AI Capstone HW1

Introduction

In this homework, I present the development and training of a classification model aimed at classifying four common objects within reCAPTCHA questions. Leveraging machine learning techniques, we can use the model to answer the questions to bypass the the reCAPTCHA system.

Dataset description

Data source

The images were screenshotted in Google Map (mostly 2021), which is also the main data source of reCAPTCHA.

Data size

The dataset consists of 5 classes – 4 objects (bus, traffic light, zebra crossing, double yellow line) that often appear in reCAPTCHA questoins and “nothing”, which contains images that do not belong to the previous 4 classes. There are 100 images for each class, 500 images in total.

Preprocessing

Since the data were screenshotted manually, the size of the images is inconsistent. Firstly, I cropped the images from the middle of the image to make the image have square size and saved it as a version. Then, I resized the images to 64*64 with cv2 resize using bicubic interpolation and make it another version. The first version is to store the images for further experiment, and the second version is used as the input of supervised/unsupervised algorithms.

Method

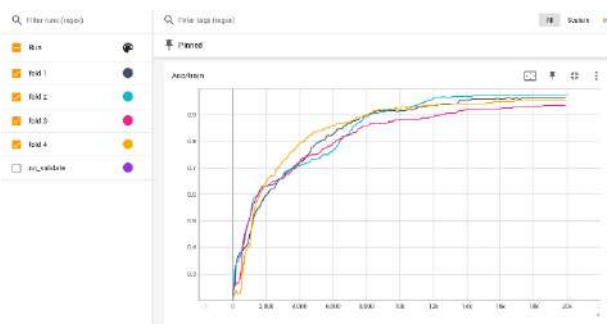
Data preparation

I split the dataset into training and testing set (4:1), and do 4 – fold cross validation on the training set, so the dataset size of training : validation : testing = 3 : 1 : 1.

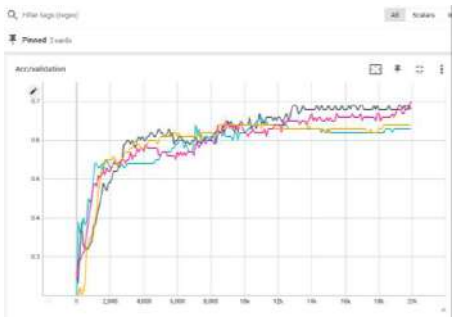
Supervised

- Method 1 - **Deep learning (CNN + NN)**

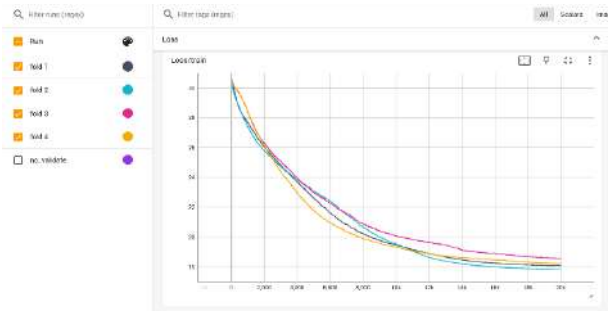
  This approach was implemented using pytorch. I used the full rgb image as input (size 3*64*64) and output a vector of probability of all classes. The input first goes through cnn for feature extraction, then be passed into 2 fully connected layers, and use the softmax of the vector as output. I tested both Adam and SGD optimizer, and the result of SGD optimizer is better. Here are the training curves.



Training



Validation



Loss

The following table indicates the accuracy of different folds, Fold 2 has the best validation accuracy, so it is chosen as the final model. The final testing result is 0.66 (accuracy 66%).

資料表

Fold	Training Accuracy	Validation Accuracy
Fold1	0.963333	0.68
Fold2	0.9767	0.7
Fold3	0.9367	0.63
Fold4	0.9567	0.68

Here is the confusion matrix. The result shows that the class “nothing” is the hardest to be correctly classified. This is intuitive because the data in “nothing” covers everything, and it will often be considered as noises in the dataset because it does not have a specific pattern to learn. Another interesting thing is that the model can not correctly classify zebra_crossing and double_yellow_line correctly. Despite having such similarity in both classes (both lines on the ground), it is still surprising that the model can not distinguish between the two classes.

