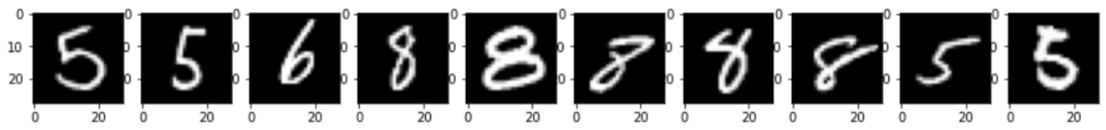
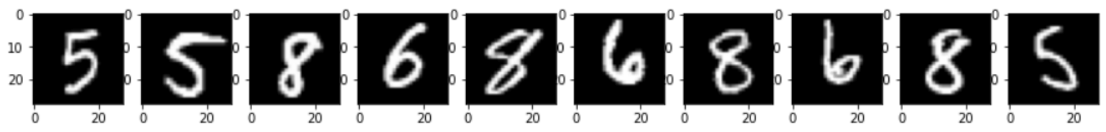
## 1.Dataset construct

In this part, for easy acessing, the pytorch API  torchvision.Datasets.MNIST is directly used. In training dataset, there are 60,000 images and there are 10,000 images in testing dataset. To construct mnist_568 dataset, the images corresponding label 5,6 or 8 will be selected. There are total 17,190 images in training dataset with labels 5,6 or 8 and 2,824 images in testing dataset with labels 5,6 or 8. And 15,000 images and 2,500 images will be respectively selected from training dataset and testing dataset randomly.

## 2.Data Preparation

The following figures shows randomly picking 10 images from the training set in the first line and another 10 images from the testing set in second line.





And print the corrpesonding labels are:

```
For the training dataset,the corresponding label are [5 5 6 8 8 8 8 8 5 5]
For the testing dataset,the corresponding label are [5 5 8 6 8 6 8 6 8 5]
```

In next step, the feature vectors are normalized such that they have unit $l_2$ norm. And the command is

```
import numpy.linalg as LA
norm_1 = LA.norm(x_train,ord=2,axis=1).reshape(-1,1)
x_train = x_train/norm_1

norm_2 = LA.norm(x_test,ord=2,axis=1).reshape(-1,1)
x_test = x_test/norm_2
```

## 3.Evaluation

Firstly, take a simple review of conclusion of Johnson-Lindenstrauss lemma.

**Lemma**: *A set of n points* $u_1, \cdots, u_n$ *in* $R^d$ *can be projected down to* $v_1, \cdots, v_n$ *in* $R^m$, $f: R^d \rightarrow R^m$ *such that all pairwise distances are preserved when* $m \geq \frac{4}{\epsilon^2 - \epsilon^3}\ln(n)$ *:*

$$(1-\epsilon)||u_i - u_j||^2 \leq ||f(u_i) - f(u_j)||^2 \leq (1+\epsilon)||u_i - u_j||^2$$

In this part,  Gaussian random projection will be used to project original data into a lower dimensional space. When constructing projecting matrix $R \in R^{d\times m}$, each element $R(i,j)$ are drawn from Gaussian distribution with mean 0 and variance 1. This sampling method can guarantee $E(||f(u)||^2) = ||u||^2$. when given projected matrix $R$, the new dataset is $\frac{1}{\sqrt{m}}RX$

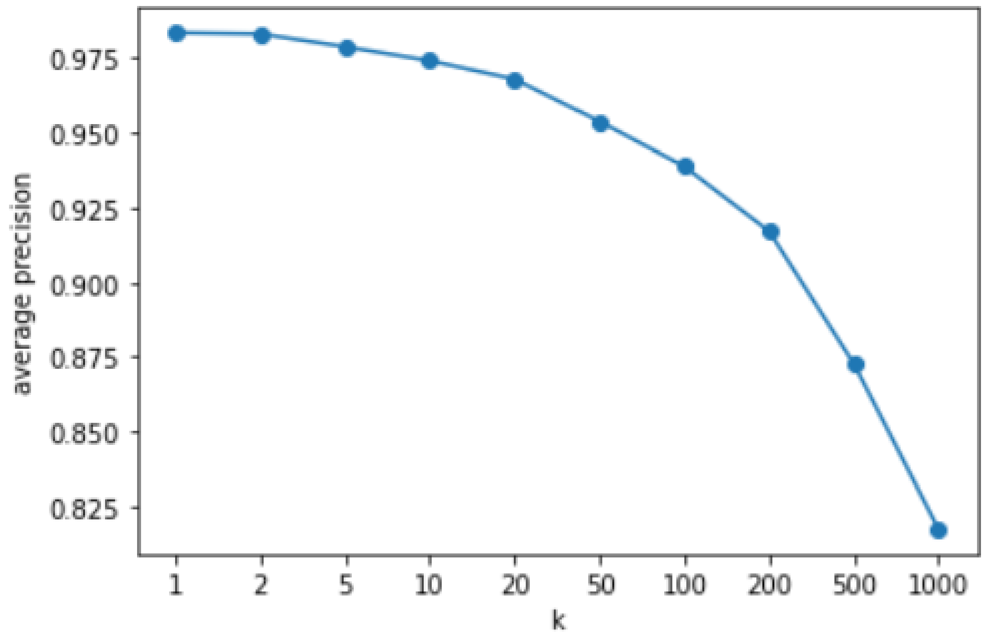
where $X$ is original dataset. In this expriment, we choose $\epsilon = 0.32$ and the correponding projected dimension $m$ is 561 after computation which is less than original dimension 784.

In implementation of $dist(X_{train}, X_{test})$, scipy.spatial.distance.cdist will be used for projected datasets. In implementation of $retrieve_k(i, M, k)$ ,np.argsort will be used to sort the distance between  data point pairs.

## 4.Retrieval performance.

In this part, we experiment different k values for $1, 2, 5, 10, 20, 50, 100, 200, 500, 1000$. The corresponding average  precision are:

```
[0.9836,0.9832,0.9788799999999994,0.9742000000000022,0.9680200000000001,
0.9538720000000067,0.938852,0.9171859999999921,0.8727976000000035,0.817490400000
0026]
```



And the corresponding running time are:

```
[18.635037183761597,15.524598598480225, 15.64721965789795, 15.976155042648315,
16.091359853744507, 15.485934495925903, 15.718036413192749, 16.05874276161194,
15.665780544281006,15.49856686592102]
```

From the figure, we can find that when k increases the average  query precision wil decrease. If the value is very large, the corresponding precision value may decrease by 15% that is very terrible. Therefore, controling k in a good interval can also keep the average query precision. Futhermore, we find that running time are not affected much by different k.