Artificial Intelligence Capstone Project 1

110612117 張仲瑜

Collaborate with 110550128 蔡耀霆 in constructing dataset

1. Introduction

   Our research question is to solve a fine-grained classification task with images of 20 different car brands.

2. Dataset (link in the appendix)

   The original dataset includes 2773 car images in different sizes, ranging from 45 to 301 images for each class. The data was collected in a semi-automatic way. That is, we first use a web crawler to download the images of each brand from google search (Eg. Rolls Royce Spectre 1920x1080), and then manually remove the wrong data while labeling.

3. Methods

   For supervised learning, ResNet18 pretrained with imagenet-1k_V2, and fine tune with 10 epochs is used as deep learning-based method, and Support Vector Machine on Histogram of oriented gradient characteristic is used as another method.

   For unsupervised learning, 20-mean clustering is used with silhouette

score to observe how those experiments change the HOG discrepancy between each cluster.

4. Experiment and Result

- ✧ In this part, 5 folds cross validation are used to get better evaluation of the robustness of models.
- ✧ All the 5–fold–aggregated confusion metrices will be shown first, and the other indicators will be shown in table form in the end for reader's convenience to compare with each other.
- ✧ The experiment down below can be comparing to the baseline training status and serves as the only dependent variable.
- ✧ Baseline status
  - Resize directly to 224*224 without maintaining the aspect ratio
  - Without data balancing
  - Without data augmentation
  - Without dimensionality reduction

a. Supervised learning — Data resize comparison

I. Resize with aspect ratio retained,
and black pixel refilled if needed

II. Resize the shorter edge to 224 first, then scale the longer edge proportionally, and crop the image to 224*224 pixel

b. Supervised learning — Data balancing

I. Balance every class to 45 pictures

c. Supervised learning — Data augmentation

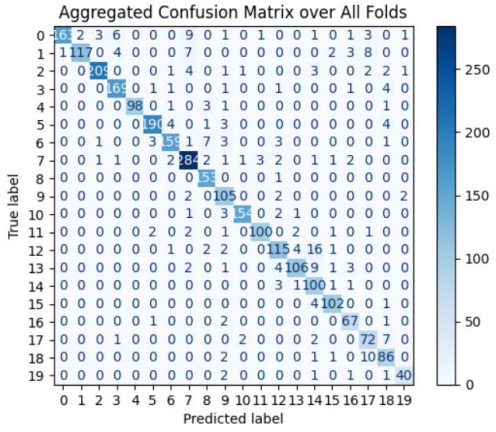
I. Horizontal Flipping

II. Image Sharpening, increasing lightness

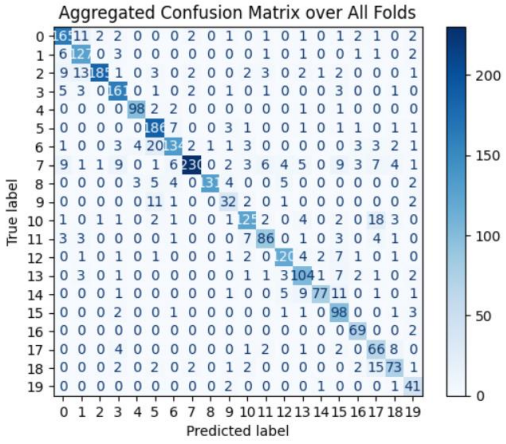
d. Unsupervised learning – observation

I. Silhouette score on HOG with experiment above

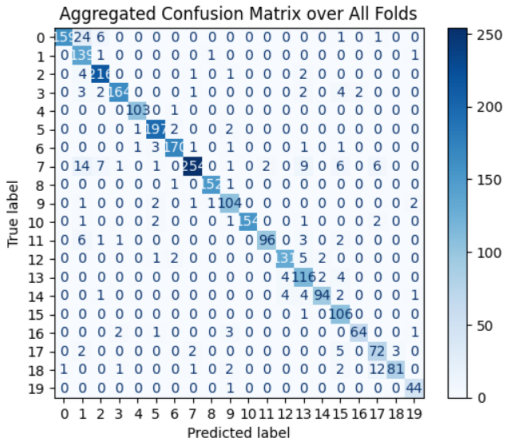
✧ Confusion metrices (ResNet18)



