CV HW5 Classifier

Group 6

- 0516054 劉雨恩
- 0756049 許耕福
- 0856736 鄭鈞聯

Introduction

This assignment is to let us know how to implenment tiny images representation, nearest neighbor classifier, bag of SIFT representation, and linear SVM classifier and use them to classify 15 catagory of images. Besides, we could also use deep learning to solve the task.

Implementation procedure

1. Tiny images representation + nearest neighbor classifier
2. Bag of SIFT representation + nearest neighbor classifier
3. Bag of SIFT representation + linear SVM classifier
4. (Bonus) Resnet

Tiny images representation

- In this part, we first resize image to 16 x 16, and then directly use raw pixel as the descriptor of image.
- After getting descriptor, we then can train a k-Nearest Neighbor classifier.

Get image descriptor

```python
def tiny_img(img, size=(16, 16)):
    return cv2.resize(img, size, interpolation=cv2.INTER_AREA)

imgs = []
for path, label in zip(train_img, train_label):
    img = cv2.imread(os.path.join(src_path, 'train', label[0], path[0
    img = tiny_img(img)
    imgs.append(img)
```

k-Nearest Neighbor classifier

- To implement knn, we first calculate the distance between testing data points and training data points.
- Next, we get the first k closest points and find the max cluster in these data points.
- To implement weighted knn, we use the inverse of distance as the weight when deciding cluster from k data points.

```python
def knn(train_img, train_label, test_img, test_label, k):
    correct = 0
    for img, label in zip(test_img, test_label):
        distances = pow(train_img - img, 2).sum(1) ** 0.5
        index = distances.argsort()
        neighbors = [train_label[idx] for idx in index[:k]]
        pred_label = get_label(neighbors, distances)
        if pred_label == label:
            correct += 1
    return correct * 100 / test_label.shape[0]

def get_label(neighbors, distances):
    labels = {}
    for label, distance in zip(neighbors, distances):
        if label in labels:
            labels[label] += 1. / distance
        else:
            labels[label] = 1. / distance
    return max(labels.keys(), key=(lambda key: labels[key]))
```

Bag of SIFT representation

- There will be three different ways to find SIFT representation, the first is use cv2's SIFT detection, the second is vl_sift and vl_dsift in vlfeat the third is dsift in cyvlfeat.

- There will be two return values in sift function. Key point will be a list of destination of key point and destination will be a numpy array of shape Number_of_Keypoints×128.
- Keypoint Descriptor: Now keypoint descriptor is created. A 16x16 neighbourhood around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

SIFT description in cv2

- After using this sift discription in cv2 the accuracy is around isn't good enough so we change to another approach.

```
sift = cv2.xfeatures2d.SIFT_create()
kp, des = sift.detectAndCompute(gray,None)
```

SIFT description in vlfeat

- According to the information from internet, I found that dense sift is better in classification so i choose to use vl_dsift in vlfeat. However, this approach is using python to run matlab code which may be really slow due to the communicatin. Besides, when setting step to 10 the accurancy could be over 50%.

```
eng = matlab.engine.start_matlab()
mat_img = matlab.single(value[i][1].tolist())
kp, des = eng.vl_dsift(mat_img, 'step', float(10), nargout=2)
```

Create SIFT description in dsift

- Using dsift in cyvlfeat will be the best approach. Not only the number of founded description points is really huge, but also the speed is really fast.
- Here is the code for setting step size to [5, 5] and we could find over 3 milions points.

```python
def create_sift_discription(train_data):

    description_bag = []
    sift_data_dic = {}

    for key, value in train_data.items():
        temp_list = []
        for i in range(len(value)):
            kp, des = dsift(value[i][1], step=[5, 5], fast=True)
            description_bag.extend(des)
            temp_list.append(des)
        sift_data_dic[key] = temp_list
    return description_bag, sift_data_dic
```

Run kmeans

- We use vl_kmeans to get the center and the assignment of the descriptions found in SIFT at first. Although the accuracy is great, the running speed is really slow, especially when using huge data point.
- We than choose the kmeans in cyvlfeat and the accuracy and running speed is really reasonable.

```python
vocab = kmeans(bag_of_features, cluster_num, initialization='PLUSPLUS
```

Create histogram

- This step is to create the histogram of every image in training data and test data. According th each images, we assign each features in the images to a cluster and add them up to from a histogram for every images.

```python
def get_bags_of_sifts(data_dict, vocab):
    feats, labels = [], []
    for i, label in enumerate(data_dict.keys()):
        for descriptors in data_dict[label]:
            distances = distance.cdist(descriptors, vocab, 'euclidean
            clusters = np.argmin(distances, axis=1)
            hist, bin_edges = np.histogram(clusters, \
                    bins=np.arange(len(vocab)+1), density=True)
            feats.append(hist)
            labels.append(i)
            assert len(hist) == (len(bin_edges)-1)
    return feats, labels
```