資料表

True \ predicted	traffic_light	zebra_crossing	bus	double_yellow_line	nothing
traffic_light	16	2	0	0	2
zebra_crossing	1	13	2	2	2
bus	0	1	12	0	7
double_yellow_line	0	5	1	13	1
nothing	1	1	4	2	12

- Method 2 – **Random forest**

This approach is implemented using the random forest classifier in scikit-learn library and the default encoder in img2vec.
The following is the table of all the folds. The accuracy is way better than the previous method.

資料表

Fold	Training Accuracy	Validation Accuracy	Testing Accuracy
Fold1	1	0.95	0.94
Fold2	1	1.0	0.96
Fold3	1	0.95	0.98
Fold4	1	0.85	0.96

Unsupervised

- Method 3 – **K means clustering**

This approach is implemented using the kmeans library in scikit-learn and the default encoder in img2vec.
I use number of misclustered image / total image as the evaluation metric, and the result is 0.88. I also computed the sum of the entropy of each cluster. The result is 2, but this metric is not so intuitive thus

hard to say whether it is a good result or not.

Experiments

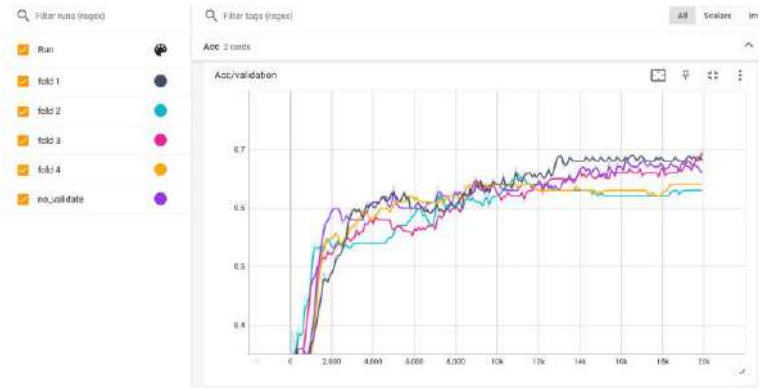
(The result of Method 2 is great, so the experiments were done on Method 1 & 3)

For Method 1 - Deep learning

- **Dataset (baseline for further experiments)**

  The deep learning approach overfits, and the modal size is already small, so I think the number of data is too small. I did 4-fold cross validation in Method 1, making all the classes have 60 pictures in total. Here I use all 80 pictures to train a new model, and I tested two random seeds in data shuffling. One model achieved accuracy of 0.74, which is a better result than all previous models, but the other one have only 0.66 accuracy. I think this is because the size of the dataset is still too small to represent the object.

  The following graph is the accuracy of validation for folds v.s. accuracy of testing set in this experiment.



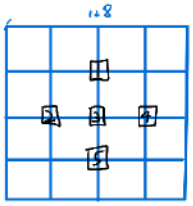
- **Gray scale**

  Grayscale is a technique that is often used to reduce input size. Here I grayscale the 64*64 images and trainined a model on them. The testing accuracy is 0.58, which is a little worse than the baseline, showing that grayscale is not helping the overall accuracy but only make the training faster. In addition, I tried to increase/decrease image size in the dataset. I used dataset version one (haven't resized yet so it has higher resolution), and resize them to (128, 128). The input size is a little

bigger than the original dataset, but it has higher resolution so I think it might achieve better performance. However, the testing result is 0.62, showing that it performs worse than the original dataset, but it did do better than the 64*64 grayscale images. I also tried resizing them to (32, 32), but the training accuracy is only 0.69 and the testing accuracy is 0.47, showing that the input size is too small to learn sufficient features. The testing accuracy curve is as follows.

