WEEK 2 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: 19th November 2023

Bachelor of Computer Science

Cambodia Academy of Digital Technology

- Apply Image Augmentation to train set.

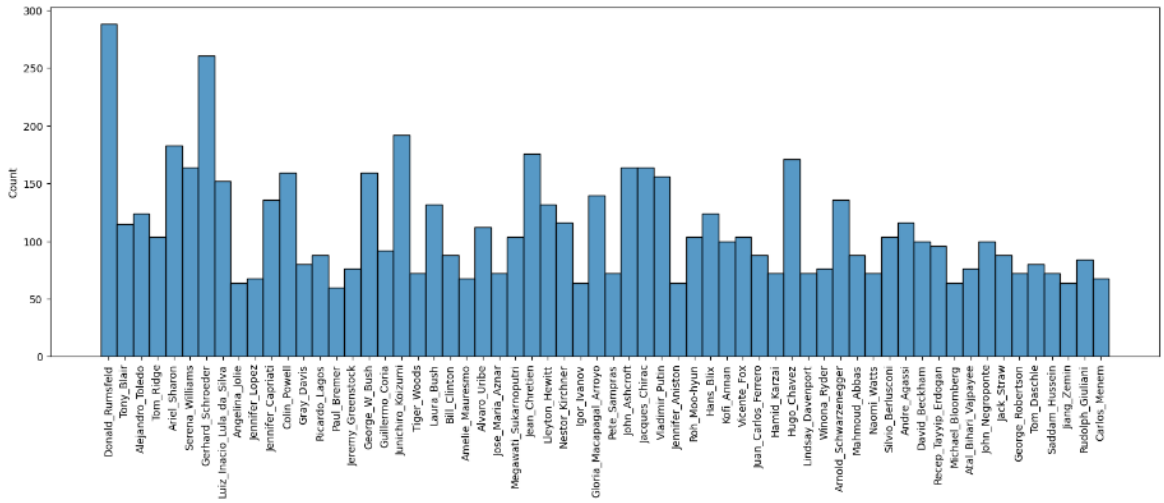
```
#Apply Image Augmentation to train set

import random
import imgaug as ia
import imgaug.augmenters as iaa

hflip= iaa.Fliplr(p=1.0)
vflip= iaa.Flipud(p=1.0)
rot = iaa.Affine(rotate=(random.randint(-180,180), random.randint(-180,180)))

target, counts = np.unique(targets_train, return_counts=True)
indentity_count = list(zip(counts, target))
image_with_name = list(zip(images_train, targets_train))

for ic in indentity_count:
    if(ic[0] < 50):
        for iwn in image_with_name:
            if(ic[1] == iwn[1]):
                for i in [hflip, vflip, rot]:
                    if(i != rot):
                        _img = i.augment_image(iwn[0])
                        images_train.append(_img)
                        targets_train.append(iwn[1])
                    else:
                        for count in range(1):
                            _img = i.augment_image(iwn[0])
                            images_train.append(_img)
                            targets_train.append(iwn[1])
    elif (ic[0] >= 50 and ic[0] < 100):
        for iwn in image_with_name:
            if(ic[1] == iwn[1]):
                for i in [hflip, vflip, rot]:
                    if(i != rot):
                        _img = i.augment_image(iwn[0])
                        images_train.append(_img)
                        targets_train.append(iwn[1])
                    else:
                        for count in range(0):
                            _img = i.augment_image(iwn[0])
                            images_train.append(_img)
                            targets_train.append(iwn[1])
print(f"Number of train images after augmentation: {len(images_train)}")
print(f"Number of train total target after augmentation: {len(targets_train)}")
print(f"Number of train identities after augmentation: {len(np.unique(targets_train))}")
```

```
Number of train images after augmentation: 6852
Number of train total target after augmentation: 6852
Number of train identities after augmentation: 62
```

  - The code above applies image augmentation techniques to a training set in machine learning. Image augmentation is a common technique used to artificially increase the size and diversity of a training dataset by applying various transformations to the images.

