WEEK 1 REPORT ON

MACHINE LEARNING FINAL PROJECT

at

Cambodia Academy Digital of Technology

Submitted By:

RAY CHANN UDAM

SIENG SATYA

THA RITHYVUTH

YI CHANDARA

Submit to: ***Dr. EN SOVANN***

Date: 13th November 2023

Bachelor of Computer Science

CADT

School of Computer Science

Cambodia Academy of Digital Technology

Table of Contents

1. Set Up a Group Work Environment

- Download the LFW dataset from the provided link and extract the data.
- Set up a GitHub repository and invite all the members.

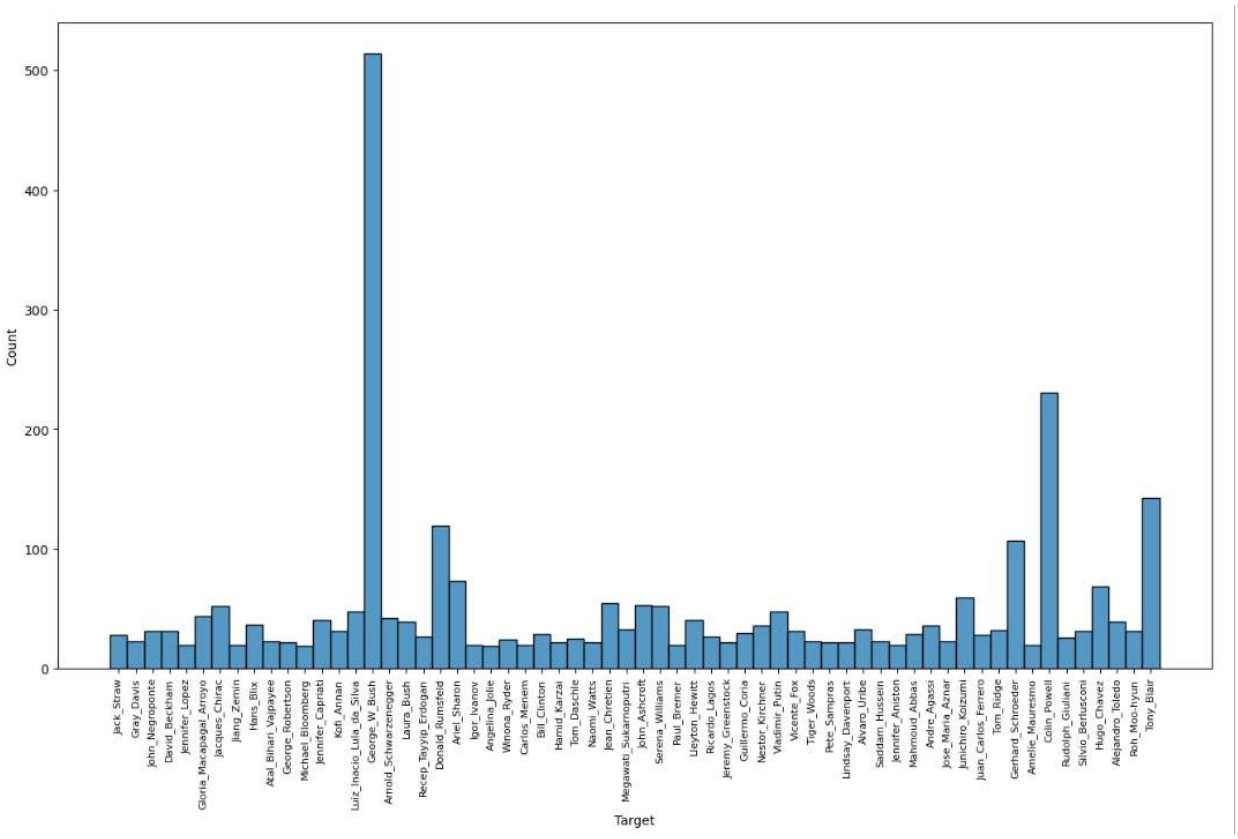
2. Divide Tasks Responsibility

We divide tasks for each member so that the one responsible for the specific task can do the research on their task, make decisions for their responsible task, and explain the reason why they do that in their task or why do did their task this way.

