HM2

February 5, 2024

```
[ ]: import os
import numpy as np
import pandas as pd
from PIL import Image

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sn

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

import torch
import torch.nn as nn
from torch import optim
from torch.utils.data import Dataset, DataLoader

from torchvision import transforms
from torchvision import datasets
from torch.utils.data import random_split
```

1 Problem 1:

1.1 Download 4 Points

```
[ ]: !mkdir -p data
!mkdir -p data/train

!wget -O data/train/img_0.jpg -nc -q https://github.com/darksigma/
 ↪Fundamentals-of-Deep-Learning-Book/raw/master/ch05_implementing_nn_pytorch/
 ↪data/train/img_0.jpg
!wget -O data/train/img_1.jpg -nc -q https://github.com/darksigma/
 ↪Fundamentals-of-Deep-Learning-Book/raw/master/ch05_implementing_nn_pytorch/
 ↪data/train/img_1.jpg
!wget -O data/train/img_2.jpg -nc -q https://github.com/darksigma/
 ↪Fundamentals-of-Deep-Learning-Book/raw/master/ch05_implementing_nn_pytorch/
 ↪data/train/img_2.jpg
```

```
!wget -O data/train/img_3.jpg -nc -q https://github.com/darksigma/
 ↪Fundamentals-of-Deep-Learning-Book/raw/master/ch05_implementing_nn_pytorch/
 ↪data/train/img_3.jpg
!wget -O data/train/labels.npy -nc -q https://github.com/darksigma/
 ↪Fundamentals-of-Deep-Learning-Book/raw/master/ch05_implementing_nn_pytorch/
 ↪data/train/labels.npy
```

[ ]:
```
class ImageDataset(Dataset):
  def __init__(self, img_dir, label_file):
    super(ImageDataset, self).__init__()
    self.img_dir = img_dir
    self.labels = torch.tensor(np.load(label_file, allow_pickle=True))
    self.transforms = transforms.ToTensor()

  def __getitem__(self, idx):
    img_pth = os.path.join(self.img_dir, "img_{}.jpg".format(idx))
    img = Image.open(img_pth)
    img = self.transforms(img).flatten()
    label = self.labels[idx]
    return {"data":img, "label":label}

  def __len__(self):
    return len(self.labels)
```

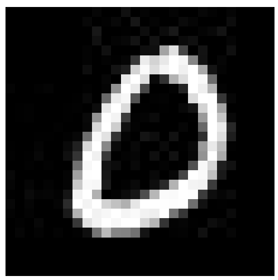
[ ]:
```
train_dataset = ImageDataset(img_dir='./data/train/',
                             label_file='./data/train/labels.npy')

train_loader = DataLoader(train_dataset,
                          batch_size=4,
                          shuffle=True)
```

[ ]:
```
for minibatch in train_loader:
  data, labels = minibatch['data'], minibatch['label']
  print(data)
  print(labels)
```

```
tensor([[0., 0., 0.,  …, 0., 0., 0.],
        [0., 0., 0.,  …, 0., 0., 0.],
        [0., 0., 0.,  …, 0., 0., 0.],
        [0., 0., 0.,  …, 0., 0., 0.]])
tensor([0, 2, 4, 1])
```

1.2 Test

[ ]:
```
ndata=data.view(4,28,28).detach().numpy()
scale = 5
im_data = ndata[0]
dpi = matplotlib.rcParams['figure.dpi']
height, width = im_data.shape
figsize = scale * width / float(dpi), scale * height / float(dpi)
fig = plt.figure(figsize=figsize)
ax = fig.add_axes([0, 0, 1, 1])
# Hide spines, ticks, etc.
ax.axis('off')
ax.imshow(im_data, vmin=0, vmax=1, cmap='gray')
plt.show()
ax.set(xlim=[0, width], ylim=[height, 0], aspect=1)
```