Linear SVM classifier

- In this step we use libsvm to implement the linear SVM procedure. In libsvm it provide multiclass classification so we could call the function directly.

```python
train_x = [{(i+1): feat[i] for i in range(len(feat))} for feat in tra
train_y = [(label+1) for label in train_labels]

prob = svm_problem(train_y, train_x)
param = svm_parameter('-t 0 -c 1e7 -b 1')
model = svm_train(prob, param)

test_x = [{(i+1): feat[i] for i in range(len(feat))} for feat in test
test_y = [(label+1) for label in test_labels]

p_label, p_acc, p_val = svm_predict(test_y, test_x, model)
```

(Bonus) Resnet

1. Dataloader
2. Training model
3. Testing data

0. Hyperparameter

- Batch size: 64
- Number of Resnet layers: 50
- Loss function: nn.CrossEntropyLoss()

- Optimizer: Adam
  - weight decay: 1e-5
  - learning rate: 1e-4 (decay*0.99 per EPOCH)

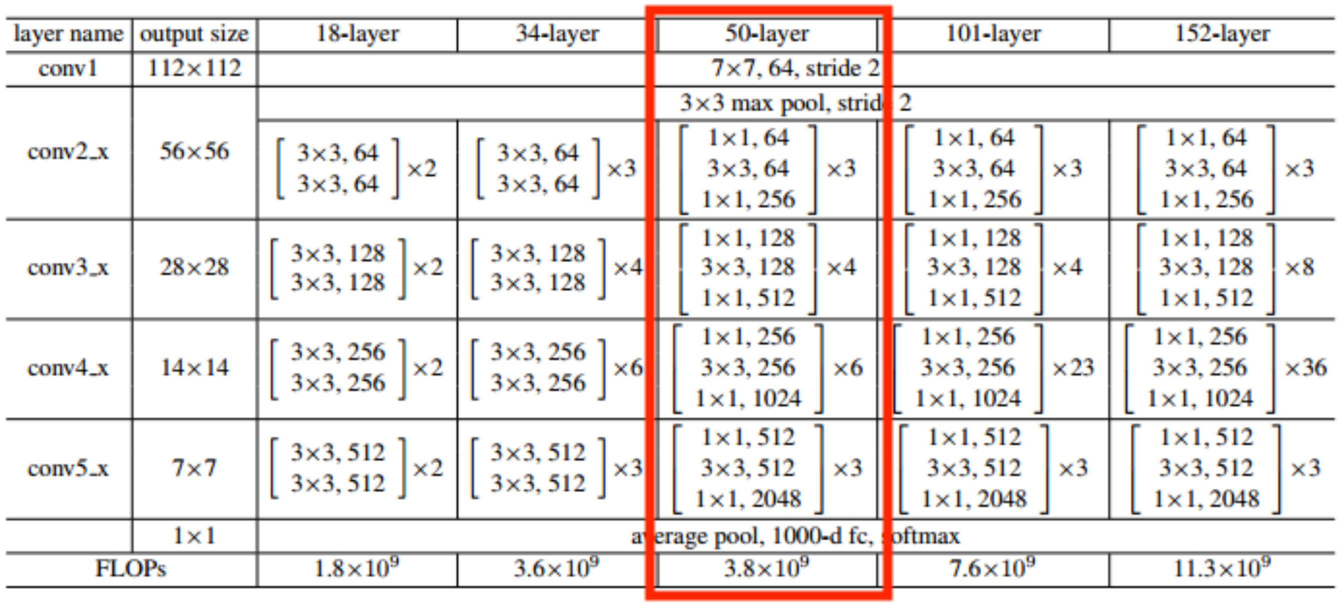
1. Dataloader

- Resize: Since the images have different shape, we have to resize the images to the same shape as (250, 250).
- Channel: Some channels of the images are 1 (gray scale), so we have to change them into 3 channels by doing `Image.convert('RGB')` .
- Normalization: Using `torchvision.transforms.ToTensor()` can make the values of the each image be normalized in range [0, 1]. Also, transform the [H, W, C] of the image into [C, H, W]. (H: Height, W: Width, C: Channel)

```
import pandas as pd
import torch as t
from torch.utils import data
import numpy as np
from PIL import Image
from torchvision import transforms

class dataLoader(data.Dataset):
    def __init__(self, src_path, mode):
        self.src_path = src_path
        self.mode = mode
        self.map_label = {'Bedroom': 0, 'Coast': 1, 'Forest': 2, \
        'Highway': 3, 'Industrial': 4, 'InsideCity': 5, 'Kitchen': 6,
        'LivingRoom': 7, 'Mountain': 8, 'Office': 9, 'OpenCountry': 1
        'Store': 11, 'Street': 12, 'Suburb': 13, 'TallBuilding': 14}
        self.img_name, self.label = self.getData()

        self.trans = transforms.Compose([transforms.Resize((250, 250)
                                                    interpolation=Image.NEARE
                                                 transforms.ToTensor(),
                                                 transforms.Normalize( \
                                                     mean=[0.485, 0.456, 0.40
                                                     std=[0.229, 0.224, 0.225

        print("> Found %d images..." % (len(self.img_name)))

    def __len__(self):
        """'return the size of dataset"""
        return len(self.img_name)

    def __getitem__(self, index):
        label_name = self.label[index]
        label = self.map_label[self.label[index]]
        path = self.src_path + self.mode \
                + '/' + label_name + '/' + self.img_name[index]

        img_origin = Image.open(path, 'r').convert('RGB')
        img = self.trans(img_origin)

        return img, label