- **Data augmentation**

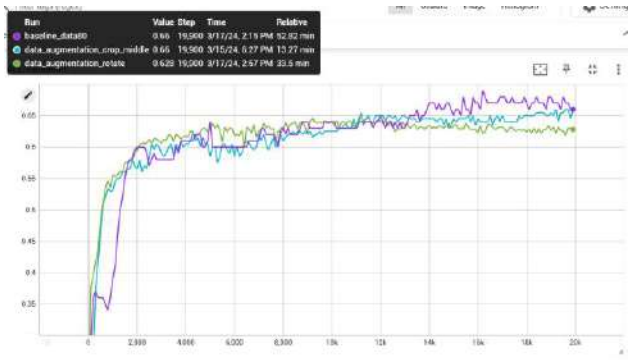
1. **Crop*5** - I reshape the image to 128*128 and crop the image 64*64 with the middle points of the following graph. The total size of the dataset is 500, which is way bigger than the origianl one. The testing accuracy was poor (about 40%). I think the reason for this result is that some of the pictures do not contain the actual object in the range, making the dataset full of noises.



2. **Crop*1 + resize** - Since I failed in the first data augmentation approach, I tried another approach to avoid what happened in the previous one. I use the middle of the 128*128 image (the 3 in the graph above) and resize the 128*128 image to 64*64 to create a new dataset. The testing result is 0.66, which is the same as the original one... This is reasonable because knowing more in the training set does not guarantee to know more about the testing set.

3. **Rotate** – I rotate all the data with opencv library to increase the

dataset size, making the dataset to have size of 400 images per class. The result is a little worse than the previous one (as shown in the following graph).



- **Graph – all experiments**

For Method 3 – K means clustering

I did the experiment on the rotated dataset, cut-middle (second in augmnentation part) dataset, and the grayscaled dataset. The result is as follows. The grayscale has the best clustering result (best accuracy and minimum entropy sum). The accuracy is obtained by dividing the correctly clustered label (I chose this by selecting the most frequent label of the cluster). The Davies-Bouldin score is defined as the average similarity measure of each cluster with its most similar cluster. The lower the score is, the better the clusters are separated. For the normal dataset and the grayscaled dataset, the grayscaled one has better accuracy, but it has bigger Davies Bouldin Score, meaning that it is not separated as well as the normal one. This shows that color can help dividing the points but is not so important to the overall performance.

資料表

Preprocessing Method	Accuracy	Davies Bouldin Score	Entropy Sum
normal	0.88	2.700001	2.001072
rotate	0.8045	2.857276	2.500544
cut middle	0.853	2.799105	2.367105
grayscale	0.894	2.851909	1.916932

Discussion

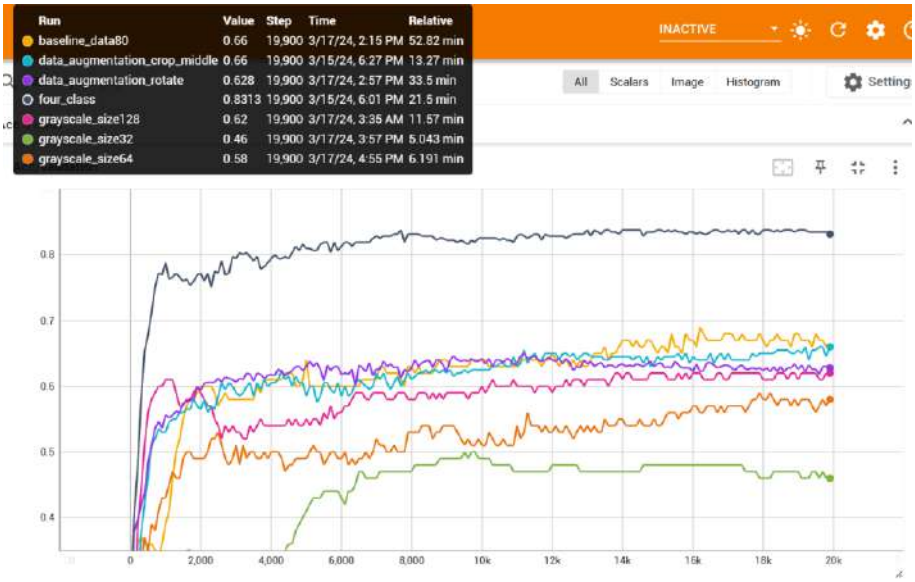
I already did a lot of discussions about the experiments in the previous part, so I'm going to talk about something else in this section.

