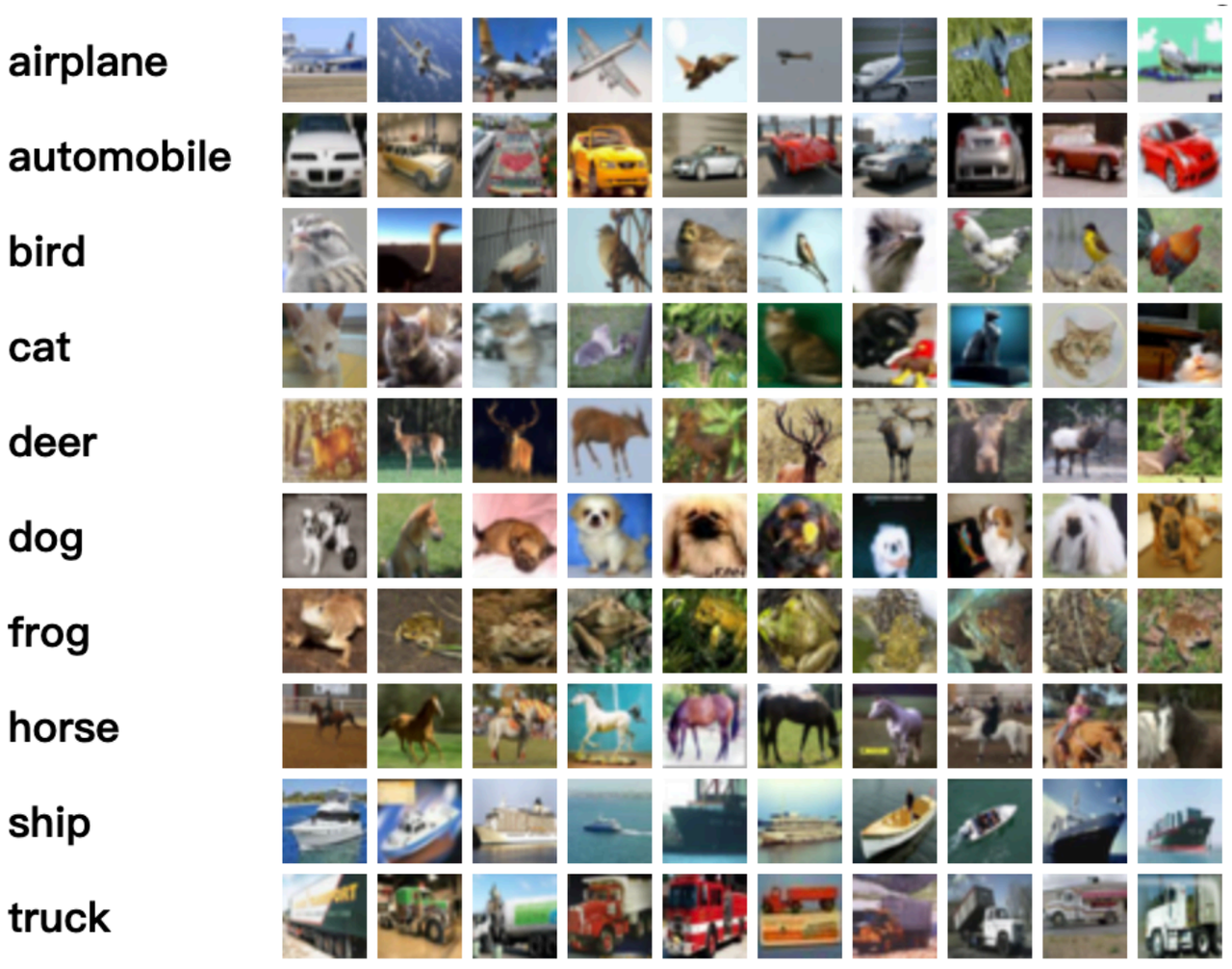
数字图像处理课设

Project #1 Image Classification

组员： 杨澍生 周展科 杨阳 唐彬

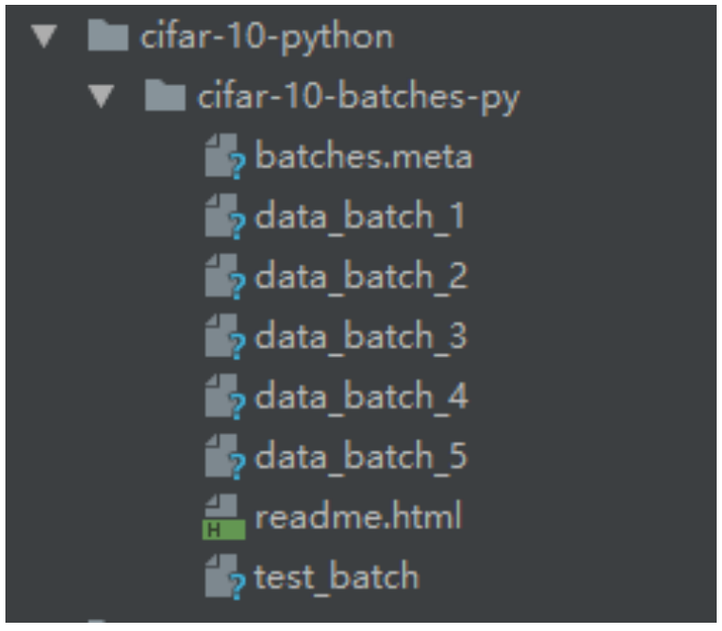
汇报人：唐彬

CIFAR-10 Dataset

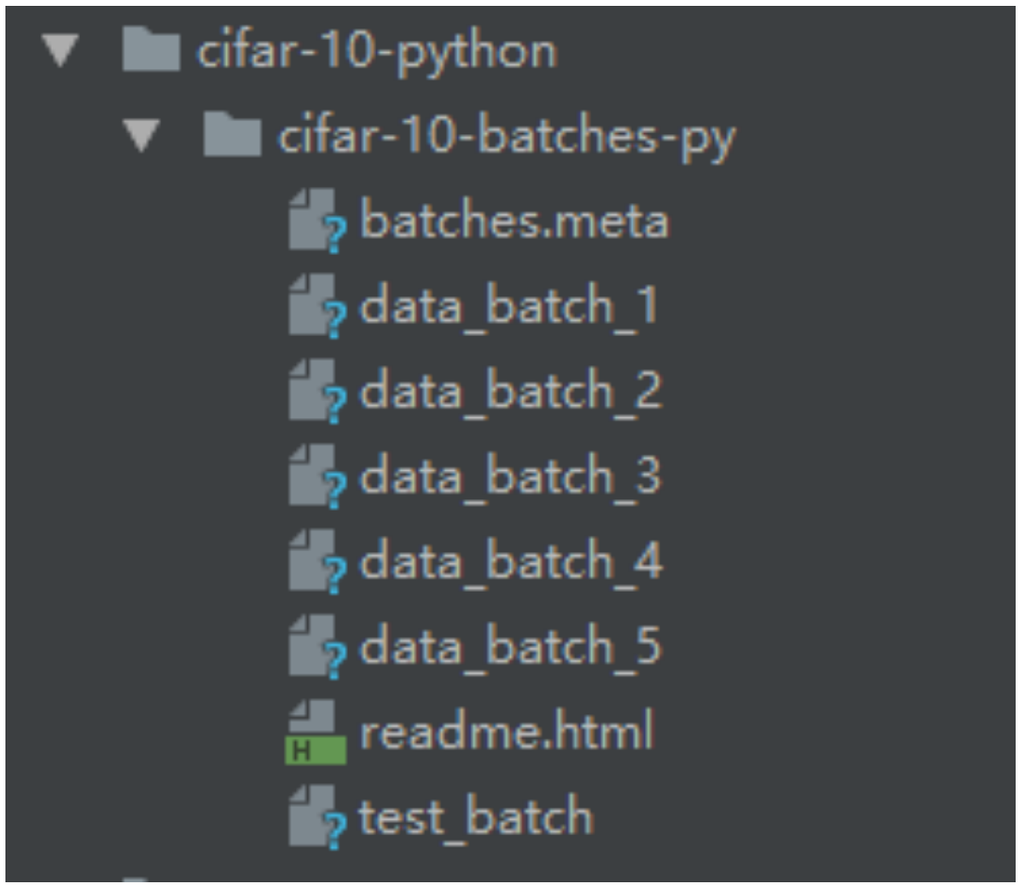


- CIFAR-10数据集由10类32x32的彩色图片组成，一共包含60000张图片，其中50000张图片作为训练集，10000张图片作为测试集。

- CIFAR-10数据集被划分成了5个训练的batch和1个测试的batch，解压之后的目录结构如下：



CIFAR-10 Dataset



- batch是一个python的dict结构。数据集官网提供了python3读取CIFAR-10的方式，以下函数可以将数据集转化为字典类型

```
def unpickle(file: str) -> dict:
    '''
    :param file: path to one cifar10 item
    :return: dict of one cifar10 item
    '''
    with open(file, 'rb') as fo:
        d = pickle.load(fo, encoding='bytes')
    return d
```

- 得到的字典中，我们主要关心’data’和’labels’两个关键key。’data’中每一张图片是以被展开的形式存储（即一张32x32的3通道图片被展开成了3072长度的list），每一个数据的格式为uint8，前1024个数据表示红色通道，接下来的1024个数据表示绿色通道，最后的1024个通道表示蓝色通道。