    def getData(self):
        img = pd.read_csv(self.src_path + self.mode + '/img.csv')
        label = pd.read_csv(self.src_path + self.mode + '/label.csv')

        return np.squeeze(img.values), np.squeeze(label.values)

# main
train_data = dataLoader(src_path,'train')
train_len = len(train_data)
test_data = dataLoader(src_path, 'test')
test_len = len(test_data)

train_loader = Data.DataLoader(
    dataset=train_data,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=12,
)
test_loader = Data.DataLoader(
    dataset=test_data,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=12,
)

for epoch_i in count(1):
    # training
    for batch_x, batch_y in train_loader:
        ...

    # testing
    for batch_x, batch_y in test_loader:
        ...

```

2. Training model (Resnet 50, pretrained)

- Net arcitecture

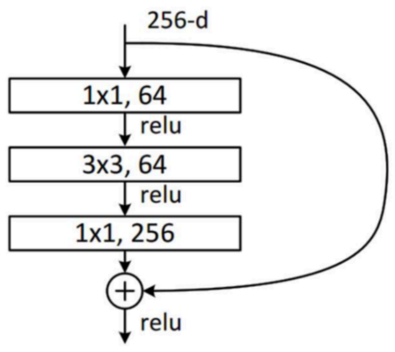


layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

- ResNet solves deep network problems
  - Vanishing/Exploding gradients
  - Degradation problem

- Residual Block:

Bottleneck block



```python
import torch.nn as nn
import torchvision.models

def lr_decay(optim):
    for param_group in optim.param_groups:
        param_group['lr'] = param_group['lr']*0.99

class ResNet_pre(nn.Module):
    def __init__(self, layer, pretrained=True):
        super(ResNet_pre, self).__init__()

        if(layer==18):
            self.classify = nn.Linear(512, 15)
        else:
            self.classify = nn.Linear(2048, 15)

        pretrained_model = torchvision.models.__dict__ \
                                    ['resnet{}'.format(layer)](pretrained=pre
        self.conv1 = pretrained_model._modules['conv1']
        self.bn1 = pretrained_model._modules['bn1']
        self.relu = pretrained_model._modules['relu']
        self.maxpool = pretrained_model._modules['maxpool']

        self.layer1 = pretrained_model._modules['layer1']
        self.layer2 = pretrained_model._modules['layer2']
        self.layer3 = pretrained_model._modules['layer3']
        self.layer4 = pretrained_model._modules['layer4']

        self.avgpool = nn.AdaptiveAvgPool2d(1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.classify(x)

        return x

# main
for epoch_i in count(1):
    right = 0
    for batch_x, batch_y in train_loader:
        batch_x = batch_x.cuda()
        batch_y = batch_y.cuda()

        # clear gradient
        optimizer.zero_grad()

        # forward + backward
        output = net.forward(batch_x.float())
        y_o = torch.max(output, 1)[1]
        y_hat = sum(batch_y == y_o).item()
        right += y_hat

        loss = loss_funct(output, batch_y)
        loss.backward()

        # update parameters
```

```
            optimizer.step()

        accuracy = right/train_len
        train_accuracy.append(accuracy * 100)
        fp_train.write(str(accuracy * 100) + '\n')
        print('\nEpoch {}, Train accuracy: {}'.format(epoch_i, accuracy))

        ...
```