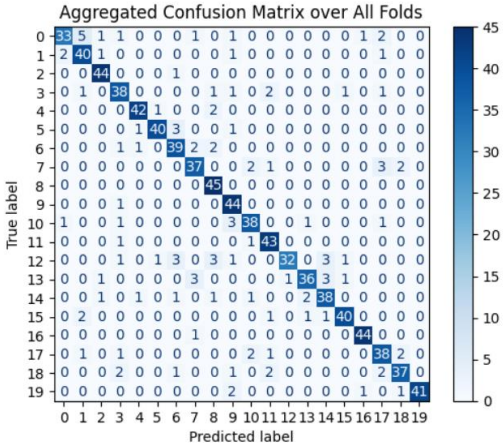
Resnet 18 - baseline



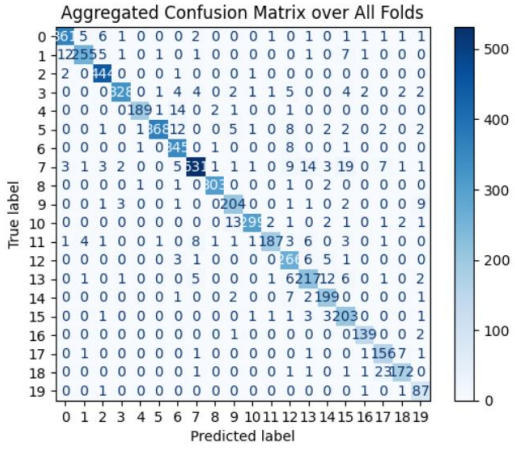
Resnet 18 – a1



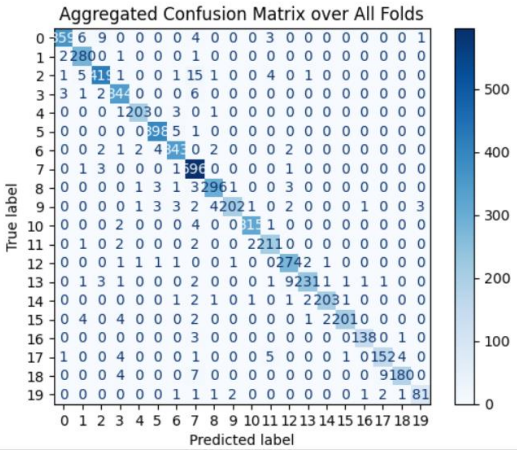
Resnet 18 – a2



Resnet 18 – b1



Resnet 18 – c1

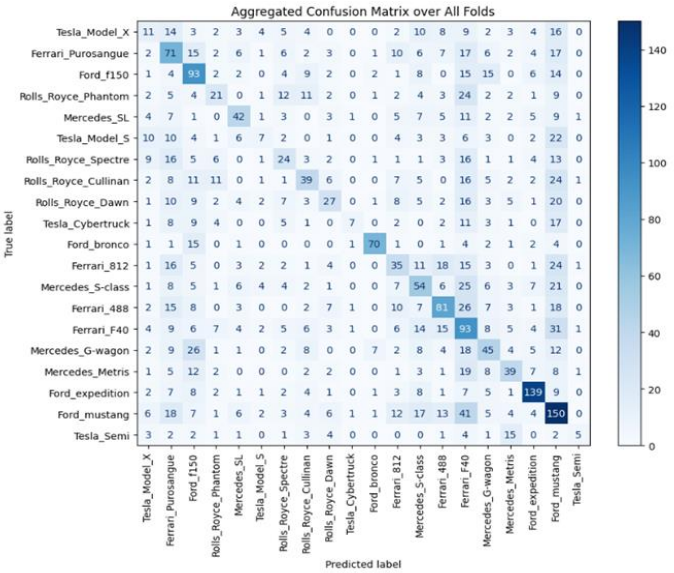


Resnet 18 – c2

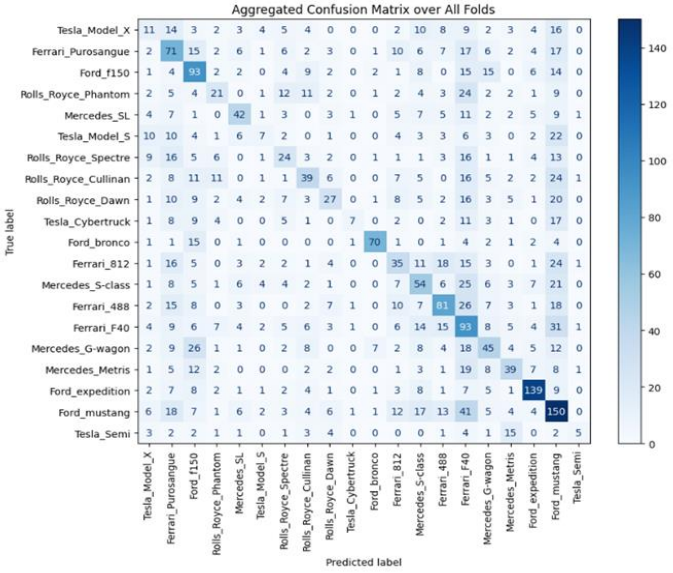
✧ Other indicators (ResNet18)