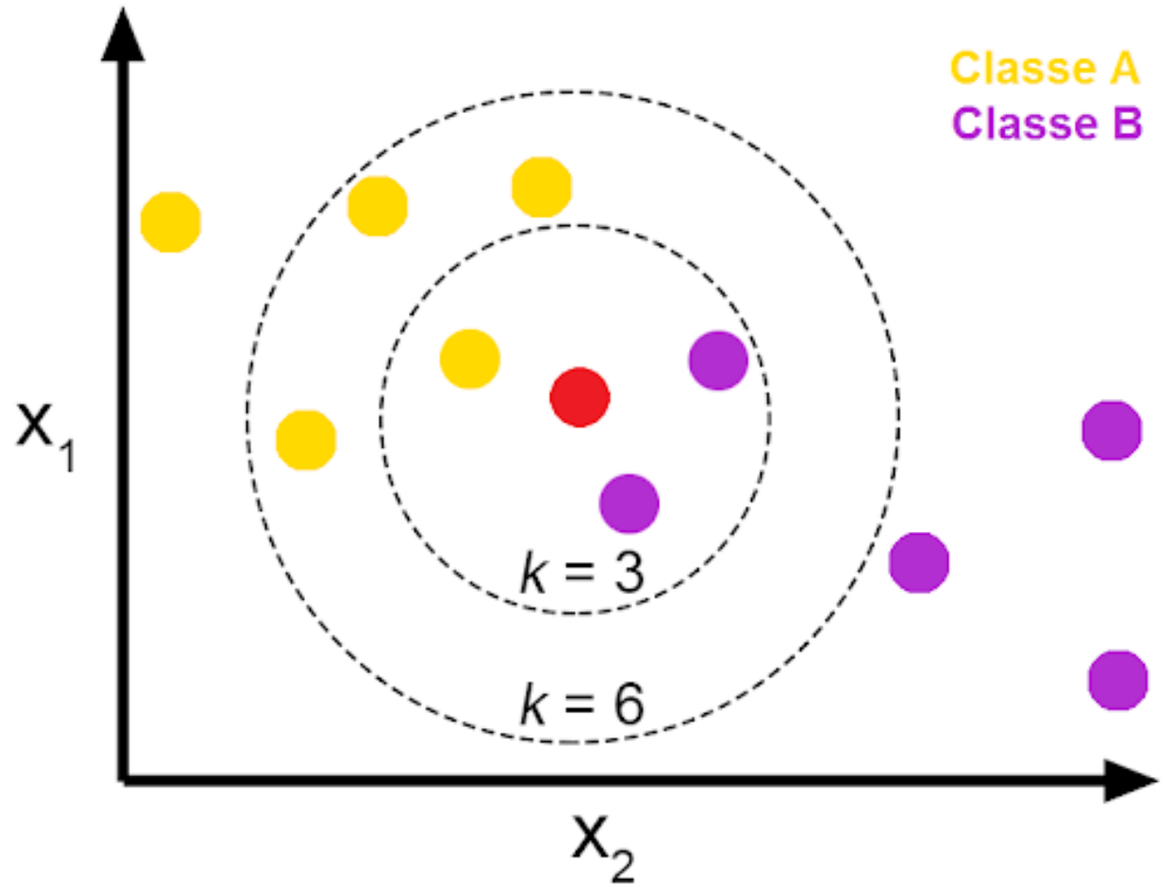
k-Nearest Neighbors

1.计算待分类的测试数据的特征向量与已知类别的训练样本的特征向量的距离。

2.将距离从小到大排序。

3.去前k个值并统计k个值中每个类别出现的频数。

4.频数最大的训练样本的类别即为测试样本的预测类别。



KNN在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别，因此该算法的结果很大程度取决于k的选择。如上图中k=3和k=6时会得到不同的结果。此次实验也正是通过枚举不同的k值运行KNN算法，从而选择出最优的k值。

距离函数

另一个对KNN预测准确些影响较大的因素是特征向量间的距离计算公式，计算距离有很多种方法，最简单的方法就是逐个像素比较，最后将差异值全部加起来。我们采用了三种常用的距离计算方法：L1距离、L2 距离、cosine距离。

L1距离：

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

```
if method=='l1':
    # for L1 distance
    data_num = x.shape[0]
    for i in tqdm(range(data_num)):
        distance.append(np.sum(np.abs(self.x - x[i]), axis = 1))
    distance = np.array(distance)
```

L2距离：

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

```
if method=='l2':
    # for L2 distance
    distance = -2 * x.dot(self.x.T) \
               + np.sum(np.square(x), axis=1, keepdims=1) \
               + np.sum(np.square(self.x), axis=1).T
```

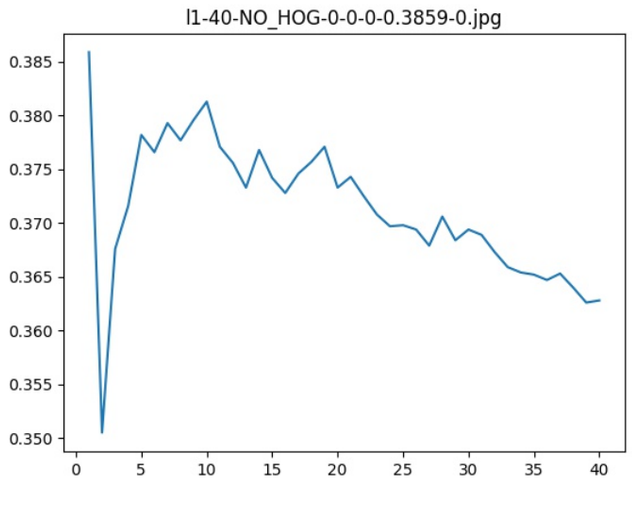
cosine距离：

$$d = 1 - sim(X, Y) = 1 - cos\theta = 1 - \frac{\vec{x} \cdot \vec{y}}{\|x\| \cdot \|y\|}$$