Dataset Characteristic

I would like to talk about the "nothing class" in the dataset. This class consists of any kinds of objects except the objects in other classes. There is no pattern in the class, which make the multiclass classisfication become more difficult. As shown in the confusion matrix, it is the hardest to be correctly classified. I did an experiment on the 'nothing' dataset, and the accuracy without the 'nothing' class is way better than the previous models. With the 'nothing' class, if you have a new image with an object that looks like one of the objects but is not, you probably misclassify that to the object. The only way to solve this problem is to have a data in the training set to help the model tell the minor differences, so the size of the dataset will strongly affect the model.

Training curve

the one without 'nothing' class converge with higher accuracy.



Valdiation accuracy curve

the one without ‘nothing’ class out-perform others

Problem formulation

To pass the reCAPTCHA questions, a binary true/false classifier is sufficient. I modeled the whole problem as a multi-class classification problem for the following two reason. The first reason is to increase the difficulty of the task. If it was a binary classification problem, I think the model will perform well for sure, and it will not need any further experiments at all. The second reason is that if I do multiclass classification, I can know more about the dataset, like which class are often misclassified, or which class has the more distinguished feature.

Future experiments

There are a few experiments I want to do in the future.

1. Ensemble method

   My idea is to do binary classification on each class and pick the one with the highest confidence. This is likely to result in better accuracy.
2. Feature extraction method

   For the deep learning method now, I use CNN to extract features, but it might be not strong enough. Maybe I can try other feature extraction (autoencoder, HOG ...).

References

1. Google 地圖
2. PyTorch
3. Day-19 PyTorch 怎麼讀取資料? Dataset and DataLoader - iT 邦幫忙::一起幫忙解決難題，拯救 IT 人的一天 (ithome.com.tw)
4. scikit-learn: machine learning in Python — scikit-learn 1.4.1 documentation
5. img2vec-pytorch · PyPI
6. 機器學習: 集群分析 K-means Clustering. Python 範例，MATLAB 範例 | by Tommy Huang | Medium
7. 聚類演演算法評價指標——Davies-Bouldin 指數（Dbi）_davies-bouldin index-CSDN 博客

Appendix

(all the code can be found in stanleyshen2003/AI_Capstone (github.com))

Preprocessing

```
import cv2
import os
classes = ['traffic_light', 'zebra_crossing', 'bus', 'double_yellow_line',
'nothing']
new_dataset_name = 'dataset'

os.makedirs(os.path.join(new_dataset_name, 'test'), exist_ok=True)
os.makedirs(os.path.join(new_dataset_name, 'train'), exist_ok=True)
for category in classes:
    count = 0

    os.makedirs(os.path.join(new_dataset_name, 'test',category),
exist_ok=True)
    os.makedirs(os.path.join(new_dataset_name, 'train',category),
exist_ok=True)
    for file_name in os.listdir(os.path.join('square', category)):
        img_path = os.path.join('square', category, file_name)
        print(img_path)
        img = cv2.imread(img_path)
        res = cv2.resize(img, dsize=(64, 64), interpolation=cv2.INTER_CUBIC)
        #res = cv2.cvtColor(res, cv2.COLOR_BGR2GRAY)
        count += 1
        if count > 80:
            folder = 'test'
        else:
            folder = 'train'
        new_img = res
        cv2.imwrite(os.path.join(new_dataset_name, folder, category,
str(count).zfill(3)+'.jpg'), new_img)
```