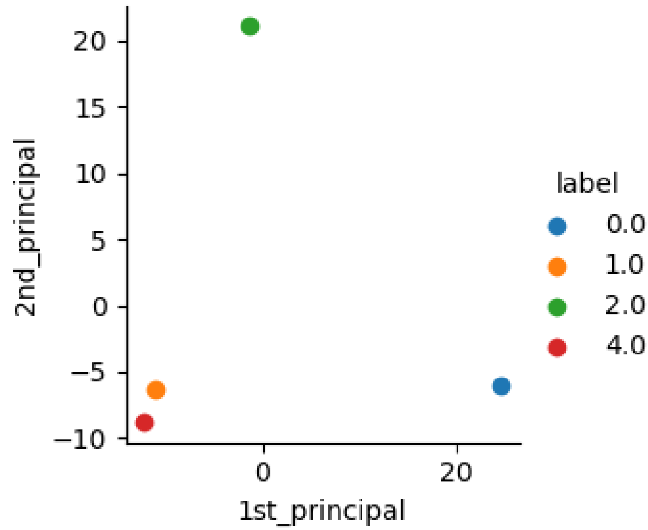
[ ]:
```
[(0.0, 28.0), (28.0, 0.0), None]
```

1.3 PCA on 4 Points

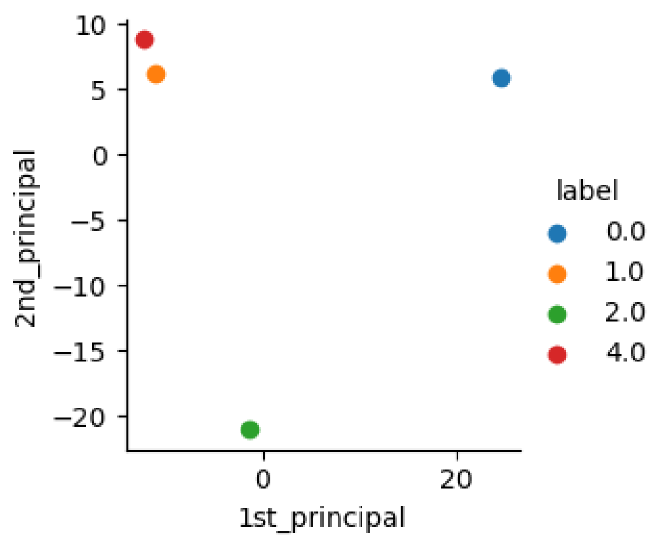
[ ]:
```
for minibatch in train_loader:
    data, labels = minibatch['data'], minibatch['label']
    scaler = StandardScaler()
    X = scaler.fit_transform(data.numpy())

    pca = PCA(n_components=2)
    reduced_data = pca.fit_transform(X)

    pca_data = np.vstack((reduced_data.T, labels)).T
    pca_df = pd.DataFrame(data=pca_data, columns=("1st_principal",␣
 ↪"2nd_principal", "label"))
    sn.FacetGrid(pca_df, hue="label").map(plt.scatter, '1st_principal',␣
 ↪'2nd_principal').add_legend()
    plt.show()
```

```python
    print(f'n_components=2: (eigenvalues={pca.singular_values_}, var={pca.explained_variance_ratio_})')

    pca = PCA(n_components=0.99, svd_solver='auto')
    reduced_data = pca.fit_transform(X)

    # pca_data = np.vstack((reduced_data.T, labels)).T
    # pca_df = pd.DataFrame(data=pca_data, columns=("1st_principal", "2nd_principal", "label"))
    # sn.FacetGrid(pca_df, hue="label").map(plt.scatter, '1st_principal', '2nd_principal').add_legend()
    # plt.show()

    print(f'n_components=0.99: (eigenvalues={pca.singular_values_}, var={pca.explained_variance_ratio_})')

    covariance_matrix = np.cov(X, ddof = 1, rowvar = False)

    print('shape of cov_mat:', covariance_matrix.shape)

    eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)


    # np.argsort can only provide lowest to highest; use [::-1] to reverse the list
    order_of_importance = np.argsort(eigenvalues)[::-1]

    # utilize the sort order to sort eigenvalues and eigenvectors
    sorted_eigenvalues = eigenvalues[order_of_importance]
    sorted_eigenvectors = eigenvectors[:,order_of_importance] # sort the columns

    # use sorted_eigenvalues to ensure the explained variances correspond to the eigenvectors
    explained_variance = sorted_eigenvalues / np.sum(sorted_eigenvalues)

    # print(f'numpy: (eigenvalues={sorted_eigenvalues[:2]}, var={explained_variance[:2]})')

    k = 2 # select the number of principal components
    reduced_data = np.matmul(X, sorted_eigenvectors[:,:k]).real # transform the original data


    pca_data = np.vstack((reduced_data.T, labels)).T
    pca_df = pd.DataFrame(data=pca_data, columns=("1st_principal", "2nd_principal", "label"))
```

```
sn.FacetGrid(pca_df, hue="label").map(plt.scatter, '1st_principal',␣
↪'2nd_principal').add_legend()
plt.show()
```




```
n_components=2: (eigenvalues=[29.496016 24.441822], var=[0.45502877 0.31244913])
n_components=0.99: (eigenvalues=[29.496017 24.441813 21.085123], var=[0.4550289
0.31244892 0.23252222])
shape of cov_mat: (784, 784)
```