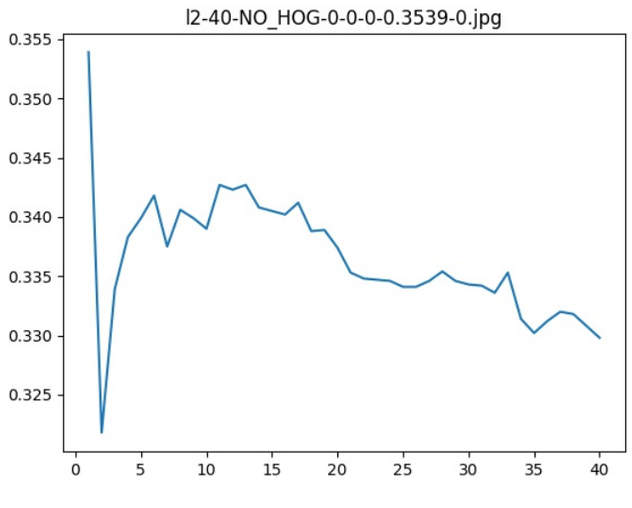
```
if method=='cosine':
    # for cosine distance
    distance = 1 - x.dot(self.x.T) \
               / np.linalg.norm(x, axis=1, keepdims=1) \
               / np.linalg.norm(self.x, axis=1).T
```

