Homework 5: kNN, Bag of SIFT, SVM and CNN

I-Sheng Fang
The Master's Degree Program in Robotics
National Yang Ming Chiao Tung University
isfang.gdr09g@nctu.edu.tw
Implementation, Experiment, Discussion

Hsiang-Chun Yang
Institute of Multimedia Engineering
National Yang Ming Chiao Tung University
yanghc.cs09g@nctu.edu.tw
Implementation, Experiment, Discussion.

Yu-Lin Yeh
Institute of Multimedia Engineering
National Yang Ming Chiao Tung University
s995503@gmail.com
Introduction, Discussion, Conclusion

Abstract

In this assignment, we implement the three kind of methods to distinguish 15 different scenes including tiny representation with nearest neighbor classifier, SIFT representation with nearest neighbor classifier and SIFT representation with linear SVM classifier. Moreover, we also adopt ResNet and its variants in this homework.

1 Introduction

In this homework, we need to conduct the scenes classifier with machine learning. In supervised machine learning, we build the mapping between features and the desired outputs of the task. For representation, we implement three methods, tiny representation, SIFT (Scale-Invariant Feature Transform) and NN (neural network). For classification, we implement three different approaches, kNN (k-nearest neighbor), SVM (Support Vector Machine) and NN. The kNN algorithm is a simple approach to classify testing data according to the their distances between different training data [1]. SIFT is a computer vision algorithm used to detect and describe local features in images. It finds the extrema in the scale space, and extracts its position, scale, and rotation invariant number [2]. SVM is a supervised learning method that uses the principle of statistical risk minimization to estimate the hyperplane of a classification boundary [3]. The NN is an end-to-end machine learning method which means it can learning the representation during the training. In this homework, we implement the ResNet as our backbone. The main advantage of ResNet is to solve the gradient vanish or explosion by using identity shortcut between each residual blocks [4].

2 Implementation Procedure

2.1 Tiny images representation with nearest neighbor classifier

First, we resize all images to 16*16 and use those 256 values to represent a single image. Second, for each testing image, we compute the L2 distance between the tiny representation of this image and tiny representations of all training images. Then, we will choose the k-nearest neighbor from those training images to determine the class of testing image.

We implement the PyTorch version of kNN with the following code segment.

```python
class kNN(nn.Module):
    def __init__(self, k, x_train, y_train, norm=2, num_classes=15):
        super().__init__()
        self.k = k
        self.x_train = x_train
        self.y_train = y_train
        self.norm = norm
        self.num_classes = num_classes

    def forward(self, x_test):
        dist = torch.cdist(x_test, self.x_train, p=self.norm)
        values, indices = dist.topk(self.k)
        return values, indices

    def predict(self, x_test, y_test):
        dist = torch.cdist(x_test, self.x_train, p=self.norm)
        values, indices = dist.topk(self.k, largest=False)
        y_pred = self.y_train[indices].reshape(x_test.size(0), -1)
        count, y_pred = F.one_hot(y_pred, self.num_classes).sum(1).max(1)
        acc = (y_test==y_pred).float().mean()
        return y_pred, acc
```

First, we use `torch.cdist` to build the distance matrix. Second, we use `torch.topk` to find the k-nearest neighbor. After we found the k nearest neighbor, we embed it to one-hot vector and do the voting. Finally, we choose the class with the most votes as the prediction.

2.2 Bag of SIFT representation with nearest neighbor classifier

First, we use SIFT to compute the descriptors of all training and testing images. Second, we apply K-means to the SIFT representations of training images to form k clusters. Then, we quantize representations of each training and testing image into a vector with length k. The i-th element in this vector represents the number of SIFT representations which belongs to the i-th cluster. And we will use these quantized vectors as training data to train the kNN model.

The following code segment is the implementation for computing SIFT representations. Here, we use `OpenCV` to detect and compute the SIFT descriptors.

```python
def sift(dataset):
    sift_descriptor = cv2.SIFT_create(
                            nfeatures=0,
                            nOctaveLayers=5,
                            contrastThreshold=0.01,
                            edgeThreshold=80,
                            sigma=0.6)
    des_per_x = []
    y = []
    for data in tqdm(dataset, ncols=80):
        kp, des = sift_descriptor.detectAndCompute(np.array(data[0]), None)
        des_per_x.append(des)
        y.append(data[1])
    return des_per_x, y
```

