Image Recognition using k-NN

Zhang Huakang
`mc35095@um.edu.mo`

Abstract

In this project, we implement a k-NN classifier to recognize the images in CIFAR-10 dataset. We use different image processing methods to preprocess the dataset, and use different distance metrics to test the performance of the classifier. We also implement a k-NN with PCA classifier, SVM and NN, to compare the performance of the k-NN.

1 Introduction

In this porject, we are going to implement a k-NN classifier to recognize the images. The dataset we use is CIFAR-10, which contains 60000 images in 10 classes. The images are 32x32 RGB images. We will use the training set to train the classifier and use the test set to test the classifier. The classifier will predict the class of the test images and compare the prediction with the ground truth to calculate the accuracy.

Not only k-NN classifier, we will also implement a k-NN with PCA classifier, SVM, and NN, to compare the performance of the k-NN.

We use Python to implement the project, and the code is available at Github[1]

2 Background

2.1 Image Recognition

Image recognition is the ability of AI to detect objects, places, people, writing and actions in images, which has been a popular topic in the field of computer vision. There are many datasets for image recognition, such as CIFAR-10, CIFAR-100, MNIST, etc. In this project, we will use CIFAR-10 dataset.

2.2 CIFAR-10

CIFAR-10[2] is a dataset of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

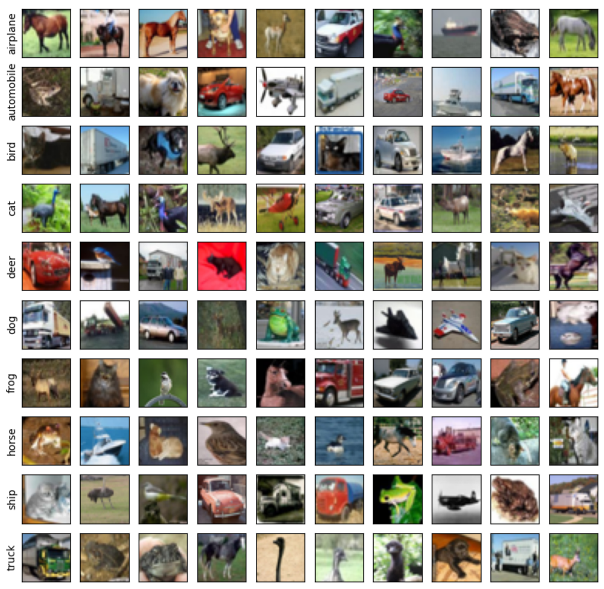

Figure 1: CIFAR-10

2.3 scikit-learn(?)

scikit-learn[3] is a free software machine learning library for the Python programming language. All algorithm used in this project are implemented by scikit-learn.

But this library has some limitations, such as it does not support GPU which means it is not suitable for large-scale machine learning.

[1] `https://github.com/BoxMars/CISC704-HW/tree/main/HW1`

[2] `https://www.cs.toronto.edu/~kriz/cifar.html`

[3] `https://scikit-learn.org/stable/`

2.4 cuML

cuML[4] cuML is a suite of fast, GPU-accelerated machine learning algorithms designed for data science and analytical tasks whose API mirrors that of scikit-learn. cuML enables data scientists, researchers, and software engineers to run traditional tabular ML tasks on GPUs without going into the details of CUDA programming.

In this project, we use cuML to implement the k-NN classifier, SVM and PCA, which is much faster than scikit-learn.

3 Method

3.1 Nearest Neighbors & k-Nearest Neighbors

Nearest Neighbors (NN) is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions), while k-Nearest Neighbors (k-NN) is a variant of NN, which is a non-parametric method used for classification and regression and is a voting algorithm based on the k closest neighbors, when $k = 1$, it is the NN algorithm.

3.2 Principal Component Analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

3.3 Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is the number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

[4] https://docs.rapids.ai/api/cuml/stable/

3.4 Image Processing

The images in CIFAR-10 are 32x32 RGB images, which means each image has 32x32x3=3072 features. When we use k-NN to classify the images, the dimension of the feature space is too large, which will cause the curse of dimensionality. So we need to reduce the dimension of the feature space.

3.5 Gary Scale

The first method we use is to convert the RGB images to gray scale images. The gray scale images only have one channel, which means the dimension of the feature space is reduced from 3072 to 1024. Fig 2 shows the gray scale images of CIFAR-10.