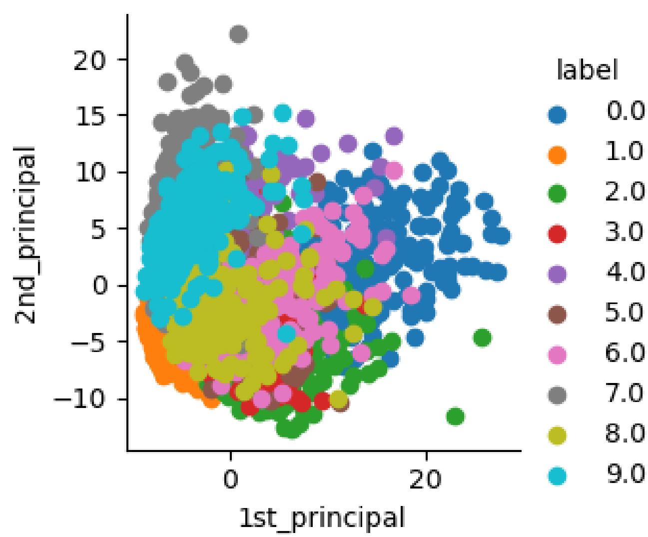
These are the similar results but 1st and 2nd principal are changed (x and y axes are flipped)

1.4 PCA on MNIST

[ ]:
```
test_dataset = datasets.MNIST(root='./data/', train=False, download= True,␣
 ↪transform=transforms.ToTensor())
length=len(test_dataset)
test_data, _ = random_split(test_dataset, [int(length/4), length - int(length/
 ↪4)], torch.Generator().manual_seed(42))
```