Algo1 – CNN + NN

```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
import torch.optim as optim
import os
from PIL import Image
from torch.utils.tensorboard import SummaryWriter

class CnnNet(nn.Module):
    def __init__(self, num_classes=5, init_weights=True):
        super(CnnNet, self).__init__()

        self.cnn = nn.Sequential(nn.Conv2d(3, 32, kernel_size=8, stride=4),
                                 nn.ReLU(True),
                                 nn.Conv2d(32, 64, kernel_size=4, stride=2),
                                 nn.ReLU(True),
                                 nn.Conv2d(64, 64, kernel_size=3, stride=1),
                                 nn.ReLU(True)
                                 )
        self.class_nn = nn.Sequential(nn.Linear(1024, 512),
                                      nn.ReLU(True),
                                      nn.Linear(512, num_classes)
                                      )

        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        x = x.float() / 255.
        x = self.cnn(x)
        x = torch.flatten(x, start_dim=1)
        C_value = self.class_nn(x)
        C_value = torch.squeeze(C_value)
        return F.softmax(C_value, dim=1)
```

```
    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.orthogonal_(m.weight, np.sqrt(2))
                nn.init.constant_(m.bias, 0.0)
            elif isinstance(m, nn.Linear):
                nn.init.orthogonal_(m.weight, np.sqrt(2))
                nn.init.constant_(m.bias, 0.0)

def evaluate(model, dataloader):
    correct = 0.0
    total = 0.0
    with torch.no_grad():
        for i, (features, targets) in enumerate(dataloader):
            outputs = model(features)
            _, predicted = torch.max(outputs.data, 1)
            total += targets.size(0)
            correct += (predicted == torch.max(targets, 1)[1]).sum().item()
    return correct / total

class CustomDataset(Dataset):
    def __init__(self, root_dir, folds, size = 100):
        self.folds = folds
        self.root_dir = root_dir
        self.classes = ['traffic_light', 'zebra_crossing', 'bus',
'double_yellow_line', 'nothing']
        self.class_to_idx = {cls_name: idx for idx, cls_name in
enumerate(self.classes)}
        self.images = self.get_images(size)

    def get_images(self, size):
        images = []
        image_name = []
        for i in self.folds:
            for j in range(int(i*size/5), int((i+1)*size/5)):
                image_name.append(str(j+1).zfill(3)+'.png')
        for cls_name in self.classes:
```

```
            class_dir = os.path.join(self.root_dir, cls_name)
            for img in image_name:
                img_path = os.path.join(class_dir, img)
                image = Image.open(img_path)
                y = np.zeros(len(self.classes))
                y[self.class_to_idx[cls_name]] = 1
                images.append((np.array(image).transpose((2,0,1)), y))
        return images

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image, label = self.images[idx]
        return image, label

if __name__ == "__main__":
    # set some parameters
    epochs = 20000
    batch_size = 16
    lr = 0.00005
    fold_n = 1
    dataset_size = 100

    dataset = CustomDataset(root_dir='data/dataset/train/',folds=[0,1,2,3],
size = dataset_size)
    dataset_val = CustomDataset(root_dir='data/dataset/test/', folds=[4],
size = dataset_size)
    dataset_test = CustomDataset(root_dir='data/dataset/test/', folds=[4],
size = dataset_size)
    # create tensorboard writer
    writer = SummaryWriter()

    # create dataLoader
    dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
    dataloader_val = DataLoader(dataset_val, batch_size=batch_size,
shuffle=True)
```

```
    dataloader_test = DataLoader(dataset_test, batch_size=batch_size,
shuffle=True)

    # create model
    net = CnnNet()
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=lr)
    total_samples = len(dataset)
    n_iterations = np.ceil(total_samples / 4)

    # training loop
    for epoch in range(epochs):
        running_loss = 0.0
        for i, (features, targets) in enumerate(dataloader):
            # forward to model
            outputs = net(features)
            # calculate loss
            loss = criterion(outputs, targets)
            # back-propagation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            #print(loss)
            running_loss += loss.item()

        writer.add_scalar("Loss/train", running_loss, epoch)
        if(epoch % 100 == 0):
            print(f'epoch {epoch}/{epochs}')
            print(running_loss)
            train_eval = evaluate(net, dataloader)
            print("Accuracy: ", train_eval)
            writer.add_scalar("Acc/train", train_eval, epoch)
            val_eval = evaluate(net, dataloader_val)
            print("Validation Accuracy: ", val_eval)
            writer.add_scalar("Acc/validation", val_eval, epoch)

    test_eval = evaluate(net, dataloader_test)
    print("test accuracy: ", test_eval)
```

```
    torch.save(net.state_dict(),f"model_all_val_{val_eval}_test_{test_eval}
.pth")
```