The following code segment are the implementations of vector quantization and K-means. For K-means, we use `faiss` package since it can support GPU acceleration.

```
def quantize(model, des_per_image, num_clusters, normalize=True):
    feature = np.zeros((len(des_per_image), num_clusters))
    for i in range(len(des_per_image)):
        _, assign_idx = model.assign(des_per_image[i])
        u, counts = np.unique(assign_idx, return_counts=True)
        counts = counts.astype(np.float32)
        feature[i,u] = counts
        if normalize:
            feature[i,u] /= counts.sum()
    return torch.tensor(feature)
```

```
km_model = faiss.Kmeans(
            d=des_vstack.shape[1],
            k=args.num_clusters,
            gpu=True, niter=300, nredo=10, verbose=True)
km_model.train(des_vstack)
```

2.3 Bag of SIFT representation with linear SVM classifier

Likewise, we use the same method to compute and quantize the SIFT representations in this section. In this section, we use SVM instead of kNN to do the classification. Here, we use `libsvm` to implement our SVM classifier and the implementation is as follows. The hyperparameter `C` here is used to control the width of the SVM margin.

```
model = svm_train(
            y_train.numpy(), x_train.numpy(),
            f'-s 0 -t 0 -c {args.c} -q')
res = svm_predict(y_test.numpy(), x_test.numpy(), model, '-q')
acc = res[1][0] / 100
```

2.4 Convolutional neural network (CNN)

We implement our CNN classifier with PyTorch. For transfer learning, we directly load the pretrained model from `torchvision`. We implement our model with two mode, pretrained weight or train from scratch. Because the model `torchvision` provided is trained with ImageNet dataset, we replace the last fully-connected layer with our own.

We use random horizontal flip and random resized crop for data augmentation during training. We use Adam optimizer which learning rate is set to 5e-4 and betas is set to (0.9, 0.999). The batch size is 64. The number of epoch is 200. The image size is 128x128.

```
class CNN_Model(nn.Module):
    def __init__(self, model_name, pretrained, num_classes=15):
        super().__init__()

        self.model_name = model_name
        self.pretrained = pretrained
        self.num_classes = num_classes
```

```
        self.last_fc = 'res' in self.model_name or \
                       self.model_name == 'googlenet'
        self.densenet = 'densenet' in self.model_name

        exec(f'self.model = models.{self.model_name}(pretrained=pretrained)',
            {'self': self, 'models': models, 'pretrained': self.pretrained})

        if self.last_fc:
            have_bias = self.model.fc.bias is not None
            self.model.fc = nn.Linear(in_features=self.model.fc.in_features,
                                      out_features=self.num_classes,
                                      bias=have_bias)
        elif self.densenet:
            have_bias = self.model.classifier.bias is not None
            self.model.classifier = nn.Linear(
                       in_features=self.model.classifier.in_features,
                       out_features=self.num_classes, bias=have_bias)
        else:
            raise NotImplementedError

    def forward(self, x):
        return self.model(x)
```

3 Experimental Result

3.1 Tiny images representation with k nearest neighbor classifier

Here, we try two different methods, with and without normalized the tiny representations. For each case, we also try various kinds of k for kNN classifier. Figure 1 is the experiment results. The highest accuracy 22.67% occurs when k is 19 and with data normalization.

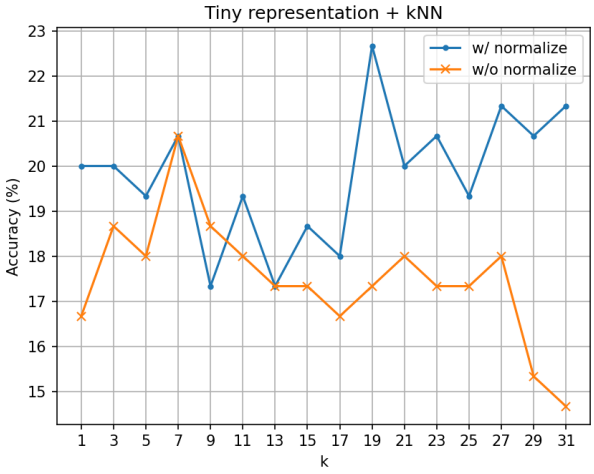

Figure 1: Comparison of tiny representations with kNN

3.2 Bag of SIFT representation with k nearest neighbor classifier

Here, we try four different numbers of clusters. For each case, we also try various kinds of k for kNN classifier. Figure 2 is the experiment results. The highest accuracy 59.33% occurs when number of clusters is 450 and k is 7.