3. Testing data

```
# main
for epoch_i in count(1):
    ...

    # test
    right = 0
    net.eval()
    with torch.no_grad():
        for batch_x, batch_y in test_loader:
            batch_x = batch_x.cuda()
            batch_y = batch_y.cuda()
            test_out = net.forward(batch_x.float())
            test_out = torch.max(test_out, 1)[1]
            test_hat = sum(batch_y==test_out).item()
            right += test_hat
    net.train()

    accuracy = right / test_len
    if(accuracy > highest_accuracy):
        torch.save(net.state_dict(), \
                    target_path+title+'_{}.pkl'.format(epoch_i))
        highest_accuracy = accuracy
        if highest_accuracy == 1:
            break
    test_accuracy.append(accuracy * 100)
    fp_test.write(str(accuracy * 100) + '\n')
    print('Test accuracy:', accuracy)
```

Experimental results

Tiny images representation + nearest neighbor classifier

- In this part, we found that when the k is equal to 12, the accuracy can reach to 18% in the testing dataset.

k	**1**	**2**	**3**	**4**	**5**
accuracy	13.3%	13.3%	13.3%	16.7%	16.7%

k	**6**	**7**	**8**	**9**	**10**
accuracy	15.3%	14.0%	16.0%	15.3%	16.7%

k	**11**	**12**	**13**	**14**	**15**
accuracy	15.3%	18.0%	17.3%	17.3%	16.7%

Bag of SIFT representation + nearest neighbor classifier

- There are several parameters that we could change, for example, the SIFT description, the step of vl_dsift, cluster number, and the k in knn.
- After some testing we found that using dsift in cyvlfeat, step [5, 5], and set k to 1 in knn will be the most reasonable parameter. There will be some difference between the number of clusters using in kmeans.
- Ferthermore, when using knn there will be differences using norm1 or norm2, the result is the following.

norm2

cluster num	**100**	**200**	**500**	**1000**
accuracy	49.3%	47.3%	52.7%	52.7%

norm1

cluster num	**100**	**200**	**500**	**1000**
accuracy	58%	59.3%	63.3%	64.7%

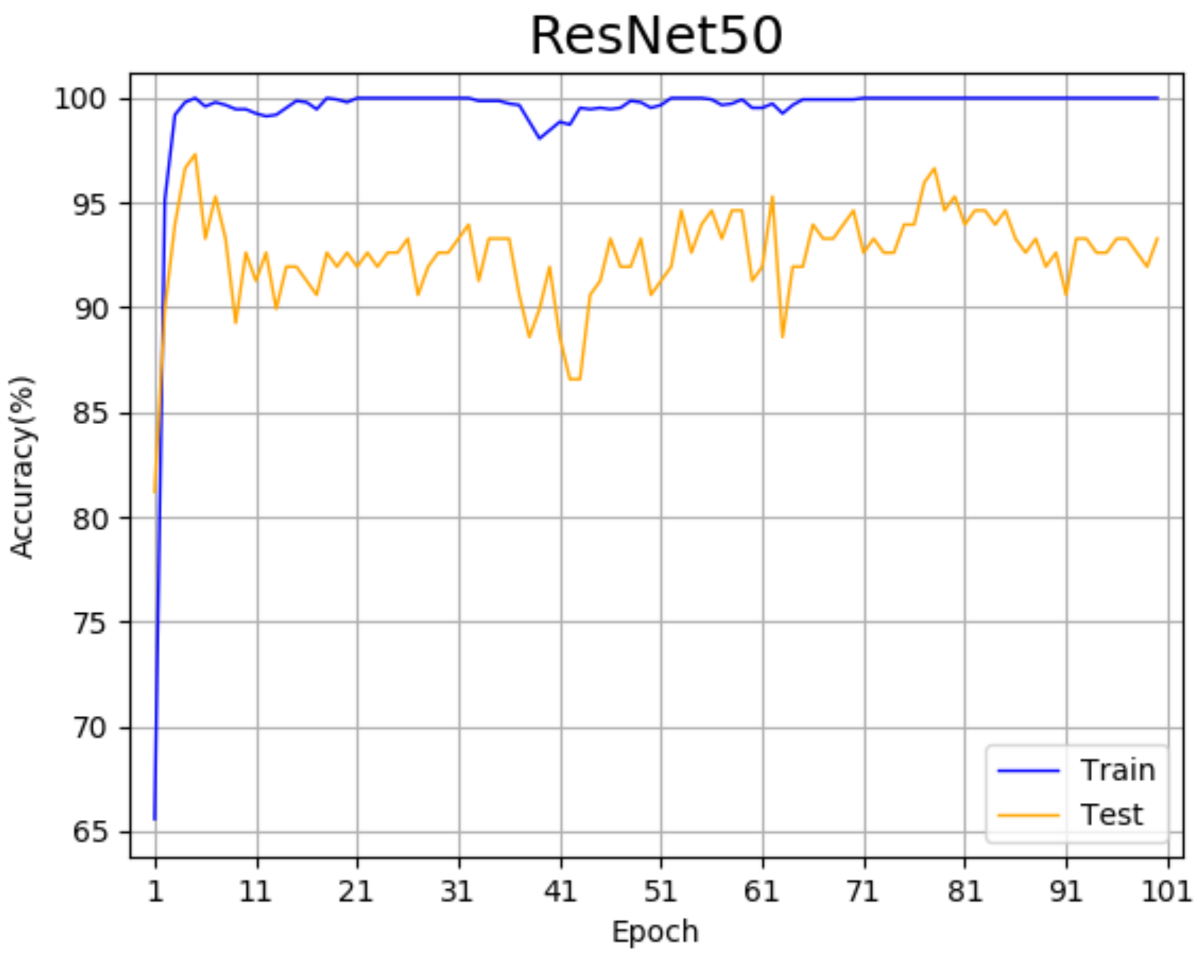
Bag of SIFT representation + linear SVM classifier