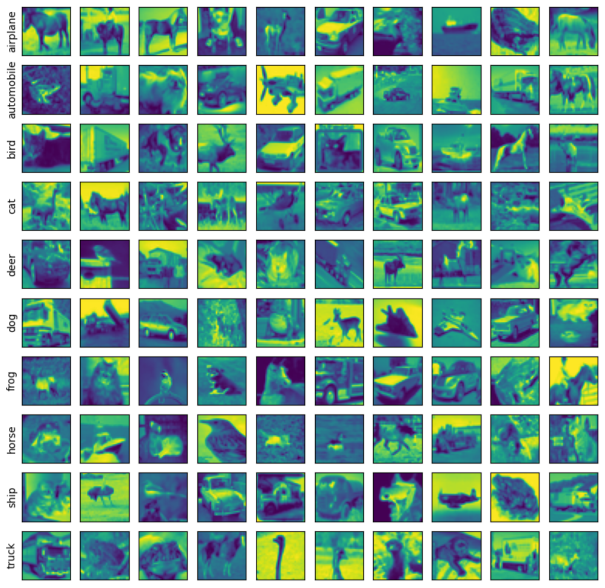

Figure 2: Gray Scale

3.6 Binary Image

The second method we use is to convert the gray images to binary images. The binary images only have two values, 0 and 1. Fig 3 shows the binary images of CIFAR-10.

3.7 HSV

The third method we use is to convert the RGB images to HSV images. HSV is a color model that describes colors (hue or tint) in terms of their shade (saturation or amount of gray) and their brightness value. The HSV images have three channels. Fig 4 shows the HSV images of CIFAR-10.

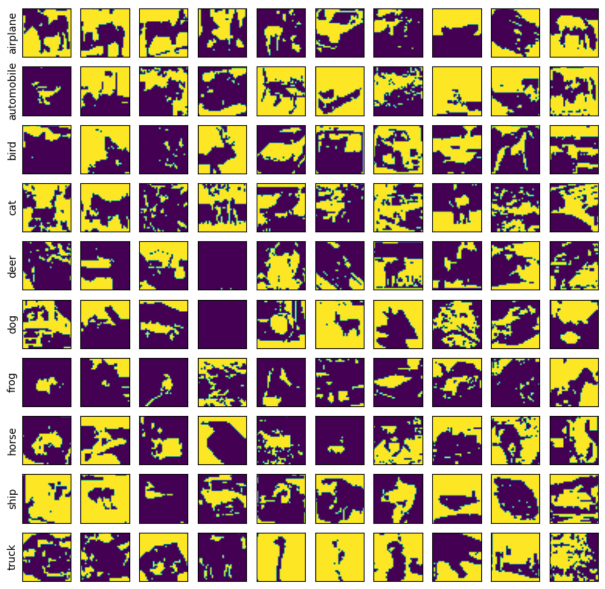

Figure 3: Binary Image

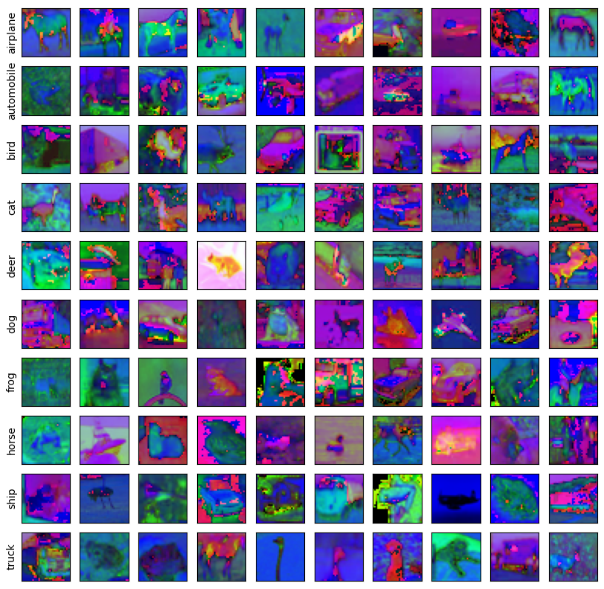

Figure 4: HSV

3.8 Edge Detection

In this project, we use `skimage.feature.canny` [5] to detect the edges of the images. The edge detection algorithm will return a binary image. Fig 5 shows the edge images of CIFAR-10.

3.9 Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of canny, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy. Fig 6 shows the HOG images of CIFAR-10.

4 Experiments

4.1 Download Dataset

CIFAR-10 dataset is available at `https://www.cs.toronto.edu/~kriz/cifar.html`. We can download the dataset from the website using `requests` python library and unzip it using `tarfile` library.

The data file of CIFAR-10 is a binary file, which can be read using `pickle` library.

4.2 Preprocess Dataset

We use the APIs provided by scikit-learn to preprocess the dataset. The first step is to convert the RGB images to gray scale images, binary images, HSV images, edge images and HOG images.

The second step is shuffle the dataset bescause the dataset is ordered by class and wo noticed the accuracy of unshuffled dataset is higher than shuffled dataset. Before shuffling the dataset, we need to convert the dataset to numpy array, and set the random seed to 0 to make sure the result is reproducible.