- Creates a histogram using "seaborn" to visualize the distribution of values in the "targets_train" dataset. The x-axis tick labels are rotated vertically for better visibility, and the plot is displayed with a specified figure size.

```
# Check Identity Count after apply augmentation for image training set

plt.figure(figsize=(19,6))
sns.histplot(targets_train)
plt.xticks(rotation=90, fontsize=10)
plt.show()
```



- Flattening them and converts the flattened images and targets into NumPy arrays.

```
# Convert from train images python list to numpy array

new_images = []
for image in images_train:
    new_images.append(image.flatten())
images_train = np.asarray(new_images)
targets_train = np.asarray(targets_train)
```

- Flattening them and converts the flattened test images and targets into NumPy arrays.

```
# Convert from test images python list to numpy array

new_images = []
for image in images_test:
    new_images.append(image.flatten())
images_test = np.asarray(new_images)
targets_test = np.asarray(targets_test)
```

- Visualizes a subset of the training images. It arranges these images in a 4x4 grid of subplots using matplotlib.

```
# Vitualize a random identity after agumentation from training set

zipped_image_00 = zip(images_train, targets_train)
counter = 0
fig,axs = plt.subplots(4, 4, figsize=(15, 15))
row=0
col=0
for (image, target) in zipped_image_00:
    try:
        if(target == targets_train[0]):
            axs[row, col].imshow(image.reshape(150, 150))
            col = col +1
            if (col == 4):
                row = row + 1
                col = 0
    except:
        continue
print(targets_train[0])
```

Donald_Rumsfeld

- Visualizing a random identity after augmentation from the training set can be helpful for understanding the effects of augmentation and examining the variations introduced to the images. It allows you to observe how augmentation techniques have transformed the images and explore the diversity within a particular identity or class.

- Standardizes the pixel values of the images in both the training and testing datasets using the "StandardScaler".

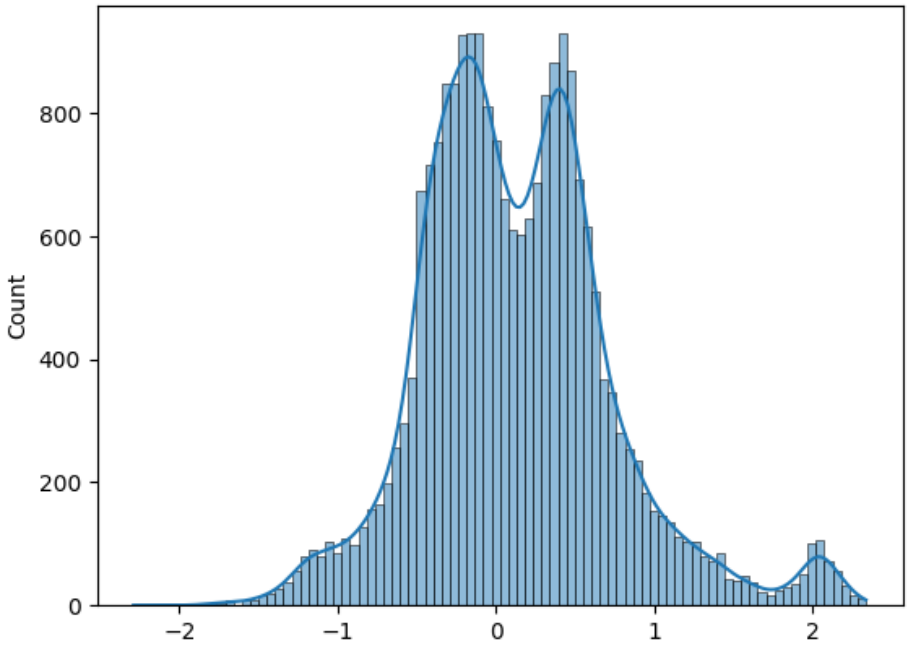
```
# Apply StandardScaler to both training and testing set
scaler = StandardScaler()
scaler = scaler.fit(images_train, targets_train)
images_train_scaled = scaler.transform(images_train)
images_test_scaled = scaler.transform(images_test)
```

- By applying the StandardScaler to both the training and testing sets, you ensure that the data is consistently scaled, prevent information leakage, maintain data integrity, and improve model compatibility. This practice promotes fair evaluation and reliable generalization of machine learning models.