实验结果

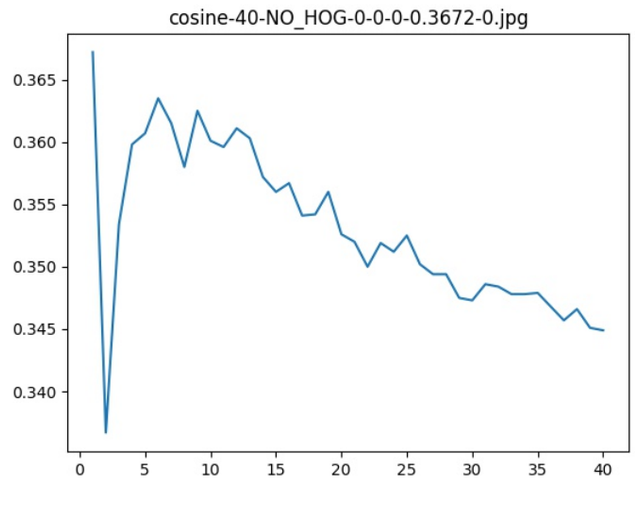
L1距离



L2距离



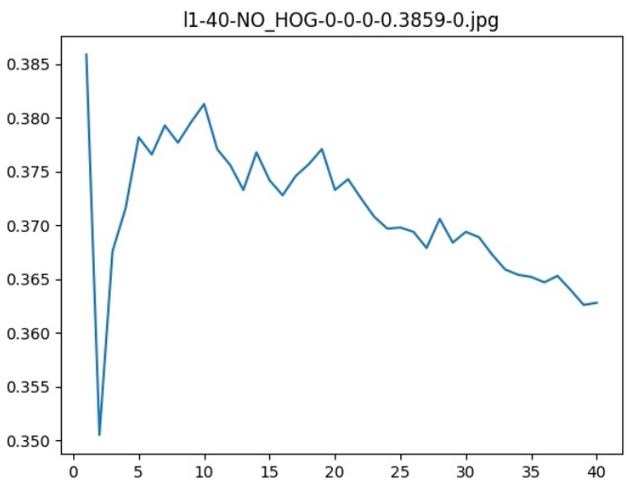
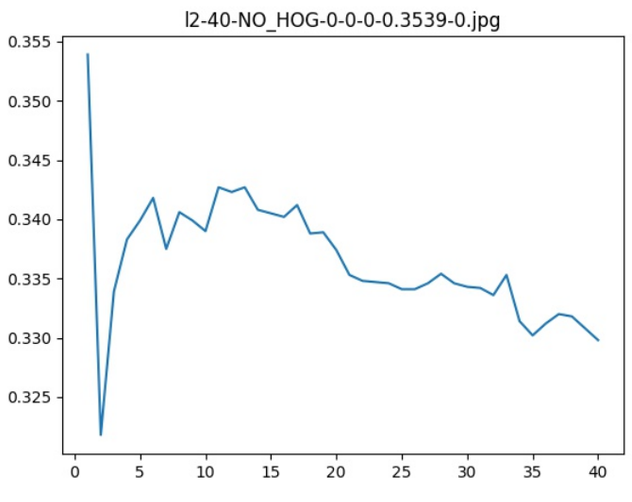
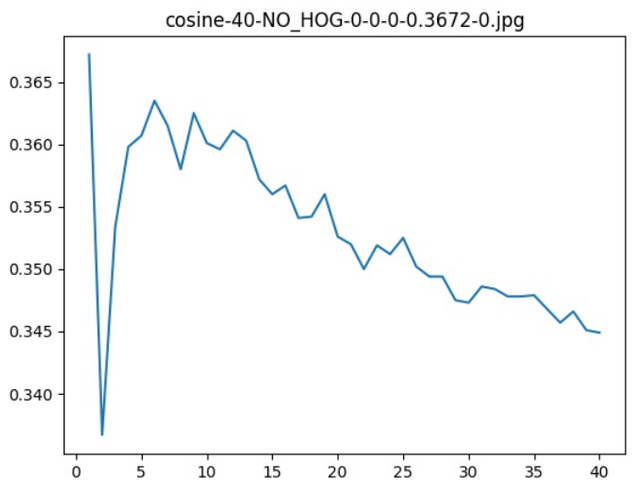
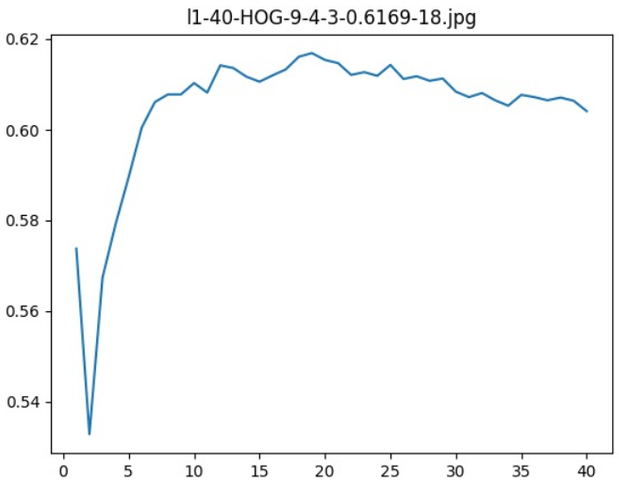
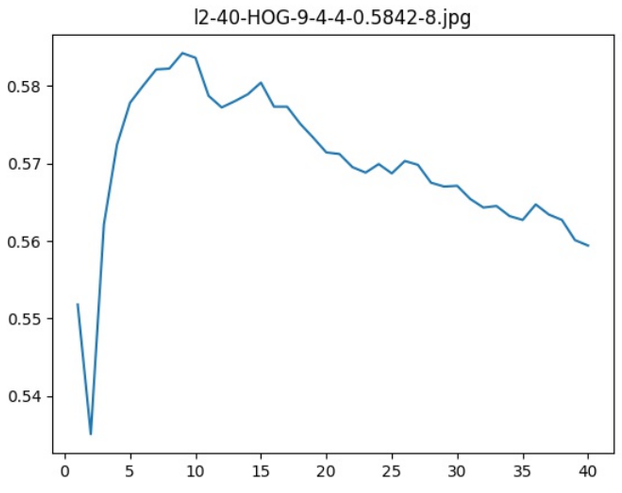
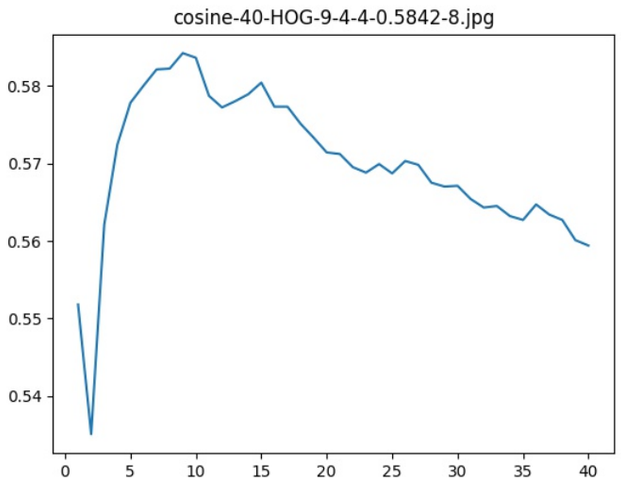
cosine距离

