Group 5

Computer Vision - HW5

1. Introduction

In this assignment, we implement four different approaches to build classifiers to categorize images into one of 15 scene types. Three of them are conventional machine learning algorithms, such as KNN and SVM. The features are directly from pixel values or extracted by the SIFT algorithms. Another approach is a neural network model, CNN. We use a pre-trained model provided from torchvision, which makes the training process easier and be able to converge faster. The whole CNN implementation is under the Pytorch framework.

2. Implementation procedure

- Tiny images representation + nearest neighbor classifier
- Bag of SIFT representation + nearest neighbor classifier
- Bag of SIFT representation + linear SVM classifier
- (Bonus) ResNet

2.1. Tiny Image representation

Each image is simply resized to a small square with fixed resolution(16*16) by cv2.resize(), ignoring their aspect ratio.The entire image is a vector of 256 dimensions.

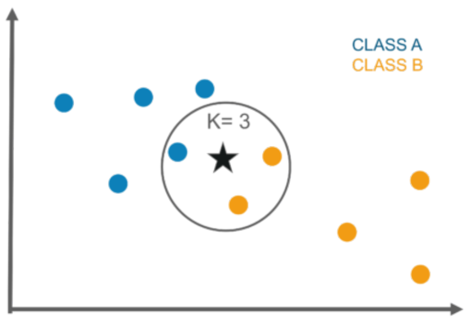
2.2. K nearst neighbor classifier

Every image representation from the training set constructs a database.

The distances between the representation of a test image and every image in the training set are calculated, and the label of the test image is voted based on the class of k, a chosen hyperparameter, nearest neighbors.

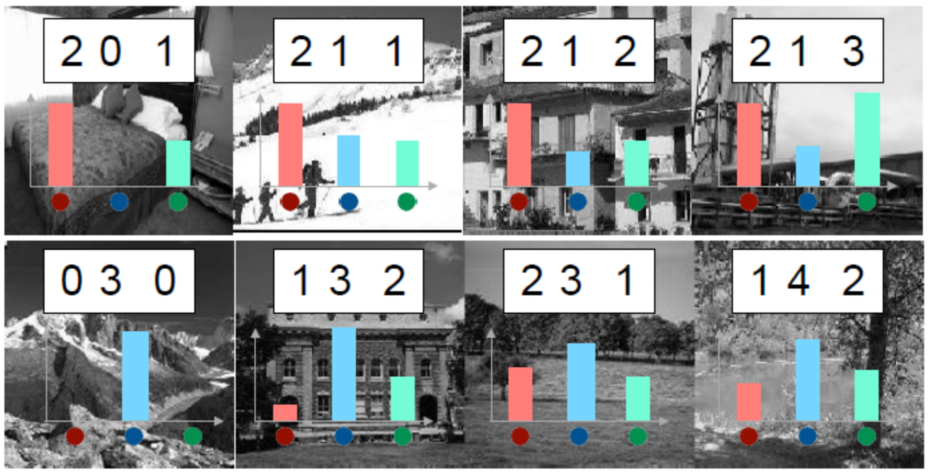
L1 distance = abs(test_feature - train_feature).sum()

L2 distance = square_root(test_feature - train_feature)**2.sum())



2.3. Bags of SIFT representation

This method is very similar to the former . The difference between are the feature types . This uses Dense-SIFT to generate features . After collecting the training data descriptor , we use k-means clustering on the whole descriptor . So each picture can be represented as a histogram with the number of each category .



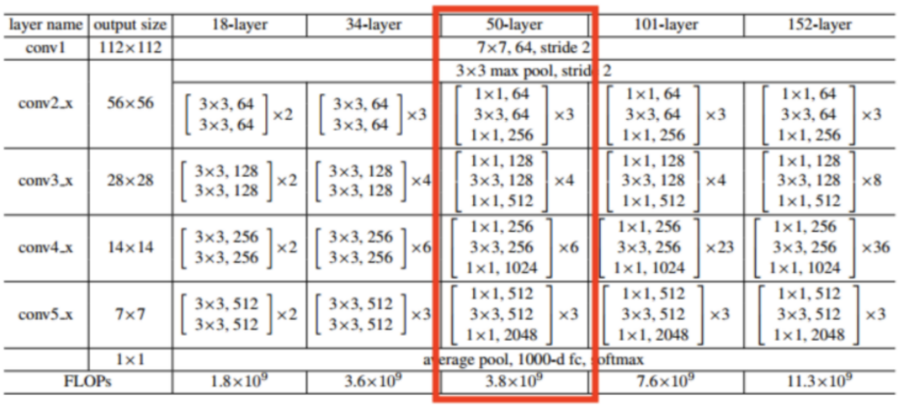
Because we have every category center point , so the testing data can also be represented as a histogram when testing phase . Then we use KNN or linear SVM to predict the final result .