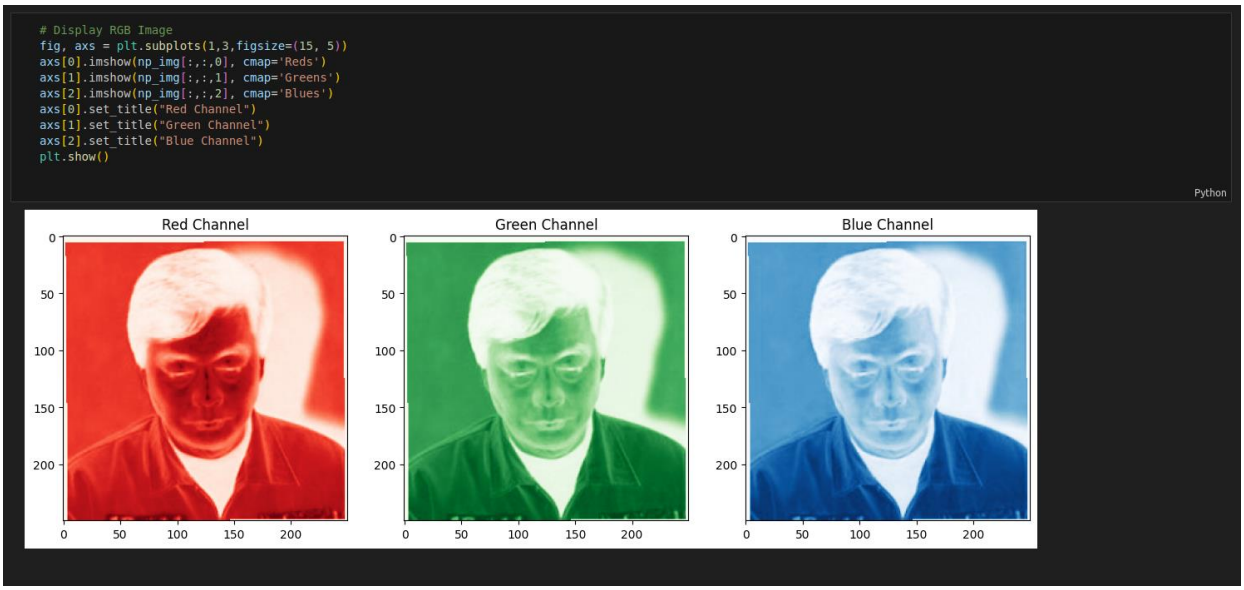
No.	Task	Responsibility
1	Data Collection: Download and preprocess the LFW dataset.	YI CHANDARA
2	Feature Extraction: Extract relevant features from facial images.	YI CHANDARA
3	Model Selection: Implement SVM or other classical ML algorithms for face recognition.	THA RITHYVUTH
4	Model Training: Train the chosen model on the dataset.	THA RITHYVUTH
5	Evaluation: Assess the performance of the model using appropriate metrics.	RAY CHANNUDAM
6	Deployment: Implement a simple interface for testing the face recognition system.	SIENG SATYA

3. Explore Data

- Folder with 1 image: 4069
- Folder with more than 1 and less than 10 images: 1522
- Folder with more than 10 images: 158
- Total JPG files: 13233



- Display Image by color channels (RGB)

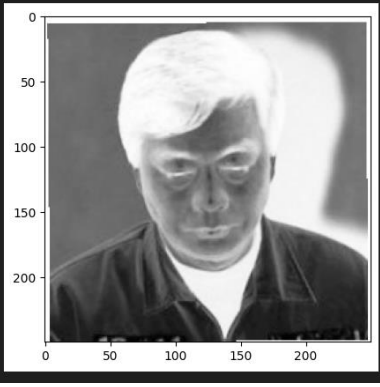


Because after we look at the shape of an image it gives us as 250px * 250px with **3-Color channel (RGB)**. For real use case when we supply model training, we would prefer **grayscale format** instead, to reduce the computational complexity by reducing the color channel.

```
#Convert Image to gray_scale
cv2_img_gray = cv2.cvtColor(plt_img, cv2.COLOR_RGB2GRAY)
plt.imshow(cv2_img_gray, cmap="Greys")
```

Python

```
<matplotlib.image.AxesImage at 0x7eb936438520>
```



4. Preprocess the Images

- Showing 300 samples of image checking to see what our data looks like or if there are unusual images.