Indicators / Independent variable	Accuracy	Recall	Precision	F1 score
Baseline	0.913	0.913	0.929	0.916
a–1	0.832	0.832	0.874	0.836
a–2	0.923	0.923	0.944	0.922
b–1	0.877	0.877	0.901	0.875
c–1	0.926	0.926	0.938	0.927
c–2	0.957	0.957	0.961	0.957

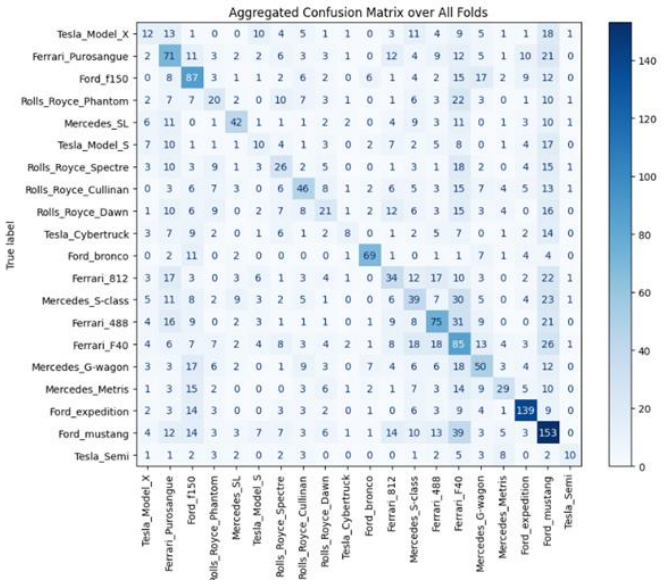
✧ Confusion metrices (HOG + SVM)