- Create a histogram of the pixel values of the first image in the scaled training dataset using seaborn and check the mean and standard deviation of the pixel values of a different image in the scaled training dataset after applying the "StandardScaler".

```
# Check an images traing set after scaled
sns.histplot(images_train_scaled[0], kde=True)

# Check Mean and Std after applying Standardscaler (Mean ~ 0, Standard Diviation ~ 1)
print(f"Mean: {np.mean(images_train_scaled[1])}")
print(f"Standard Diviation: {np.std(images_train_scaled[1])}")
```

```
Mean: 0.781046829112766
Standard Diviation: 0.7180148920406083
```



- Use PCA to reduce the dimensionality of the scaled training and testing datasets while retaining 95% of the variance. The transformed data is stored in "`images_train_pca`" and "`images_test_pca`". The code also prints the shape of the transformed datasets to provide information on the dimensionality reduction achieved.

```
# Apply Principle Component Analysis with 95% of variance radio to both train and test

pca = PCA(0.95)
pca = pca.fit(images_train_scaled, targets_train)
images_train_pca = pca.transform(images_train_scaled)
images_test_pca = pca.transform(images_test_scaled)
print(f"After apply PCA for train: overall data is in shape {images_train_pca.shape}")
print(f"After apply PCA for test: overall data is in shape {images_test_pca.shape}")
```

```
After apply PCA for train: overall data is in shape (6852, 372)
After apply PCA for test: overall data is in shape (529, 372)
```

- Use Linear Discriminant Analysis (LDA) to further reduce the dimensionality of the data that has already undergone PCA dimensionality reduction. The resulting transformed datasets are stored in images_train_lda and images_test_lda. The code also prints the shape of the transformed datasets to provide information on the dimensionality reduction achieved by LDA.

```
# Apply LDA to both training and testing dataset

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda = lda.fit(images_train_pca, targets_train)
images_train_lda = lda.transform(images_train_pca)
images_test_lda = lda.transform(images_test_pca)
print(f"After apply LDA for train: overall data is in shape {images_train_lda.shape}")
print(f"After apply LDA for test: overall data is in shape {images_test_lda.shape}")
```

```
After apply LDA for train: overall data is in shape (6852, 61)
After apply LDA for test: overall data is in shape (529, 61)
```

- Creates a DataFrame "df" using the PCA-transformed training data and target labels. It then resets the index, renames a column to "target", and displays the first few rows of the resulting DataFrame.

```
# Convert numpy array to pandas dataframe

df = pd.DataFrame(images_train_pca, targets_train)
df.reset_index(inplace=True)
df.rename(columns={'index':'target'}, inplace=True)
df.head()
```

	target	0	1	2	3	4	5	6	7	8
0	Donald_Rumsfeld	23.664825	-1.843131	-48.101713	-1.136112	17.868632	-9.733655	18.474020	-13.817081	13.80886
1	Tony_Blair	118.783848	15.636983	-6.212145	-39.284672	7.591376	2.837273	16.860450	5.716318	17.77774
2	Alejandro_Toledo	29.830366	53.409524	-7.661550	17.112278	-29.832384	19.722066	-24.884003	9.760902	-9.10495
3	Tom_Ridge	10.720643	57.087977	-34.003495	5.422875	4.549083	20.242553	-7.144517	94.539266	-6.66671
4	Ariel_Sharon	21.014022	-34.452636	-29.303071	-7.267131	41.597879	-8.120352	0.429446	0.844415	3.036578