[ ]:
```
data = [sample.numpy() for sample, _ in iter(test_data)]
labels = [label for _, label in iter(test_data)]
data = np.array(data).reshape(2500,784)

scaler = StandardScaler()
X = scaler.fit_transform(data)

pca = PCA(n_components=2)
reduced_data = pca.fit_transform(X)

pca_data = np.vstack((reduced_data.T, labels)).T
pca_df = pd.DataFrame(data=pca_data, columns=("1st_principal", "2nd_principal",␣
 ↪"label"))
sn.FacetGrid(pca_df, hue="label").map(plt.scatter, '1st_principal',␣
 ↪'2nd_principal').add_legend()
plt.show()

print(f'n_components=2: (eigenvalues={pca.singular_values_}, var={pca.
 ↪explained_variance_ratio_})')

pca = PCA(n_components=0.99, svd_solver='auto')
reduced_data = pca.fit_transform(X)

# pca_data = np.vstack((reduced_data.T, labels)).T
# pca_df = pd.DataFrame(data=pca_data, columns=("1st_principal",␣
 ↪"2nd_principal", "label"))
# sn.FacetGrid(pca_df, hue="label").map(plt.scatter, '1st_principal',␣
 ↪'2nd_principal').add_legend()
# plt.show()

print(f'n_components=0.99: (eigenvalues={pca.singular_values_}, var={pca.
 ↪explained_variance_ratio_})')

covariance_matrix = np.cov(X, ddof = 1, rowvar = False)

print('shape of cov_mat:', covariance_matrix.shape)

eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
```

```python
# np.argsort can only provide lowest to highest; use [::-1] to reverse the list
order_of_importance = np.argsort(eigenvalues)[::-1]

# utilize the sort order to sort eigenvalues and eigenvectors
sorted_eigenvalues = eigenvalues[order_of_importance]
sorted_eigenvectors = eigenvectors[:,order_of_importance] # sort the columns

# use sorted_eigenvalues to ensure the explained variances correspond to the␣
↪eigenvectors
explained_variance = sorted_eigenvalues / np.sum(sorted_eigenvalues)

# print(f'numpy: (eigenvalues={sorted_eigenvalues[:2]},␣
↪var={explained_variance[:2]})')

k = 2 # select the number of principal components
reduced_data = np.matmul(X, sorted_eigenvectors[:,:k]).real # transform the␣
↪original data


pca_data = np.vstack((reduced_data.T, labels)).T
pca_df = pd.DataFrame(data=pca_data, columns=("1st_principal", "2nd_principal",␣
↪"label"))
sn.FacetGrid(pca_df, hue="label").map(plt.scatter, '1st_principal',␣
↪'2nd_principal').add_legend()
plt.show()
```