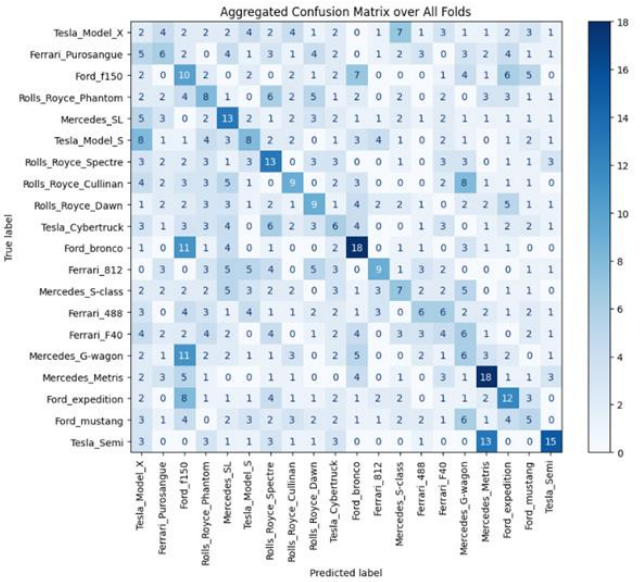
SVM - baseline



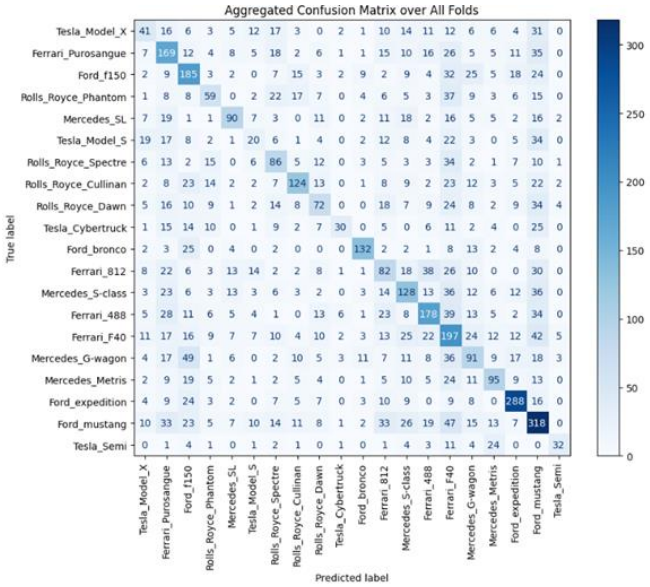
SVM – a1



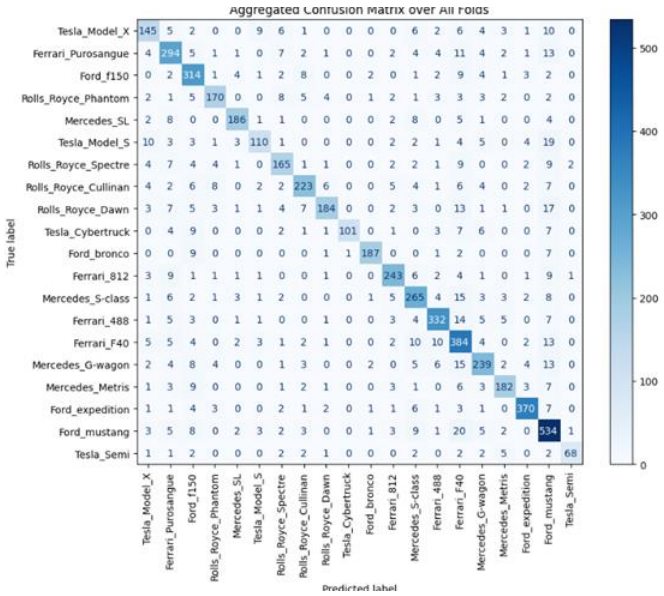
SVM – a2



SVM – b1



SVM – c1



SVM – c2

✧ Average Silhouette

Indicators / Independent variable	average silhouette
Baseline	0.0067
a–1	0.0084
a–2	0.0053
b–1	0.0021
c–1	–0.0002
c–2	–0.0003

5. Discussion

a. ResNet18

✧ I had expected that a1 (resize and pad with black) would be the best way when it comes to resize the image, but it turns out that a3 (resize and crop) performs the best. There might be two reasons. First, the black pixels cause harm to the learning process. Second, the cropping keeps the critical part of the car since the cars lie in the center of the pictures in most cases, which helps the model to concentrate on it.