2.4. Linear SVM classifier

Based on the Bag of SIFT feature, we use the libsvm to implement the SVM model training. SVM classifiers use the kernel function, which we use linear function in this homework, to divide data on hyperplane to classify 2 classes of points, meanwhile, keeping the maximized margin to the closest point. To improve the performance of the svm model, we have tested different Bag-of-words size to train to obtain the better result.

2.5. ResNet

- CNN model or deep learning model is a machine learning method to mimic how human beings are learning things. The CNN model learns how to classify images by extracting the feature from neural networks and adjust the parameters during the training epochs, and at the end converge to an optimal model.
- Hyperparmeter
    - epoch: 50
    - optimizer: SGD
    - learning rate: 1e-3
    - lr_scheduler: ReduceLROnPlateau
    - loss function: nn.CrossEntropyLoss()
    - number of model layer: 50
    - batch size: 30
    - Train/ Validation split: 0.8/ 0.2
- Dataloader
    - resize: Since the images are different size, we resize every image to the same shape (250, 250)
    - normalization: We normalize the image to put them into a common distribution in terms of size and pixel values, which avoid some bias during the training process.
    - ToTensor: The data type is required by the Pytorch framework.
    - Data augmentation: RandomHorizontalFlip, RandomRotation
- Model
    - ResNet50 architecture

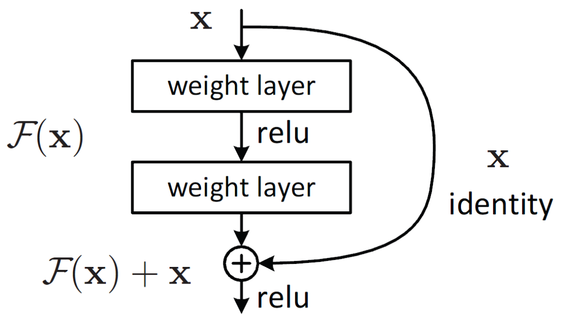


layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
conv2_x	56×56			3×3 max pool, stride 2		
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
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