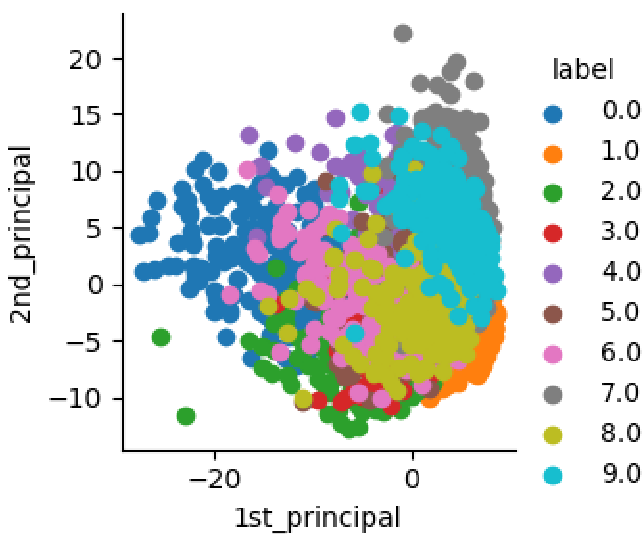

```
n_components=2: (eigenvalues=[328.312   267.39743], var=[0.06747331 0.04475823])
```

```
n_components=0.99: (eigenvalues=[328.31192   267.3974    257.62158   229.13211
217.43462   201.35616
 183.06398   180.20135   166.44815   161.26115   157.02655   149.85417
 146.48288   141.44576   140.0831    137.60918   135.10204   133.01936
 126.76211   125.16599   124.28719   122.10526   121.75734   121.02499
 117.18458   116.71128   114.493706  113.032074  111.71567   110.106865
 108.56846   105.37345   104.83329   104.00411   102.34528   101.52652
 100.33418    99.52088    99.17838    98.349976   96.416115   95.96739
  94.265526   93.428276   92.28855    91.31641    90.29164    89.603294
  89.19964    88.71624    87.255615   86.47323    86.05316    85.492645
  84.37844    83.20649    82.84735    82.498245   81.906334   81.12046
  80.668564   80.29925    79.78235    78.920044   78.31974    78.09795
  76.51799    75.834496   75.34887    74.50025    74.0016     73.00515
  72.35509    71.78595    71.6675     71.487144   70.75498    70.50553
  70.23113    69.68468    69.12309    68.42439    68.05117    67.58496
  67.17955    66.79936    66.472305   65.92506    65.228294   65.029785
  64.05229    63.538208   62.94462    62.763405   62.390343   62.17717
  61.416233   60.72823    59.887638   59.6627     59.42035    58.83872
  58.398193   58.22887    57.92011    57.358498   57.2022     56.703865
  56.34179    55.521313   55.168587   54.85908    54.48974    54.118763
  53.690247   53.355476   53.118855   52.796593   52.375027   52.110683
  51.770363   51.531605   51.078167   50.726322   50.272686   50.152832
  49.814354   49.698517   49.599907   49.045628   48.91064    48.595474
  48.553696   48.080536   47.686886   47.515034   47.454575   47.132374
  46.93153    46.709797   46.581955   46.137306   45.914463   45.269203
  44.732506   44.48463    44.35254    43.91887    43.73486    43.679592
  43.000042   42.811462   42.29912    42.06694    41.871227   41.353924
  41.127563   40.97888    40.867043   40.65539    40.408653   39.84722
  39.828667   39.48416    39.275257   39.204865   38.757927   38.560837
  38.45193    38.23161    37.977936   37.713436   37.576435   37.456192
  37.028587   36.903584   36.596703   36.131126   35.882812   35.812958
  35.640133   35.512062   35.163403   34.813774   34.66869    34.544823
  34.271786   34.142887   33.794388   33.535904   33.266876   33.138023
  33.002472   32.85953    32.609104   32.389584   32.328922   32.17732
  31.94707    31.766428   31.634539   31.468397   31.13243    30.840733
  30.812666   30.58545    30.544067   30.183054   29.86675    29.83591
  29.545544   29.375145   29.272648   29.090355   29.033228   28.906193
  28.579414   28.435385   28.377432   28.171307   28.108995   27.886118
  27.708902   27.6652     27.583378   27.449196   27.240307   27.159203
  27.064922   26.835629   26.629143   26.418125   26.211212   26.086546
  26.039387   25.84593    25.75487    25.635752   25.508875   25.25983
  25.115753   25.082325   24.948633   24.876059   24.622053   24.545036
  24.42119    24.201094   24.19116    24.063025   24.015362   23.94254
  23.815329   23.526089   23.410034   23.340532   23.20676    23.013582
  22.97622    22.830761   22.815567   22.646149   22.585587   22.506699
  22.373274   22.24033    22.083746   21.998075   21.867683   21.837898