✧ Data augmentation exerts gigantic influence when the scale is small as I expected. Especially the c2 (sharpening, increase lightness), it boosts the accuracy drastically in the fine–grained classification since it magnifies the tiny line difference between each brand.

- ✧ Though the result of data balancing is worse than the baseline, I think it was cause by the deduction of the data number. The performance is decent with nearly half pictures to train with. If I set the standard pictures number of each class to the median number of each class instead of the minimum class, and gathers more data to replenish the lacking classes, I speculate the performance will be on par with the baseline or even better.

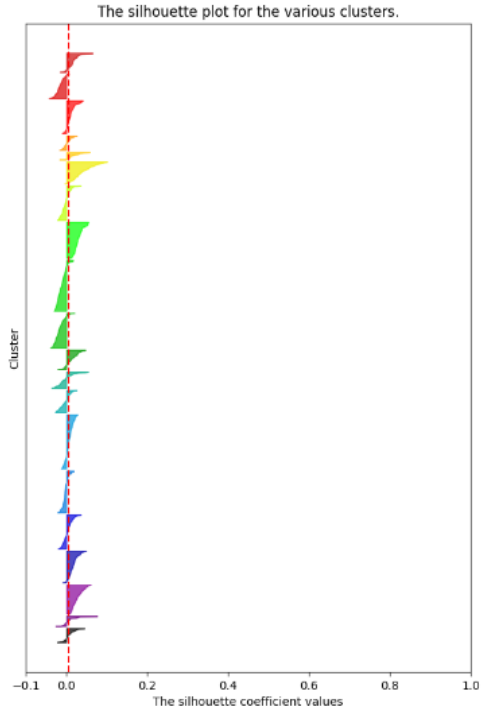
b. HOG + SVM

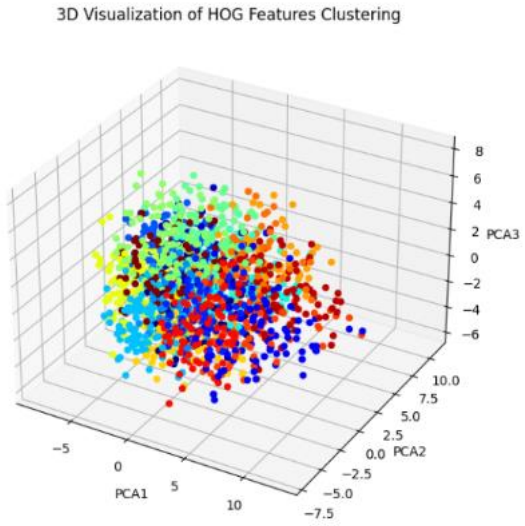
- ✧ It seems that classify the original pictures with HOG struggles to deal with the fine-grained classification task even though I've added the color channel. After researching, I think it's reasonable since it's sensitive to the perspective change due to the directional gradient statistics, and various angle pictures consist of single class. Furthermore, it's questionable for it to collect the global structure when it cut pictures into blocks and cells.

- ✧ c2 (sharpening, increase lightness) again massively increases all the indicators, and the result is well beyond my expectations. The sharpening plays a pivot role in enhancing the clarity of details of cars, which makes the directional gradient works effectively.

c. 20–clustering

Most of the experiments didn't boost the discrepancy between classes significantly, and lots of borders of clusters overlap with each other. The K–means clustering struggles to deal with the high dimensional data since most of the distance relations becomes ambiguous, which is too difficult to separate with it. PCA or other dimensionality reduction technics may be needed.





E.g. The negative part means the data may be clustered incorrectly.

(From one of the experiments.)

d. Summary

It's a magic journey for me to learn lots of technic while finishing the project simultaneously. It's my first experience of data collecting with web crawler and the manual labeling. It's also a valuable experience to attempt various trials and analyze the possible result. I would like to essay more characteristic like SIFT to utilize and more modern models like Swin-Transformer to aim for better performance if I have more time.