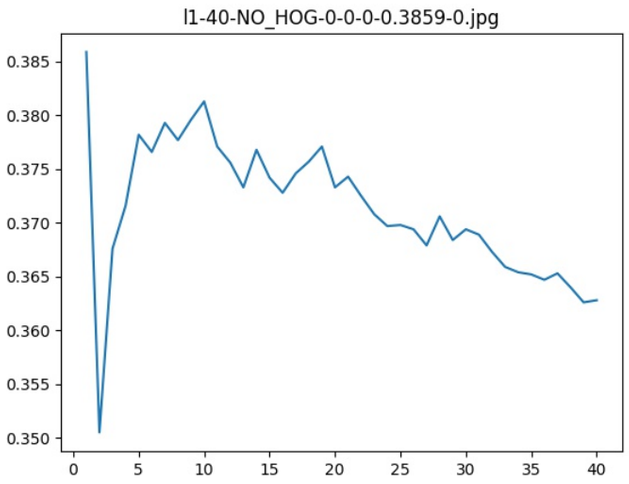
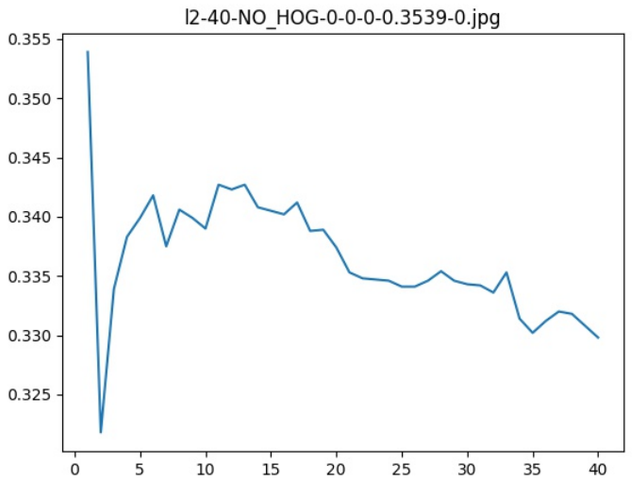
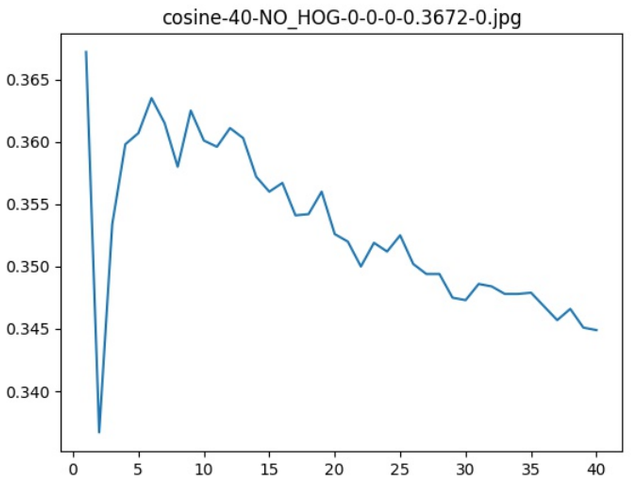
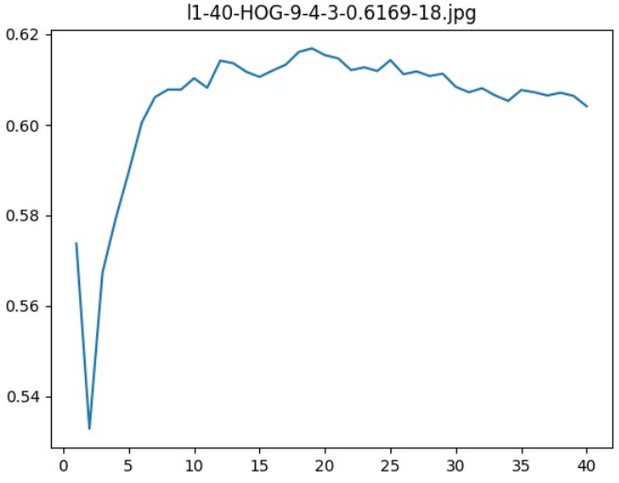
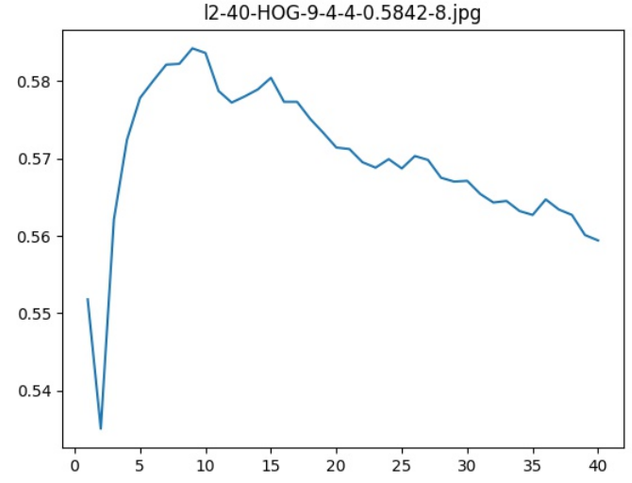
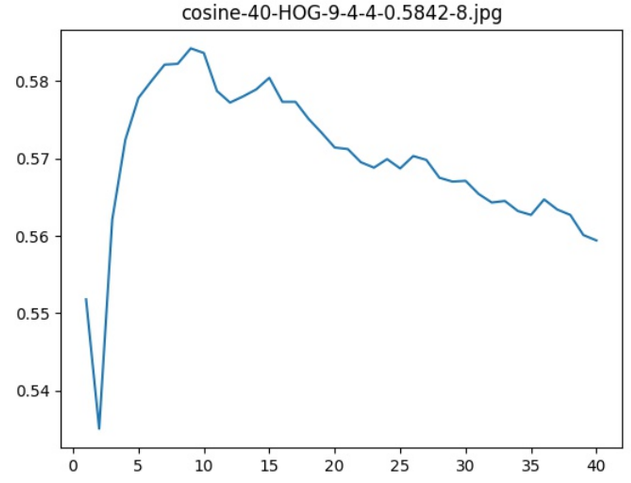


从上图看出，distance metric选择L1距离比 cosine和L2好。我们认为，这是因为L2距离很容易受到噪声的干扰，有一点点的loss，在L2距离函数的作用下就有可能被放大，从而对模型的判断造成影响。cosine和L2类似。