  21.695042   21.602856   21.528465   21.320724   21.2605     21.19343
  21.07828    21.037537   20.88414    20.839655   20.803225   20.703749
```

```
  20.590302   20.383099   20.345655   20.284245   20.216682   20.0972
  19.889366   19.870996   19.842371   19.689129   19.582462   19.573273
  19.485552   19.426973   19.390772   19.330124   19.283442   19.099888
  18.999655   18.896458   18.788937   18.777979   18.628191   18.51084
  18.454617   18.409172   18.292454   18.248297   18.131062   17.989788
  17.974579   17.949236   17.85058    17.826633   17.727345   17.663677
  17.59844    17.578781   17.553392   17.453152   17.394815   17.363094
  17.248188   17.137388   17.071587   16.999393   16.932302   16.879297
  16.809134   16.709072   16.66007    16.589052   16.512663   16.430841
  16.328695   16.282661   16.213207   16.106339   16.098492   16.041939
  15.9646     15.945635   15.874446   15.837626   15.797942   15.724082
  15.685494   15.6303625  15.550492   15.501922   15.426749   15.36526
  15.32884    15.296106   15.2508     15.169233   15.111655   15.038998
  14.977996   14.926137   14.902096   14.794979   14.757321   14.731519
  14.691891   14.601949   14.564461   14.519247   14.414367   14.398438
  14.36961    14.348224   14.294868   14.226135   14.167667   14.15516
  14.111712   14.091227   14.002069   13.941366   13.875924   13.8474865
  13.772425   13.690768   13.680371   13.614604   13.56007    13.502054
  13.460384   13.430844   13.409784   13.336676   13.301787   13.260845
  13.173428   13.1452     13.082229   13.052574   13.04252    12.962846
  12.915641   12.863602   12.833826   12.814762   12.763894   12.705644
  12.691904   12.601709   12.555081   12.538312   12.50714    12.449052 ],
var=[6.74733669e-02 4.47582826e-02 4.15454581e-02 3.28647979e-02
 2.95948684e-02 2.53798421e-02 2.09780391e-02 2.03270894e-02
 1.73427109e-02 1.62786581e-02 1.54349515e-02 1.40571333e-02
 1.34317568e-02 1.25238802e-02 1.22837387e-02 1.18536977e-02
 1.14257019e-02 1.10761486e-02 1.00586098e-02 9.80690029e-03
 9.66967363e-03 9.33314115e-03 9.28002968e-03 9.16872919e-03
 8.59607104e-03 8.52677412e-03 8.20582546e-03 7.99765158e-03
 7.81245017e-03 7.58905802e-03 7.37847155e-03 6.95058703e-03
 6.87950989e-03 6.77111372e-03 6.55684201e-03 6.45235181e-03
 6.30168850e-03 6.19994057e-03 6.15733955e-03 6.05490850e-03
 5.81913348e-03 5.76509489e-03 5.56243397e-03 5.46406349e-03
 5.33156516e-03 5.21983439e-03 5.10333618e-03 5.02582127e-03
 4.98064142e-03 4.92680445e-03 4.76591010e-03 4.68082493e-03
 4.63545881e-03 4.57526837e-03 4.45678877e-03 4.33384581e-03
 4.29651467e-03 4.26038168e-03 4.19946574e-03 4.11926676e-03
 4.07349970e-03 4.03628685e-03 3.98448994e-03 3.89882480e-03
 3.83973774e-03 3.81802162e-03 3.66510311e-03 3.59991868e-03
 3.55396024e-03 3.47435800e-03 3.42800398e-03 3.33630736e-03
 3.27715673e-03 3.22580407e-03 3.21516767e-03 3.19900527e-03
 3.13381315e-03 3.11175524e-03 3.08758137e-03 3.03972047e-03
 2.99092405e-03 2.93076481e-03 2.89888028e-03 2.85929674e-03
 2.82509625e-03 2.79321056e-03 2.76592607e-03 2.72057136e-03
 2.66336766e-03 2.64718127e-03 2.56819767e-03 2.52713822e-03
 2.48014042e-03 2.46588071e-03 2.43665371e-03 2.42003123e-03
 2.36116000e-03 2.30855541e-03 2.24508834e-03 2.22825492e-03
 2.21018912e-03 2.16713268e-03 2.13480345e-03 2.12244177e-03
```