6. Appendix

- ✧ Dataset Link:

  https://drive.google.com/drive/folders/183k4o6cBABARknqFfAQ74wpbbRTHF53x?usp=sharing

- ✧ Code (critical only, all the full codes in the link)

  https://drive.google.com/drive/folders/1kO4_lVvvdjtxbXl74FgK326jgWnc6iZl?usp=sharing

  Though professor said that screen shot is not recommended, but reading code with text is awful, so I still decided to use it.

- ✧ Web Crawler

```
keywords_2 = ['Ford f150',
              'Ford mustang',
              'Ford bronco',
              'Ford expedition',
              'Ferrari 488pista',
              'Ferrari Purosangue',
              'Ferrari 812',
              'Ferrari F40',
              'Mercedes SL',
              'Mercedes G-wagon',
              'Mercedes Metris',
              'Mercedes S-class',
              'Tesla Model S',
              'Tesla Cybertruck',
              'Tesla Semi',
              'Tesla Model X',
              'Rolls Royce Dawn',
              'Rolls Royce Cullinan',
              'Rolls Royce Phantom',
              'Rolls Royce Spectre',]
```

Set the keyword of car brands as a list as part of the searching keyword.

```
36    url = f'https://www.google.com/search?hl=en&tbm=isch&q={keyword}
37            1920x1080 -site:https://cars.usnews.com/cars-trucks'
38    driver.get(url)
39    driver.maximize_window()
40    time.sleep(2)
41    for i in range(3):
42        driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
43        time.sleep(random.randint(2, 5))
44
45    image_elements = driver.find_elements(By.CLASS_NAME, 'rg_i')
46    for i, image in enumerate(image_elements):
47        try:
48            image.click()
49            high_res_image = WebDriverWait(driver, 5).until(EC.visibility_of_element_located((By.CLASS_NAME, 'sFlh5c.pT0Scc.iPVvYb')))
50            print(high_res_image)
51            image_url = high_res_image.get_attribute('src')
52            # print(image_url)
53            response = requests.get(image_url)
54            if response.status_code == 200:
55                with open(os.path.join(f"{keyword}", f"{keyword}_{i}.jpg"), 'wb') as file:
56                    file.write(response.content)
57        except Exception as e:
58            print(f"Error downloading image: {e}")
59            print(f"Failed to download image with URL: {image}")
60    driver.quit()
```

36~39

Browse the searching link as the format of google picture search

One of the websites will cause the lag, so I use the — technic to avoid the searching result from that website.

41~43

Scroll to the end for five times to access more pictures.

45~49

Click the small result first to get the high–resolution version of images and wait for at most 5 seconds.

53~56

Download the picture to the folder.

✧ Code — Experiments Preparations

I did the all the dataset change with python code on my notebook for two reasons, to avoid uploading time to Kaggle and to check the revised images clearly by album to make sure the result is what I want.

✧ Data Squeezing

```
9   for root, dirs, files in os.walk(folder_path):
10      for file in files:
11          if file.lower().endswith(('png', 'jpg', 'jpeg')):
12              try:
13                  img_path = os.path.join(root, file)
14                  img = Image.open(img_path)
15                  img = img.convert('RGB')
16                  img = img.resize((output_size, output_size), Image.Resampling.LANCZOS)
17
18
19                  relative_path = os.path.relpath(root, folder_path)
20                  output_dir = os.path.join(output_folder, relative_path)
21                  if not os.path.exists(output_dir):
22                      os.makedirs(output_dir)
23
24
25                  output_path = os.path.join(output_dir, file)
26                  img.save(output_path, 'JPEG')
27              except Exception as e:
28                  print(f"Error processing {img_path}: {e}")
```

9~11

Walk through all the files ends with common images format

13~16

Use the function provided by pillow to directly resize the images

19~22

Construct the same hierarchical path to the output folder

25~28

Output the revised image

✧ Data Cropping

```
6     ratio = target_size / min(img.size)
7     new_size = (int(round(img.size[0] * ratio)), int(round(img.size[1] * ratio)))
8     img = img.resize(new_size, Image.LANCZOS)
9
10
11    left = (img.width - target_size) / 2
12    top = (img.height - target_size) / 2
13    right = (img.width + target_size) / 2
14    bottom = (img.height + target_size) / 2
15
16
17    img = img.crop((left, top, right, bottom))
18    return img
```

Most of the part are same as the squeeze one.