- Organize the data into feature and target variable sets "X_train" and "y_train".

```
# Split Data to features and target

X_train = df.drop('target', axis=1)
y_train = df['target']

#Check the counts of each identity

print(y_train.value_counts())
```

```
target
Donald_Rumsfeld      288
Gerhard_Schroeder    261
Junichiro_Koizumi    192
Ariel_Sharon         183
Jean_Chretien        176
                    ...
Michael_Bloomberg     64
Igor_Ivanov           64
Jiang_Zemin           64
Angelina_Jolie        64
Paul_Bremer           60
Name: count, Length: 62, dtype: int64
```

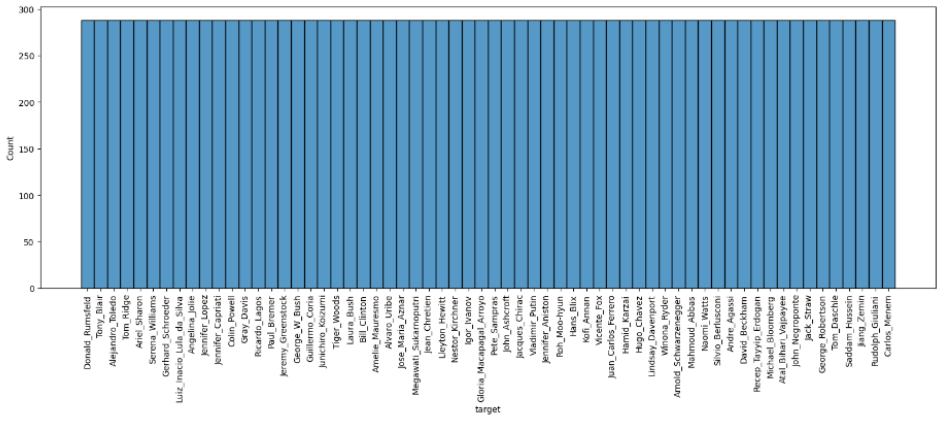
- Use SMOTE to oversample the minority class in the training data "X_train" and "y_train" to address class imbalance. The resulting oversampled data is stored in "X_res" and "y_res".

```
# Balance Data by using OverSampling SMOTE Aproach

from imblearn.over_sampling import SMOTE
smote= SMOTE(random_state=1, sampling_strategy="all")
X_res, y_res = smote.fit_resample(X_train, y_train)
```

- Show the total number of images for model training after applying SMOTE "y_res.shape" and visualizes the distribution of classes in the resampled target variable "y_res" using a histogram plot.

```
# Check the counts of each identity after applying over sampling technique

print(f"Total images for model training: {y_res.shape}")
plt.figure(figsize=(19,6))
sns.histplot(y_res)
plt.xticks(rotation=90, fontsize=10)
plt.show()
```

```
Total images for model training: (17856,)
```



- Start an SVM classifier with specific settings and fits it to the resampled training data ("X_res" and "y_res"). The "class_weight='balanced'" and "probability=True" settings are particularly useful for handling imbalanced datasets and obtaining probability estimates, respectively.

```
from sklearn import svm
clf = svm.SVC(class_weight='balanced',
 decision_function_shape='ovo',
 gamma='scale',
 kernel='linear',
probability=True)
```

```
# Fit model
clf.fit(X_res, y_res)
```

```
SVC(class_weight='balanced', decision_function_shape='ovo', kernel='linear',
    probability=True)
```

- Evaluate the performance of the SVM model on the test data by predicting labels "y_pred" and comparing them to the true labels "targets_test".

```
from sklearn.metrics import accuracy_score
y_pred = clf.predict(images_test_pca)
accuracy_score(targets_test, y_pred)
```

```
0.5255198487712666
```

- Generates a confusion matrix for the predictions of a classification model and displays it using a heatmap. The confusion matrix helps in understanding how well the model is performing in terms of classifying instances into different classes. The heatmap visualization makes it easier to interpret the patterns of correct and incorrect predictions.

```
# Confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
data = confusion_matrix(targets_test, y_pred)
data_display = ConfusionMatrixDisplay(confusion_matrix=data, display_labels=clf.classes_)
fig, ax = plt.subplots(figsize=(15,15))
data_display.plot(ax=ax)
plt.xticks(rotation=90)
plt.show()
```