在此基础上，我们提取了图像的HOG特征进行KNN分类，HOG的思想是在边缘具体位置未知的情况下，边缘方向的分布也可以很好的表示目标的外形轮廓。

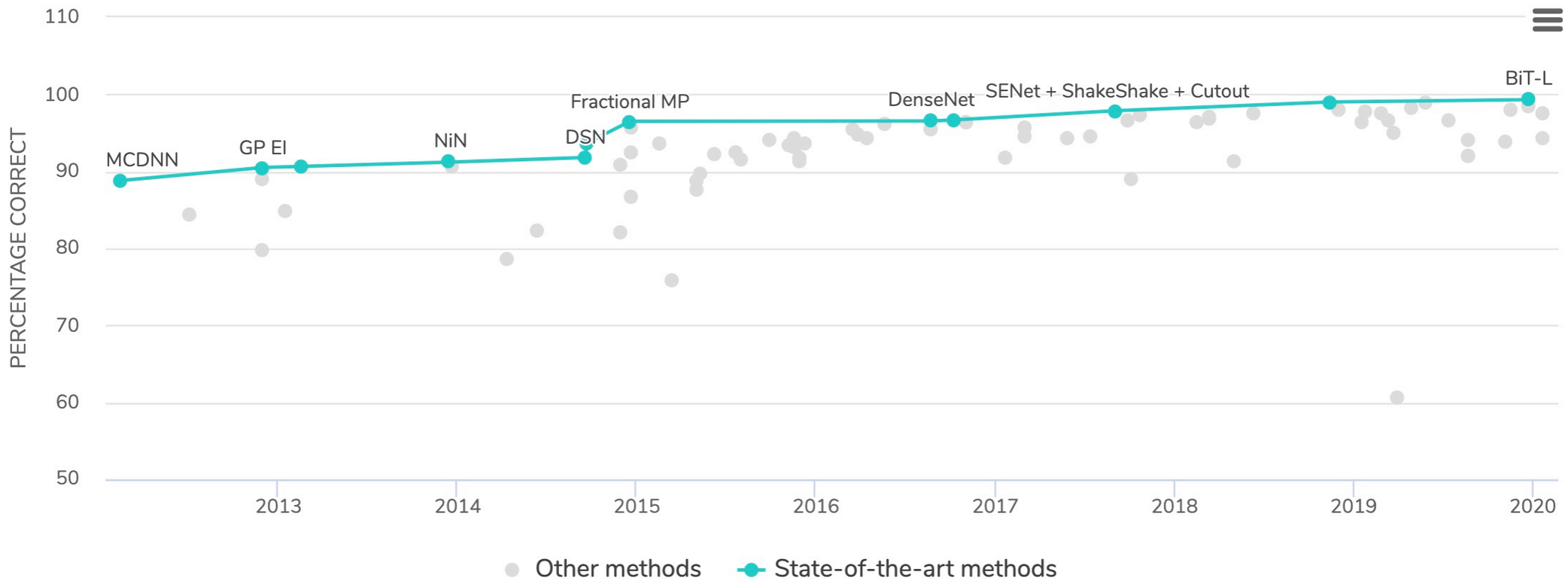
实验结果

	L1距离	L2距离	cosine距离
不使用HOG特征提取			
使用HOG特征提取			

	L1距离	L2距离	cosine距离
不使用HOG特征提取			
使用HOG特征提取			

从以上的结果可知，使用raw pixels作为图像特征，分类的准确率约为36%，k=1时最佳；提取HOG特征可以提高分类的精度，约为60%，对于L1距离，K=18时最佳，对于L2、cosine距离，k=8时最佳；使用L1距离在CIFAR-10数据集上的预测效果较好。

Image Classification on CIFAR-10



从上图可以看出，对于CIFAR-10数据集的分类问题，目前大部分的算法预测正确率都在80%以上，KNN在该数据集上的表现相比深度学习算法还不够理想。

KNN算法本身非常简单，在训练过程中没有任何计算。也因此其包含了以下缺点：

- 整个训练过程需要将所有的训练样本及其label存储起来，空间成本较大。
- 测试过程中，每个测试样本都需要与所有的训练样本进行比较，即复杂度为O(n)，计算距离的代价很高。

尝试SVM

```
class SVM:
    def __init__(self, loss="vectorized") -> None:
        # param initialize
        self.W = None

        assert loss in ["vectorized", "naive", "softmax"]
        if loss == "vectorized":
            self.loss = self.svm_loss_vectorized
        elif loss == "naive":
            self.loss = self.svm_loss_naive
        elif loss == "softmax":
            self.loss = self.softmax

def train(self, x, y, reg=1e-5, learning_rate=1e-3, num_iters=
    """
    功能：使用随机梯度下降法训练SVM

    输入：
    -x:(numpy array)训练样本（N,D）
    -y:(numpy array)训练样本标签(N,)
    -reg:(float)正则化强度
    -learning_rate:(float)进行权重更新的学习率
    -num_iters:(int)优化的迭代次数
    -batch_size:(int)随机梯度下降法每次使用的梯度大小
    -verbose:(bool)取True时，打印输出loss的变化过程

    输出：-history_loss:(list)存储每次迭代后的loss值
    """
```

SVM result

Classifier	Configuration	Accuracy
SVM	lr: 1e-7	0.31
SVM	lr: 1e-7 + Hog	0.33

我们尝试使用二分类器SVM来完成多分类问题，虽然说SVM与HOG搭配会有较好的效果，但是在我们的实验中，加入了HOG之后精确度提升并不太明显。
由于超参数没有调到一个较好的值，SVM的效果并不理想，这一块的工作是后续还有待改善的。

问题与思考

1. KNN学习过程太慢了

我们的代码同时支持CPU和GPU(cupy)两种模式，在1080Ti上运行需要的时间很少。同时还要加上一些小trick:

- L2距离和cosine的距离的函数式完成可以在numpy的层面上优化
- 计算好距离矩阵之后直接排序，保留结果，之后search k时就不用重复计算

```
if args.cuda:
    import cupy as np
else:
    import numpy as np
```

```
1    finished   /tmp/ts-out.p745ry   0        198.39/139.73/59.48 python main.py --cuda=1
2    finished   /tmp/ts-out.jzxI5N   0        94.32/84.79/10.32 python main.py --cuda=1 -
3    finished   /tmp/ts-out.gf6D7i   0        95.66/85.79/10.58 python main.py --cuda=1 -
4    finished   /tmp/ts-out.nOgTO6   0        119.98/103.42/17.18 python main.py --cuda=1
5    finished   /tmp/ts-out.UxsZnV   0        99.36/93.38/6.90 python main.py --cuda=1 --
6    finished   /tmp/ts-out.u8GRPO   0        99.05/93.03/6.87 python main.py --cuda=1 --
7    finished   /tmp/ts-out.DO8uuh   0        127.39/110.62/17.52 python main.py --cuda=1
8    finished   /tmp/ts-out.b5bYU0   0        105.37/99.26/6.85 python main.py --cuda=1 -
9    finished   /tmp/ts-out.oVVXO9   0        105.20/99.26/6.70 python main.py --cuda=1 -
10   finished   /tmp/ts-out.3Jfiuf   0        269.91/199.61/71.13 python main.py --cuda=1
11   finished   /tmp/ts-out.M2lzHF   0        139.77/131.82/8.96 python main.py --cuda=1
12   finished   /tmp/ts-out.WoWWdw   0        138.41/130.41/8.83 python main.py --cuda=1
13   finished   /tmp/ts-out.xxWoW1   0        254.10/195.08/59.87 python main.py --cuda=1
14   finished   /tmp/ts-out.MlN7BE   0        150.88/143.18/8.45 python main.py --cuda=1
15   finished   /tmp/ts-out.2Fekck   0        149.75/142.18/8.48 python main.py --cuda=1
16   finished   /tmp/ts-out.RixruP   0        230.33/190.20/40.85 python main.py --cuda=1
17   finished   /tmp/ts-out.T7gl5a   0        161.52/154.46/7.84 python main.py --cuda=1
18   finished   /tmp/ts-out.n9ENr6   0        160.46/153.28/7.94 python main.py --cuda=1
```

2. 数据集有坑

图像数据的存储是用的uint8格式，而直接用读取出来的数据做KNN L1/L2算法，会导致严重的精度下降。因为在uint8中，不存在负数，比如1-2得到的是255，L1-KNN实验中，使用uint8数据会比使用int32数据在精度上低10多个百分点。

问题与思考

3. K的最优取值具备怎么样的特征?

在实验的过程中，我们会发现，会出现三种情况：
- 使用10K的训练集，K的最优取值普遍在相对较高的地方
- 使用50K的训练集，K的最优取值都是1
- 使用50K的训练集和HOG方法，K的最优取值也分布在相对较高的地方

为什么会出现这种情况呢？ 这个时候需要我们回到KNN本身来。KNN中的NN是一种惰性算法，最近邻算法，训练的开销基本是0。KNN中的K可以理解成一种多模型的ensemble，一种投票机制。也正是这种机制导致了上述的问题。

在数据集不足时，数据在样本空间中的分布比较离散，不能表征整个样本空间，这种情况下模型的 ensemble是有助于模型效果提升的。 在数据集充足的情况下，数据集可以说是占满了整个样本空间，最优匹配的可信度会大幅度提高，在这种情 况下模型的ensemble不一定能带来效果的提升。 HOG方法可以理解为是一种特征的蒸馏、提纯（保留轮廓等显著特征），这可以理解为对数据做的一种聚 类(cluster)，做完聚类之后，同类数据的类间距得到缩小，异类数据之间的类间距得到扩大，在这种情况 下，模型的ensemble可以消除部分噪声所带来的影响，提高模型的精度。

问题与思考

4. HOG的三个参数的选择: orientations, pixels_per_cell(ppc), cells_per_block(cpb)

orientatoins使用8/9都可以，对于180度的角来说，分成8份或9份具有相同的区分度。因为我们的输入是(32, 32)，ppc还是不能设置的太大，又为了考虑可以被32整除，所以我们取4或者8。同时cbp不 要设置的太大，因为block之内的梯度是要做归一化的，而对一个大区域，比如整个图片的梯度做归一化，会使得 各个cell之间梯度信息被减弱，区域与区域之间的差异性减弱，特征相应也就少了。同时cbp也不能太小，因为梯 度归一化可以排除光照等外界条件的影响。所以我们在(2,3,4)中做搜索。我们的实验就是这么出来的。 同时，这几个参数的选择还有一点要注意，选择的参数会影响到之后输出的特征长度，不宜太小（太小的原因一般 是ppc和cpb的设置，相对于原图，太大，导致感受野就是整个图片，局部信息和局部的差异性被忽略），也不宜 太大（太大的原因是因为ppc和cpb的设置相对于原图，太小，感受野非常非常小，无法把握到图片整体的信息）