np.concatenate([[letter], value])
            print(value)
            with open(csvDir + letter + "_img_pixels.csv", 'a') as f:
                writer = csv.writer(f)
                writer.writerow(value)
        except IOError as e:
            print(e, "ERROR - failed to convert '%s'" % infile)

    print("100%")


def           LaunchConvert(subdirectories=True,           inpath="gray_frames/",
outpath="CSV_Files/"):
    print("Converting...")

    if subdirectories:
        dirs = os.listdir(inpath)
        for d in dirs:
            print("Converting image files in directory:", inpath + d + '/')
```

```
            ConvertImagesToCSV(inpath + d + '/', outpath + d + '/')
    else:
        ConvertImagesToCSV(inpath, outpath)



args = sys.argv
if len(args) < 2:
                    LaunchConvert(subdirectories=True,     inpath="gray_frames/",
outpath="CSV_Files/")
elif len(args) < 4:
    print(
            "Not enough arguments. Please provide: 1)subdirectories (boolean),
2)inpath (relative to working directory, ending in /), 3)outpath (ending in /)")
else:
    LaunchConvert(subdirectories=True if args[1] == "True" else False,
                inpath=args[2], outpath=args[3])
```

Table of Contributions

The table below identifies contributors to various sections of this document.

	Section	Writing	Editing
1	**Data Sources**	**AB**	
2	**Data Pre-Processing**	**AB**	
3	**Appendix**	**AB**	

Grading

The grade is given on the basis of quality, clarity, presentation, completeness, and writing of each section in the report. This is the grade of the group. Individual grades will be assigned at the end of the term when peer reviews are collected.