Showing sample of images checking to see if there are unusual image

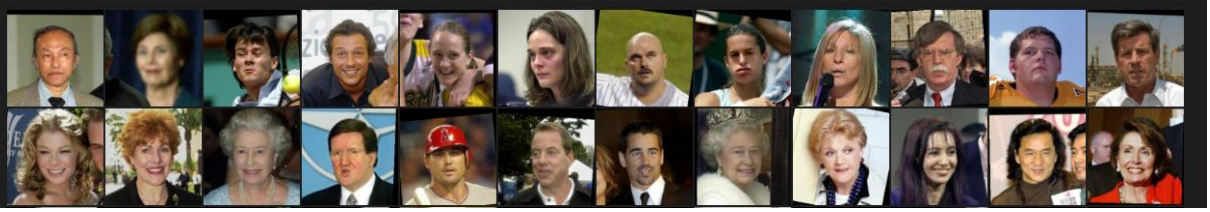
```
from easyimages import EasyImageList
easy_list = EasyImageList.from_multilevel_folder(dataset_path)
easy_list
```

[2] Python

```
<ImageList with 13233 EasyImages [filter OFF]>
```

```
easy_list.html(sample=300, size=100, by_class=False)
```

[3] Python



- Read images from a dataset, extract features, and store information about each person along with their corresponding images.

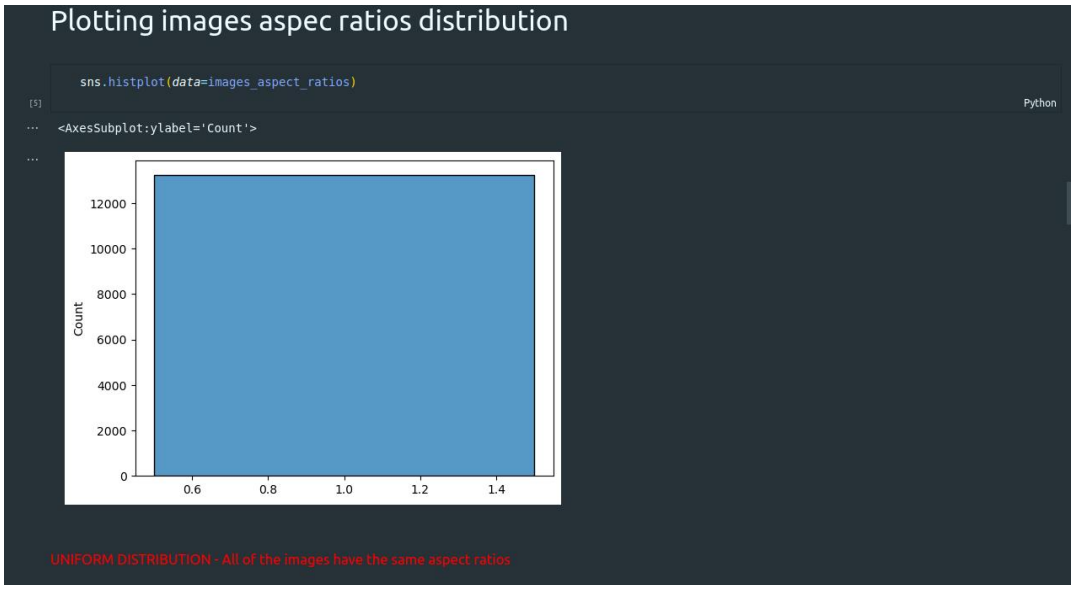
```python
# Choosing Image Resize + Image Cropping + Grayscale
images = []
targets = []
unique = 0
images_aspect_ratios = []

for dir_name in dir_names:
    try:
        file_names = os.listdir(base_image_path + dir_name)
        if(len(file_names) >= 20):
            unique = unique + 1
            for file_name in file_names:
                img = plt.imread(f"{base_image_path}{dir_name}/{file_name}")
                try:
                    img = img[30:, 30:]
                    img = cv2.resize(img, None, fx=0.75, fy=0.75)
                    images_aspect_ratios.append(img.shape[1] / img.shape[0])
                    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
                    img = img.flatten()
                    images.append(img)
                    targets.append(dir_name)
                except:
                    continue

    except:
        continue
images = np.asarray(images)
targets = np.asarray(targets)
print(f"Number of identities: {unique}")
```