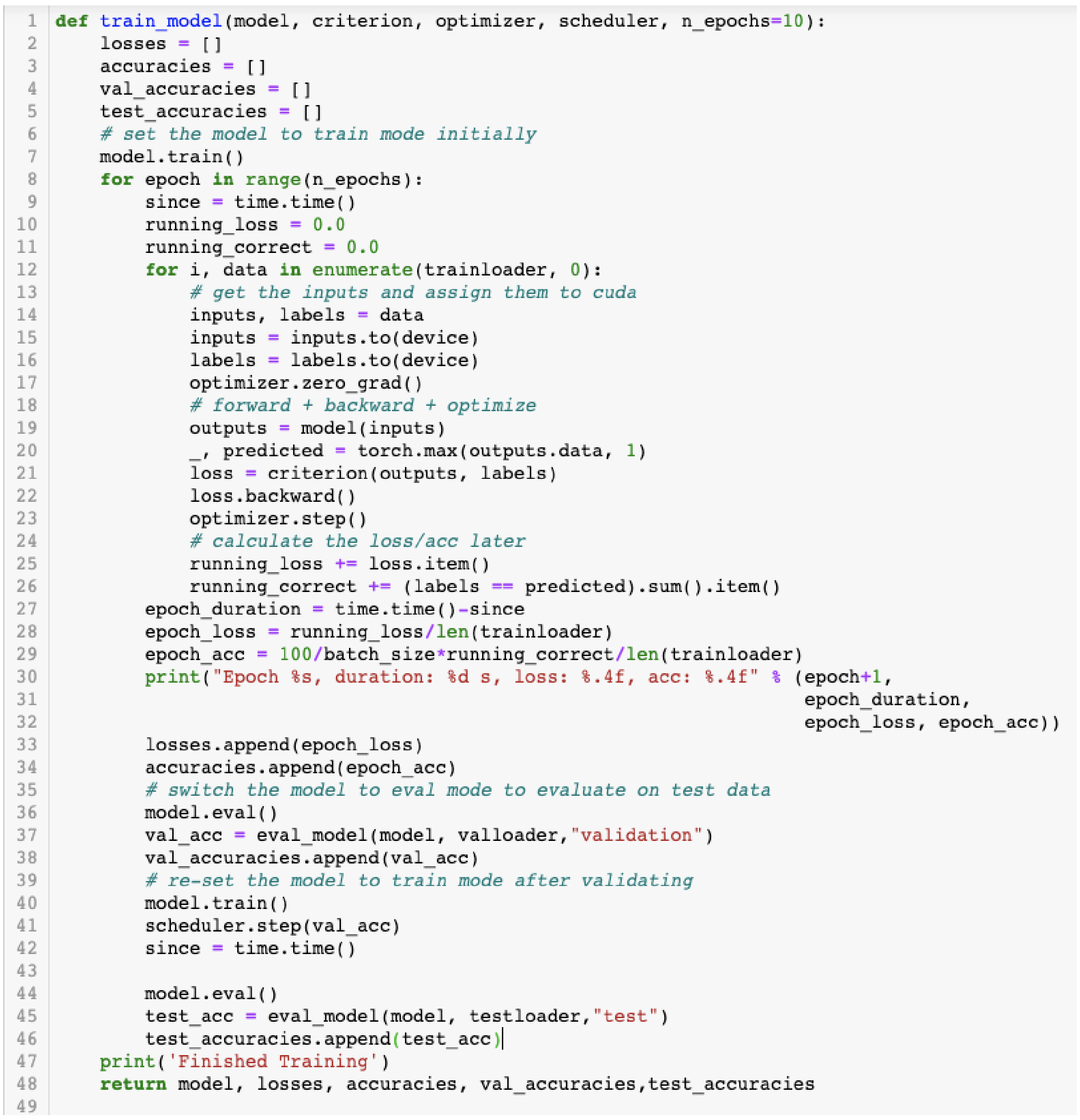
    - ResNet solves deep neural network problem

        - Vanish/ Exploding gradients
        - Degradation problem

    - Residual Block



- **Training**



```python
def train_model(model, criterion, optimizer, scheduler, n_epochs=10):
    losses = []
    accuracies = []
    val_accuracies = []
    test_accuracies = []
    # set the model to train mode initially
    model.train()
    for epoch in range(n_epochs):
        since = time.time()
        running_loss = 0.0
        running_correct = 0.0
        for i, data in enumerate(trainloader, 0):
            # get the inputs and assign them to cuda
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()
            # forward + backward + optimize
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            # calculate the loss/acc later
            running_loss += loss.item()
            running_correct += (labels == predicted).sum().item()
        epoch_duration = time.time()-since
        epoch_loss = running_loss/len(trainloader)
        epoch_acc = 100/batch_size*running_correct/len(trainloader)
        print("Epoch %s, duration: %d s, loss: %.4f, acc: %.4f" % (epoch+1,
                                                                   epoch_duration,
                                                                   epoch_loss, epoch_acc))

        losses.append(epoch_loss)
        accuracies.append(epoch_acc)
        # switch the model to eval mode to evaluate on test data
        model.eval()
        val_acc = eval_model(model, valloader,"validation")
        val_accuracies.append(val_acc)
        # re-set the model to train mode after validating
        model.train()
        scheduler.step(val_acc)
        since = time.time()

        model.eval()
        test_acc = eval_model(model, testloader,"test")
        test_accuracies.append(test_acc)
    print('Finished Training')
    return model, losses, accuracies, val_accuracies,test_accuracies

```

3. Experiment Result

3.1. Tiny images with KNN

KNN L1

k	4	5	6	10	26	50
acc	0.17	0.12	0.13	0.093	0.1	0.08

KNN L2

k	4	16	25	26	34	50
acc	0.14	0.17	0.20	0.21	0.21	0.18

3.2. Bag of SIFT representation with KNN

Bag-of-Words size =
[100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 400]