4.3 k-NN Classifier on GPU

We use cuML to implement the k-NN classifier on GPU.

```python
from cuml.neighbors import NearestNeighbors

classifiers = [
    ('NN with L1 distance', NearestNeighbors(n_neighbors=1, verbose=True, metric='l1')),
    ('NN with L2 distance', NearestNeighbors(n_neighbors=1, verbose=True, metric='l2')),
    ('NN with cosine', NearestNeighbors(n_neighbors=1, verbose=True, metric='cosine')),
]
```

Listing 1: k-NN Classifier on GPU

When `n_neighbors` is 1, it is the NN classifier. We use three different distance metrics, L1 distance, L2 distance and cosine distance and test the performance of different `n_neighbors` from 1 to 20. The reslut is shown in Fig 7. From the result, we can see that the accuracy of NN with L1 distance is higher than L2 distance and cosine distance, and

[5] `https://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.canny`

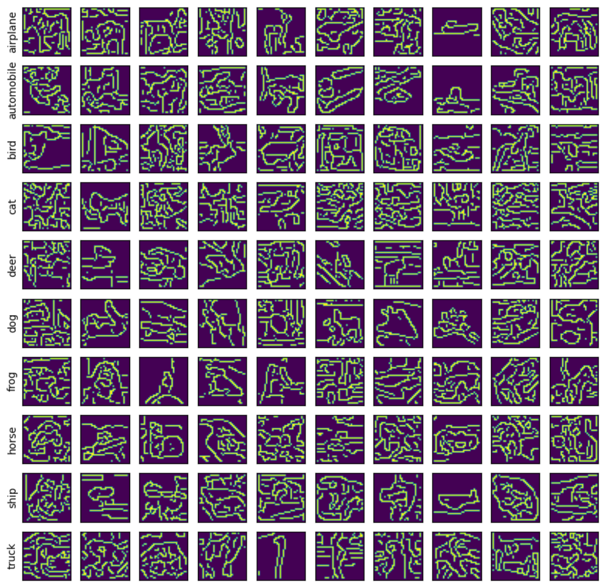

Figure 5: Edge Detection

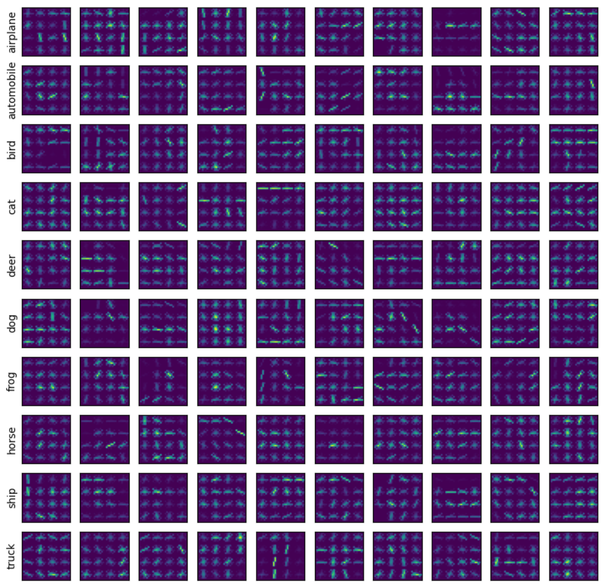

Figure 6: Histogram of Oriented Gradients

the image with hog feature has the highest accuracy, and the second is the image with gray scale feature.

The accuracy of the classifier will increase as the number of neighbors increases. When the number of neighbors is larger than 5, the accuracy of the classifier will not increase significantly.

```
def predict(classifiers, train_imgs,
    train_labels, test_imgs):
  train_imgs = train_imgs.reshape
    ((50000, -1))
  test_imgs = test_imgs.reshape((10000,
    -1))

  # shuffle data
  np.random.seed(0)
  np.random.shuffle(train_imgs)
  np.random.seed(0)
  np.random.shuffle(train_labels)
  np.random.seed(0)
  np.random.shuffle(test_imgs)
  np.random.seed(0)
  np.random.shuffle(test_labels)


  # combination of train img and label
  train = np.hstack((train_imgs,
    train_labels))
  test = np.hstack((test_imgs,
    test_labels))

  accs={}
  for name, model in classifiers:
      accs[name] = []
      model.fit(train)
      for i in range(1,21):
          distance, indices = model.
    kneighbors(test, n_neighbors=i,
    return_distance=True)

          pred = train_labels[indices]
          # find the most common label
          pred = np.apply_along_axis(
    lambda x: np.argmax(np.bincount(x)),
     axis=1, arr=pred)

          pred = pred.reshape((10000, 1)
    )
          acc = accuracy_score(
    test_labels, pred)
          # print('Accuracy of {} - k:
    {}: {}'.format(name, i, acc))
          accs[name].append(acc)
  return accs
```