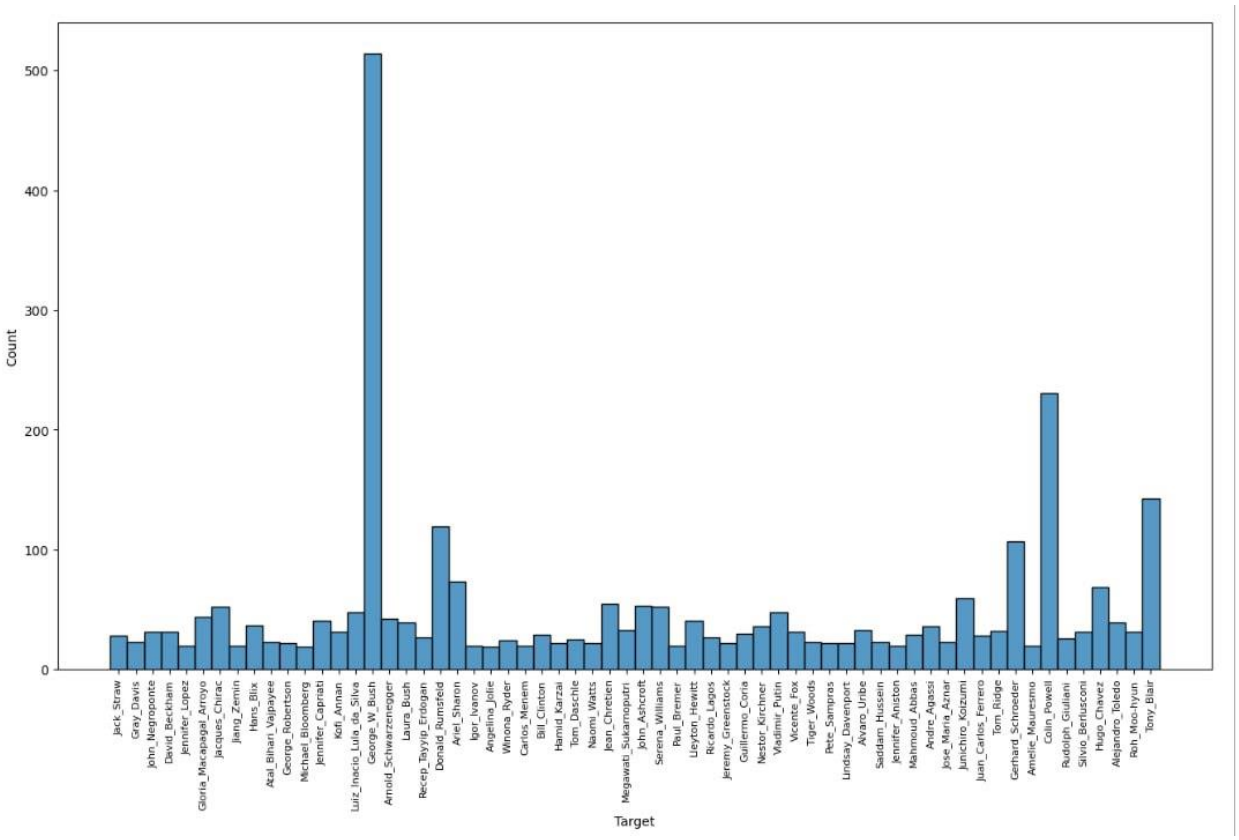
[198] Python

```
Number of identities: 62
```

After running this code, the people list will contain labels for each person, the features list will contain the corresponding images, and the images_aspect_ratios list will contain the aspect ratios of the images, which might be useful for visualization or analysis. Each image is cropped-out with 30px from left and 30px from top, to reduce the size of the image. And also apply resizing with 75% quality as well. As we mentioned above, we will change the color format of each image from RGB to GRAY.

Plotting the distribution of the number of images of each identity after filtering the image threshold to >= 20.