- As same as the previous part, we could choose several SIFT description. We simply use the dsift in cyvlfeat since we did several experiments before. However, if we increase the number of cluster, it will increase the accuracy rate obviously. When using 1000 clusters the accruacy could be 72%.

cluster num	100	200	500	1000
accuracy	63.3%	66.7%	68%	72%

Resnet

- Training process



  - Epoch: 100
  - Highest training accuracy: 100.0%
  - Highest testing accuracy: 97.315%

Discussion

nearest neighbor classifier

- In Bag of SIFT representation + nearest neighbor classifier part if we change the norm2 distance to norm1 the accurancy increased hugely. We are not sure why cause this kind of phenomenon.

Bag of SIFT representation

- In this assingnment we are not allowed to use sklearn. However, I had met many problems using vlfeat. The first thing I met is that I couldn't setup vlfeat in python. Therefore, I tried to install vlfeat in matlab and use python to run matlab code. The result will be good enough but the running speed is really slow due to the trasmition between the two programs. Finally, I had finished setting up cyvlfeat and both the performace and speed is really reasonable.
- I had tried the sift function in cv2, vl_dsift, and dsift. The sift in cv2 could only found around 700 thousand keypoints. Using vl_dsift could found 2 milion keypoints if your step is only 5, 900 thousand when step is 10. If we use dsift with [5,5] step, we could get 3 millions points. The result using vl_dsift and dsift is much better than using sift in cv2.

linear SVM classifier

- We found that if we increase the cost in libsvm, the accuacy will increase largly. We use cost 1 at first with 200 clusters, the accuracy is only 38%. If we increase it to 4, it will turn to 44%. Last, we change it to 10^7 than the accracy will be 66.7%.
- The reason why adding the cost will increase the performace is in SVM we would like to minimum the function $\frac{1}{2}\|w\|^2 + C\sum_i \xi_i$ subject to $y_i\left(w^T x_i - b\right) - 1 + \xi_i \geq 0 \quad \forall i, \xi_i \geq 0 \quad \forall i$. There is a cost C in this function which represent the penalty of error terms ξ_i If we do not give the enough penaty the $\frac{1}{2}\|w\|^2$ part will be useless.

Resnet

- If the `shuffle` of the `Data.DataLoader` isn't `True` , the testing accuracy would be very bad as 3%. It's because the model overfits each class every time during each batch, and finally can't learn general case.
- As the batch size is increased, the accuracy also improves a lot, but the memory usage will change relatively. As a result, considering the machine performance, the improvement of the batch size will be limited.
- When the weight decay mechanism is added, the testing accuracy improved a little bit, while the training accuracy dropped a little bit. It's because the weight decay mechanism helps the model not to overfit the training data too much.

Conclusion

After finishing the assignment, we know about several machine learning tecniques to solve classification problem in computer vision. Besides, we also learn about how to use deep learning tecnique such as Resnet to solve the classification problems.

Work assignment plan

- We wrote our own code separately.
    - Bag of SIFT representation, nearest neighbor classifier & linear SVM: 0856736 鄭鈞聯
    - Tiny images representation & nearest neighbor classifier: 0756049 許耕福
    - Resnet: 0516054 劉雨恩
- Then, we use Messenger to discuss the problem we had encountered and finished the report together.