Replace the resize function with the code above

6

Find the aspect ratio for adjustment

7~8

Use the resize function to adjust the shorter side to target size

(224) first

11~14

Set the new image to the center

17

Crop the image to 224 * 224

✧ Data Resize and Padding

```
4  def resize_and_pad(img, size, fill=0):
5      img.thumbnail((size, size), Image.Resampling.LANCZOS)
6      new_img = Image.new("RGB", (size, size), fill)
7      x = (size - img.size[0]) // 2
8      y = (size - img.size[1]) // 2
9      new_img.paste(img, (x, y))
10     return new_img
```

Most of the part are same as the squeeze one.

Replace the resize function with the code above

5

Resize with thumbnail function will maintain the aspect ration

6

Fill the blank part with fill (0 means black)

7~9

Set the new image to the center.

✧ Data augmentation

```
8  seq = iaa.Sequential([
9      iaa.Sharpen(alpha=(0.5,1.0), lightness=(0.5,1.0)),
10 ])
```

```
8  seq = iaa.Sequential([
9      iaa.HorizontalFlip(1.0)
10 ])
```

I used the imgaug library to achieve the data augmentation

It can create a sequential block the apply multiple augmentation on the data.

The upper one is the sharpening and lightening. The value means the extent you want to apply.

The lower one is the horizontal flip. The value means the chance it applies the augmentation. Since I want to add all of the horizontal–flipped images into the original one, I set p = 1.0 here.

```
30                  output_path = os.path.join(output_dir, file)
31                  imageio.imwrite(output_path, img)
32
33                  aug_output_path = os.path.join(output_dir, f"aug_" + file)
34                  imageio.imwrite(aug_output_path, img_aug)
```

30~31

I copy the original data into the new dataset with the augmented data.

33~34

Output the augmented data to the new dataset.

✧ ResNet18

```
model = models.resnet18(pretrained=True)
model.fc = nn.Linear(model.fc.in_features, 20)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Use the pretrained model, and set the number of classes to 20

Apply Cross Entropy Loss as the loss function.

Set learning rate = 0.001

```
k_folds = 5
kfold = KFold(n_splits=k_folds, shuffle=True)
```

Apply 5 folds cross validation

```
def imshow(img):
    img = img / 2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
```

Show the batches images to check whether the input data are

correct.

```
for images, labels in train_loader:
    images, labels = images.to(device), labels.to(device)
    optimizer.zero_grad()
    outputs = model(images)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    running_loss += loss.item()
print(f"Current epoch train loss = {running_loss / len(train_loader)}")
train_losses.append(running_loss / len(train_loader))

model.eval()
running_loss = 0.0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        running_loss += loss.item()
print(f"Current epoch test loss = {running_loss / len(test_loader)}")
valid_losses.append(running_loss / len(test_loader))
```

Main training and testing process and output the training loss and valid loss.