Algo2 – RF

```
import os
import pickle

from img2vec_pytorch import Img2Vec
from PIL import Image
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


# prepare data

img2vec = Img2Vec()

data_dir = './data/dataset_aug'
train_dir = os.path.join(data_dir, 'train')
test_dir = os.path.join(data_dir, 'test')

classes = ['traffic_light', 'zebra_crossing', 'bus', 'double_yellow_line',
'nothing']
data = {}
for j, dir_ in enumerate([train_dir, test_dir]):
    features = []
    labels = []
    for category in classes:
        for img_path in os.listdir(os.path.join(dir_, category)):
            img_path_ = os.path.join(dir_, category, img_path)
            img = Image.open(img_path_)

            img_features = img2vec.get_vec(img)

            features.append(img_features)
            labels.append(category)

    data[['training_data', 'testing_data'][j]] = features
    data[['training_labels', 'testing_labels'][j]] = labels
```

```
fold_n = 0      # 0 1 2 3
data['validation_data'] = data['training_data'][fold_n*20:(fold_n+1)*20]
data['validation_labels'] =
data['training_labels'][fold_n*20:(fold_n+1)*20]
for i in range(20):
   del data['training_data'][fold_n*20]
   del data['training_labels'][fold_n*20]
# train model
model = RandomForestClassifier(random_state=0)
model.fit(data['training_data'], data['training_labels'])

# validation performance
y_pred = model.predict(data['training_data'])
score = accuracy_score(y_pred, data['training_labels'])
print(f'training accuracy: {score}')

# validation performance
y_pred = model.predict(data['validation_data'])
score = accuracy_score(y_pred, data['validation_labels'])
print(f'validation accuracy: {score}')

# test performance
y_pred = model.predict(data['testing_data'])
score = accuracy_score(y_pred, data['testing_labels'])
print(f'testing accuracy: {score}')
```

Algo3 – K-means clustering

```
import numpy as np
#import matplotlib.pyplot as plt
from PIL import Image
from sklearn.cluster import KMeans
from img2vec_pytorch import Img2Vec
from sklearn.metrics import accuracy_score,davies_bouldin_score
import os
import seaborn as sns
from scipy.stats import entropy
```

```
def get_entropy(labels):
    _,counts = np.unique(labels, return_counts=True)
    return entropy(counts)

# prepare data
img2vec = Img2Vec()

data_dir = './data/dataset'
datasize = 100
train_dir = os.path.join(data_dir, 'train')
test_dir = os.path.join(data_dir, 'test')

folders = [train_dir, test_dir]
classes = ['traffic_light', 'zebra_crossing', 'bus', 'double_yellow_line',
'nothing']
features = []

for category in classes:
    for folder in folders:
        for img_path in os.listdir(os.path.join(folder, category)):
            img_path_ = os.path.join(folder, category, img_path)
            img = Image.open(img_path_).convert(mode='RGB')
            img_features = img2vec.get_vec(img)
            features.append(img_features)

features = np.array(features)

# set & train model
np.random.seed(1)
kmeans = KMeans(n_clusters=len(classes),init='random')
kmeans.fit(features)
Z = kmeans.predict(features)
print(Z)

# evaluate
true_labels = []
for i in range(len(classes)):
```

```
    class_i = [sum([1 for num in Z[datasize*(i):datasize*(i+1)] if num ==
j]) for j in range(len(classes))]
    class_i = np.argmax(class_i)
    for j in range(datasize):
        true_labels.append(class_i)
true_labels = np.array(true_labels)

accuracy = round(accuracy_score(Z, true_labels), 4)
print(f"Accuracy using k-means clustering: {accuracy}")

davies = davies_bouldin_score(features, Z)
print(f"Davies Bouldin score using k-means clustering: {davies}")

total = 0
for i in range(len(classes)):
    indices = [int(j/datasize) for j, x in enumerate(Z) if x == i]
    total += get_entropy(indices)
    print(f"Entropy of cluster {i}: {get_entropy(indices)}")
print(f"Entropy sum of all clusters: {total}")
```