- Creating a DataFrame using the features and target labels obtained from the previous code.

```
create dataframe

    df = {'Features': features, 'Target': people}
    df = pd.DataFrame(df)
    df.head()
```

- Filter to get identity that have image equal to or more than 30 images

```
target_counts = df['Target'].value_counts()
valid_targets = target_counts[target_counts >= 30].index
```

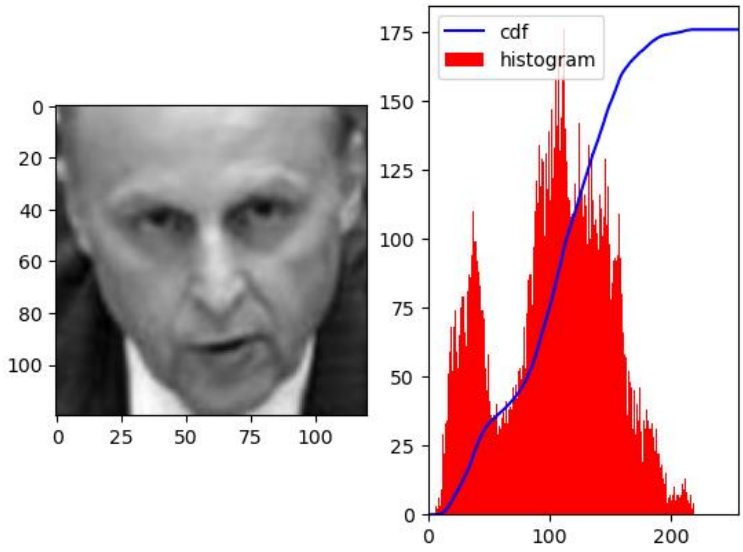
```
filtered_df = df[df['Target'].isin(valid_targets)]
```

```
filtered_df.reset_index(inplace=True, drop=True)
filtered_df.head()
```

	Features	Target
0	[[[15, 37, 65], [15, 39, 67], [19, 43, 73], [2...	John_Negroponte
1	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	John_Negroponte
2	[[[2, 0, 0], [2, 0, 0], [2, 0, 0], [2, 0, 0], ...	John_Negroponte
3	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	John_Negroponte
4	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	John_Negroponte

- Performing face detection using Haar cascades in OpenCV and then displaying a histogram and cumulative distribution function (CDF) of pixel intensity values for the detected face.

```
face_classifier = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
)

img = cv2.cvtColor(filtered_df.iloc[0]['Features'], cv2.COLOR_RGB2GRAY)

faces = face_classifier.detectMultiScale(
    img,
    scaleFactor=1.2,
    minNeighbors=5,
    minSize=(20, 20)
)

for x,y,w,h in faces:
    img = img[y:y+h,x:x+w]


plt.subplot(1, 2, 1)
plt.imshow(img, cmap = plt.cm.gray)

hist, bins = np.histogram(img.flatten(), 256, [0, 256])

cdf = hist.cumsum()
cdf_normalized = cdf * hist.max() / cdf.max()

plt.subplot(1, 2, 2)
plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(), 256, [0,256], color='r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.show()
```



- Processing each image in the “filtered_df” DataFrame, detecting faces using the Haar cascade classifier, resizing the detected face, and then performing histogram equalization. Finally, the flattened and processed face is added to the listFace. If a face is not detected in an image, the corresponding label in the DataFrame is set to "NaN". The processed faces are flattened to a one-dimensional array with a length of 80 * 100

```
listFace = []
for index in np.arange(len(filtered_df)):
    img = ''

    img = cv2.cvtColor(filtered_df.iloc[index]['Features'], cv2.COLOR_RGB2GRAY)

    faces = face_classifier.detectMultiScale(
        img,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
    )

    for x,y,w,h in faces:
        img = img[y:y+h,x:x+w]

    if img.shape[1] > 0 and img.shape[0] > 0:
        img = cv2.resize(img, (80,100), interpolation = cv2.INTER_AREA)

        hist, bins = np.histogram(img.flatten(), 256, [0, 256])

        cdf = hist.cumsum()
        cdf_normalized = cdf * hist.max() / cdf.max(),

        cdf_m = np.ma.masked_equal(cdf, 0)
        cdf_m = (cdf_m - cdf_m.min())*255 / (cdf_m.max()-cdf_m.min())
        cdf = np.ma.filled(cdf_m, 0).astype('uint8')

        img = cdf[img]
        #_, img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
        listFace.append(img.reshape(80*100))
    else:
        filtered_df.iloc[index, 1] = "NaN"
```