```
    cm = confusion_matrix(targets, predictions)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(cmap=plt.cm.Blues)
    plt.show()

    accum_predictions.extend(predictions)
    accum_targets.extend(targets)
    print(f'Accuracy (Weighted): {accuracy_score(targets, predictions)}')
    print(f'F1 Score (Weighted): {f1_score(targets, predictions, average="weighted")}')
    print(f'Recall (Weighted): {recall_score(targets, predictions, average="weighted")}')
    print(f'Precision (Weighted): {precision_score(targets, predictions, average="weighted")}')
    accuracies.append(accuracy_score(targets, predictions))
    f1_scores.append(f1_score(targets, predictions, average="weighted"))
    precisions.append(precision_score(targets, predictions, average="weighted"))
    recalls.append(recall_score(targets, predictions, average="weighted"))

print(f'Average Accuracy: {np.mean(accuracies)}')
print(f'Average F1 Score (Weighted): {np.mean(f1_scores)}')
print(f'Average Recall (Weighted): {np.mean(recalls)}')
print(f'Average Precision (Weighted): {np.mean(precisions)}')
cm = confusion_matrix(accum_targets, accum_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title('Aggregated Confusion Matrix over All Folds')
plt.show()
```

Upper part:

Record and show the confusion matrix and indicators of every folds.

Lower part:

Record and show the accumulated confusion matrix and average indicators of every folds.

✧ HOG + SVM

```
def extract_hog_features(image_path):
    image = io.imread(image_path, as_gray=False)

    hog_features = feature.hog(image, orientations=18, pixels_per_cell=(8, 8),
```

```
for train_index, test_index in skf.split(features, labels):
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = labels[train_index], labels[test_index]

    #clf = SVC(kernel='rbf', C=5.0, gamma=3.0)
    clf = SVC(kernel='linear', C=2.5, gamma='scale')
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)
```

Set the angle cut range, cell pixel size, block size, and color channel.

Use linear kernel to perform SVM to predict the class.

The rest of result plotting are same as ResNet18.

✧ K–mean Clustering with HOG + Silhouette score

```python
def process_images_with_hog_clustering_and_visualize(dataset_path, n_clusters=20):
    feature_vectors = []
    image_paths = []


    for root, dirs, files in os.walk(dataset_path):
        for file in files:
            if file.lower().endswith(('png', 'jpg', 'jpeg')):
                img_path = os.path.join(root, file)
                hog_features = extract_hog_features(img_path)
                feature_vectors.append(hog_features)
                image_paths.append(img_path)

    feature_vectors = np.array(feature_vectors)


    kmeans = KMeans(n_clusters=n_clusters, random_state=42).fit(feature_vectors)
```

Build the feature vector and set the clusters number to 20.

```python
silhouette_avg = silhouette_score(feature_vectors, kmeans.labels_)
print("For n_clusters =", n_clusters, "The average silhouette_score is :", silhouette_avg)
```

Calculate the average silhouette score and print it.

```python
sample_silhouette_values = silhouette_samples(feature_vectors, kmeans.labels_)
fig, ax1 = plt.subplots(1)
y_lower = 10
for i in range(n_clusters):
    ith_cluster_silhouette_values = sample_silhouette_values[kmeans.labels_ == i]
    ith_cluster_silhouette_values.sort()
    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i
    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)
    y_lower = y_upper + 10
ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")
```

Plot the individual silhouette according to each cluster.

✧ PCA to 3 dimensions to show the result of clustering

```
    pca = PCA(n_components=3)
    reduced_features = pca.fit_transform(feature_vectors)

fig = plt.figure(figsize=(15, 10))
ax = fig.add_subplot(121, projection='3d')
colors = plt.cm.jet(np.linspace(0,1,n_clusters))


for i in range(n_clusters):

    indices = np.where(kmeans.labels_ == i)[0]
    selected_indices = indices[:samples_per_cluster]

    for index in selected_indices:
        ax.scatter(reduced_features[index, 0],
                   reduced_features[index, 1],
                   reduced_features[index, 2],
                   color = colors[i],
                   label=f'Cluster {i}' if index == selected_indices[0] else "")

ax.set_xlabel('PCA1')
ax.set_ylabel('PCA2')
ax.set_zlabel('PCA3')
ax.set_title('3D Visualization of HOG Features Clustering')
```

Plot the figure on 3–dimensional space.

Thanks the hardworking TAs, the report ends here.