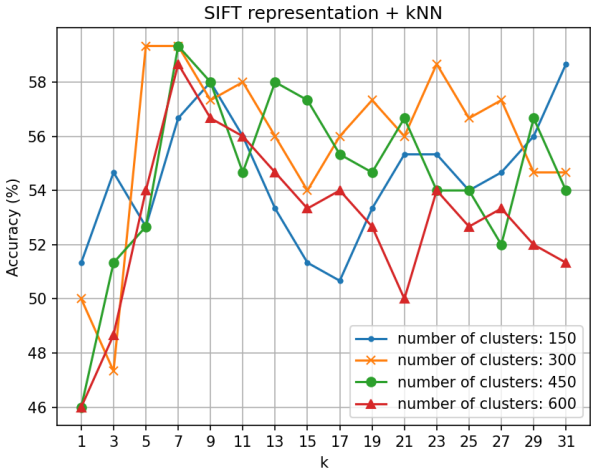

Figure 2: Comparison of SIFT representations with kNN

3.3 Bag of SIFT representation with linear SVM classifier

Here, we try four different numbers of clusters. For each case, we also try various kinds of C for SVM classifier. Figure 3 is the experiment results. The highest accuracy 68.67% occurs when number of clusters is 450 and `C` is 1000.

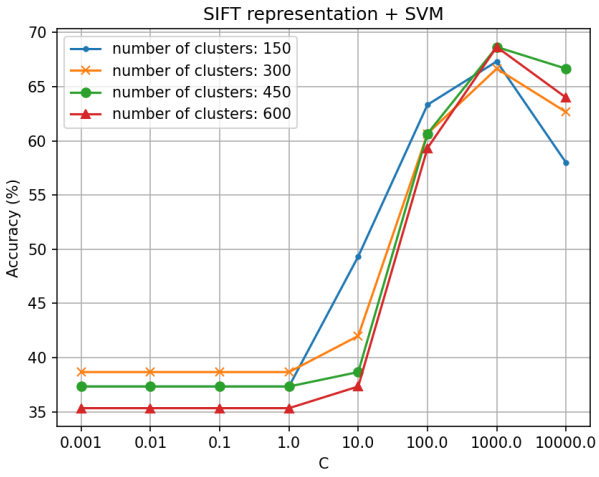

Figure 3: Comparison of SIFT representations with SVM

3.4 Convolutional neural network (CNN)

We conduct the large scale experiment with ResNet and its variants. The experiment results is shown in Table 1. We discover that fine-tuned ResNet34 has the best performance. For model trained form scratch, the WideResNet101 has the best performance. Overall, all fine-tuned models with ImageNet pretrained weight have a least 85% accuracy.

	ResNet18	ResNet34	ResNet50	ResNet101	ResNeXt50	ResNeXt101	WideResNet50	WideResNet101
Scratch	0.7400	0.7467	0.7133	0.7267	0.6933	0.7000	0.7333	0.7533
Pretrained	0.8533	**0.8867**	0.8667	0.8800	0.8733	0.8733	0.8733	0.8733
Acc. Gain	15.31%	18.75%	21.51%	21.10%	21.96%	**24.76 %**	19.09%	15.93%

Table 1: The test accuracy of different CNN model and the accuracy performance gain between training from pretrained and scratch.

4 Discussion

Generally, among all classification methods, we can observe that the performance of tiny representation is the lowest, while the SIFT representation with SVM classifier has the middle accuracy and the CNN classifier has the highest accuracy.

For tiny representation, when we normalize the training data before training, there are about 5% gain in accuracy compare to without normalization.

For two tasks using bag of SIFT representations, we discover that the parameters of SIFT descriptor play an important role for computing high quality key points and their descriptors. Thus, we spend some times tuning those parameters and the final decision of `SIFT_create` configuration is as following.

- `nOctaveLayers`: 5
- `contrastThreshold`: 0.01
- `edgeThreshold`: 80
- `sigma`: 0.6

For SVM classifier, we notice that the accuracy is higher when we set a larger `C`. The reason is that, for the larger `C`, the model tends to choose a hyperplane with smaller margin. Thus, the number of misclassified cases might decrease.

For CNN classifier, we found that ImageNet pretrained models provide better representations. As Table 1 shown, all the fine-tuned models have a least 15% accuracy performance gain compare to its training from scratch version.

5 Conclusion

Through the analysis of experiment results, we discover that using the deep learning model will obtain the best accuracy. There are still some important reasons that affect the performance such as data normalization, the number of clusters, parameter of SIFT and the `C` of SVM classifier. In conclusion, this experience will inspire us for the way of classification research in the future.

References

[1] Z. Min-Ling &Z. Zhi-Hua (2007) ML-KNN: A lazy learning approach to multi-label learning *Pattern recognition*

[2] K. Yan &S. Rahu (2004) PCA-SIFT: A more distinctive representation for local image descriptors *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*

[3] S. Christian &L. Ivan &C. Barbara (2004) Recognizing human actions: a local SVM approach *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*

[4] Z. Sergey &K. Nikos (2016) Wide residual networks *arXiv preprint arXiv:1605.07146*