KNN k =
[10, 15, 20, 25, 30, 35, 40]
And show the best result with every vocabulary size

KNN L2

Bag-of-Words size	knn - k	accuracy
100	10	0.58
125	25	0.54
150	10	0.593
175	10	0.547
200	15	0.547
225	10	0.593
250	20	0.547
275	10	0.553
300	20	0.540

325	15	0.540
350	25	0.573
400	200	0.540

KNN L1
Vocabulary=150

knn - k	accuracy
10	0.58
15	0.613
20	0.627
25	0.60
30	0.627
35	0.613
40	0.633

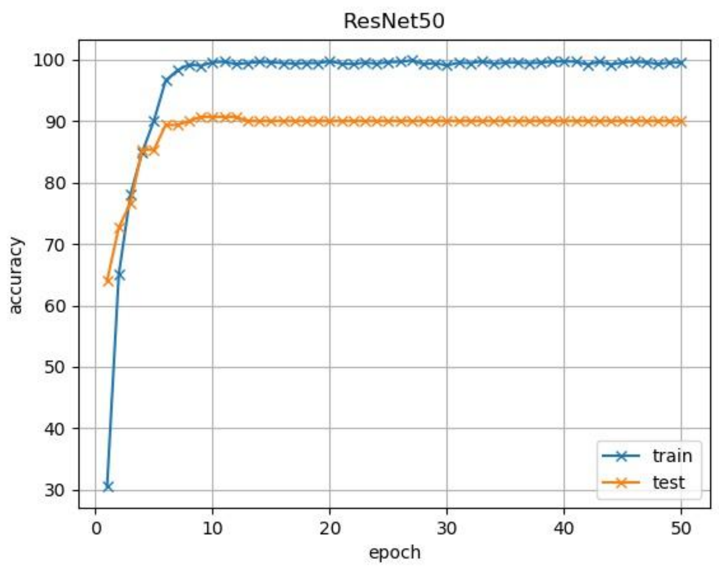
3.3. Bag of SIFT with SVM

Bag-of-Words size =
[100, 200, 400, 800, 1000, 1200, 1500]

Bag-of-Words size	accuracy
100	0.60
200	0.62
400	0.62
800	0.63
1000	0.67
1200	0.68
1500	0.66

3.4. ResNet

Highest training accuracy: 99.7%
Highest test accuracy: 90%



4. Discussion

4.1. In the beginning, we used the cv2 library to extract sift features. The performance couldn't reach 50% no matter how we adjusted the number of bag-of-words and KNN class. We noticed that Dense-SIFT might be a better approach on classification tasks. However, we took a long time to install a proper environment of cyvlfeat.

4.2. In Bag of SIFT representation, we found that SIFT and Dense-SIFT exist a big difference, the accuracy only reaches 40~49% accuracy. However, it reaches almost 59% accuracy when using Dense-SIFT and almost exceeds the expected accuracy describing in the TA's PDF .

4.3. Two methods to compute the feature distance in the KNN algorithm. We find that the task of tiny images gets higher accuracy with L2 Norm while the method of the bag of SIFT with KNN of L1 Norm obviously outperforms L2 Norm.

4.4. Choosing a proper number of k is an important issue. According to the results, different combinations of the number of bag-of-words and k from KNN affect the performance a lot. Moreover, the three tasks require different optimal parameters. We observed that the k from KNN should not be set too small and at least equal to 10 in the second task. The result of the third task is optimal when the number of bags-of-words equals to 1200.

4.5. The accuracy of the training set of the CNN model nearly converges to 100% while the accuracy of the test set is about 90%. We guess the batch size in the training process is too small, but we don’t have more GPU size to address the problem. On the other hand, the training set contains only 1500 images, which is not enough to generalize the model.

5. Conclusion

In this homework we have totally implemented four ways of classifying scenes. For the tiny image representation + kNN we got accuracy about 21%, Bag of SIFT representation + kNN about 62.7%, Bag of SIFT + SVM about 68% and finally the deep learning model ResNet50 with over than 90% accuray.
There are still many other models that can be tested to get a better result. As we observed from our work, we found out that the deep learning model without hand crafting features outperforms the feature generated by us which is quite amazing.

Work Assignment Plan with team members

We discussed and finished this assignment together.