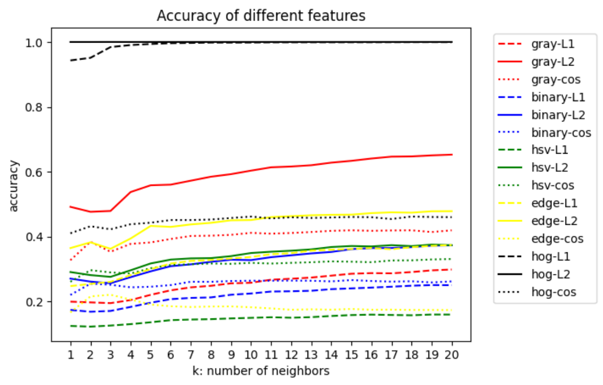

Figure 7: Accuracy of k-NN with different number of neighbors

Listing 2: k-NN Classifier on GPU with different neighbors

4.4 PCA

We use cuML to implement the PCA algorithm on GPU, and use the PCA to reduce the dimension of the feature space. The reslut accuracy of k-NN with PCA with different number of components is

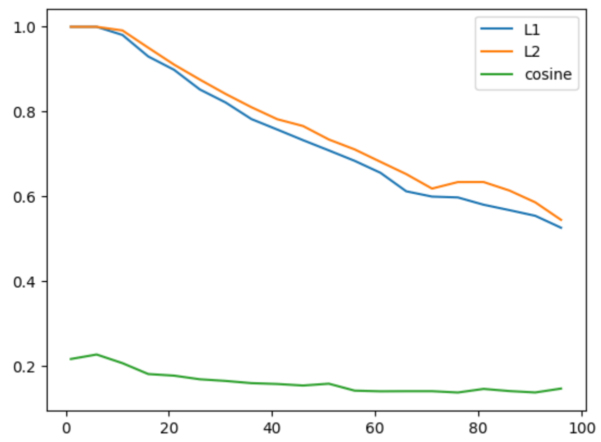

Figure 8: Accuracy of k-NN with PCA with different number of components

shown in Fig 8. From the result, we can see that the accuracy of the classifier will increase as the number of components increases. When the number of components is larger than 10, the accuracy of the classifier will not increase significantly.

```python
from cuml import PCA

pca = PCA(whiten=True, n_components=1)
pca.fit(train_gray.reshape((50000, -1)).
    astype(np.float32))

train_pca = pca.transform(train_gray.
    reshape((50000, -1)).astype(np.
    float32))
test_pca = pca.transform(test_gray.
    reshape((10000, -1)).astype(np.
    float32))
```

Listing 3: PCA

We test the performance of three k-NN classifier with PCA on GPU with different `n_components` from 1 to 20, the reslut is shown in Fig 8.

4.5 SVM

We use cuML to implement the SVM algorithm on GPU. The reslut accuracy of SVM with different kernel is shown in Fig 9.

The accuracy of SVM classifier is not very high, and the accuracy of Linear SVM with L2 is hightest.

```python
from cuml import LinearSVC, SVC
classifiers = [
    ('Linear SVM l1', LinearSVC(penalty=
    'l1', loss='squared_hinge', tol=1e
    -4, C=1.0)),
    ('Linear SVM l2', LinearSVC(penalty=
    'l2', loss='squared_hinge', tol=1e
    -4, C=1.0)),
    ('SVM rbf', SVC(C=1.0, kernel='rbf',
     gamma='auto', tol=1e-3)),
    ('SVM sigmoid', SVC(C=1.0, kernel='
    sigmoid', gamma='auto', tol=1e-3)),
]
```

Listing 4: SVM

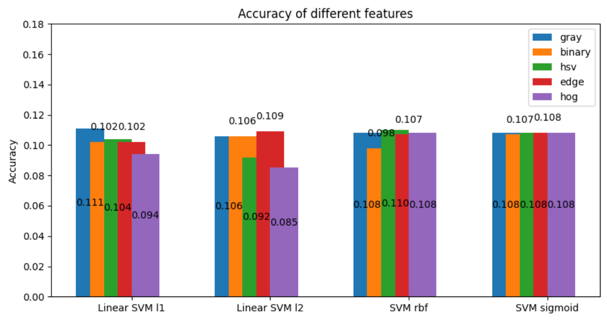

Figure 9: Accuracy of SVM with different kernel

5 Conclusion

In this project, we implement a k-NN classifier to recognize the images in CIFAR-10 dataset. We use different image processing methods to preprocess the dataset, and use different distance metrics to test the performance of the classifier. We also implement a k-NN with PCA classifier, SVM and NN, to compare the performance of the k-NN.

References