2.09999294e-03	2.05946597e-03	2.04825751e-03	2.01272476e-03
1.98710291e-03	1.92964997e-03	1.90520973e-03	1.88389246e-03
1.85861101e-03	1.83338975e-03	1.80447078e-03	1.78203848e-03
1.76626747e-03	1.74490118e-03	1.71714742e-03	1.69985800e-03
1.67772768e-03	1.66228844e-03	1.63316366e-03	1.61074125e-03
1.58206106e-03	1.57452666e-03	1.55334559e-03	1.54612970e-03
1.54000032e-03	1.50577351e-03	1.49749627e-03	1.47825957e-03
1.47571904e-03	1.44709717e-03	1.42349850e-03	1.41325709e-03
1.40966289e-03	1.39058556e-03	1.37875951e-03	1.36576209e-03
1.35829614e-03	1.33248873e-03	1.31964788e-03	1.28281722e-03
1.25258020e-03	1.23873679e-03	1.23139110e-03	1.20742840e-03
1.19733182e-03	1.19430770e-03	1.15743559e-03	1.14730571e-03
1.12000969e-03	1.10774790e-03	1.09746447e-03	1.07051444e-03
1.05882704e-03	1.05118530e-03	1.04545534e-03	1.03465456e-03
1.02213398e-03	9.93928523e-04	9.93003021e-04	9.75899107e-04
9.65599727e-04	9.62141552e-04	9.40329628e-04	9.30790557e-04
9.25540458e-04	9.14964534e-04	9.02862812e-04	8.90330528e-04
8.83873727e-04	8.78226012e-04	8.58288666e-04	8.52503465e-04
8.38384032e-04	8.17188178e-04	8.05994321e-04	8.02859315e-04
7.95129221e-04	7.89424987e-04	7.73999782e-04	7.58684648e-04
7.52374297e-04	7.47007551e-04	7.35245761e-04	7.29725522e-04
7.14904862e-04	7.04010425e-04	6.92760455e-04	6.87404303e-04
6.81792211e-04	6.75898977e-04	6.65635976e-04	6.56704186e-04
6.54246600e-04	6.48125017e-04	6.38882688e-04	6.31678151e-04
6.26443769e-04	6.19880971e-04	6.06715505e-04	5.95399470e-04
5.94316283e-04	5.85583446e-04	5.83999965e-04	5.70276403e-04
5.58386673e-04	5.57234103e-04	5.46440657e-04	5.40155917e-04
5.36392967e-04	5.29733137e-04	5.27654600e-04	5.23047172e-04
5.11288119e-04	5.06147684e-04	5.04086725e-04	4.96790220e-04
4.94594977e-04	4.86782723e-04	4.80615417e-04	4.79100592e-04
4.76270769e-04	4.71648382e-04	4.64497134e-04	4.61735355e-04
4.58535156e-04	4.50798630e-04	4.43888071e-04	4.36880917e-04
4.30064189e-04	4.25982958e-04	4.24444152e-04	4.18160867e-04
4.15219518e-04	4.11387562e-04	4.07325599e-04	3.99410899e-04
3.94867617e-04	3.93817201e-04	3.89630179e-04	3.87366628e-04
3.79496312e-04	3.77125951e-04	3.73329851e-04	3.66630877e-04
3.66329972e-04	3.62459541e-04	3.61025042e-04	3.58838879e-04
3.55035852e-04	3.46464338e-04	3.43054562e-04	3.41020583e-04
3.37122765e-04	3.31533549e-04	3.30457988e-04	3.26287060e-04
3.25852918e-04	3.21031577e-04	3.19316867e-04	3.17090075e-04
3.13341676e-04	3.09628929e-04	3.05284368e-04	3.02920351e-04
2.99339910e-04	2.98525032e-04	2.94632075e-04	2.92133540e-04
2.90125026e-04	2.84552894e-04	2.82947614e-04	2.81165179e-04
2.78118183e-04	2.77044048e-04	2.73018610e-04	2.71856756e-04
2.70907098e-04	2.68322445e-04	2.65389943e-04	2.60075525e-04
2.59120920e-04	2.57559004e-04	2.55846098e-04	2.52830912e-04
2.47628690e-04	2.47171469e-04	2.46459880e-04	2.42667753e-04
2.40045571e-04	2.39820321e-04	2.37675544e-04	2.36248670e-04

```
 2.35369007e-04 2.33898973e-04 2.32770602e-04 2.28360368e-04
 2.25969838e-04 2.23521783e-04 2.20985356e-04 2.20727656e-04
 2.17220309e-04 2.14492087e-04 2.13191131e-04 2.12142448e-04
 2.09460923e-04 2.08450889e-04 2.05781107e-04 2.02586816e-04
 2.02244395e-04 2.01674513e-04 1.99463655e-04 1.98928828e-04
 1.96719062e-04 1.95308574e-04 1.93868589e-04 1.93435684e-04
 1.92877327e-04 1.90680730e-04 1.89408165e-04 1.88717997e-04
 1.86228455e-04 1.83843527e-04 1.82434436e-04 1.80894727e-04
 1.79469687e-04 1.78347807e-04 1.76868183e-04 1.74768735e-04
 1.73745168e-04 1.72267057e-04 1.70684187e-04 1.68996892e-04
 1.66902188e-04 1.65962454e-04 1.64549652e-04 1.62387543e-04
 1.62229349e-04 1.61091564e-04 1.59542033e-04 1.59163217e-04
 1.57745220e-04 1.57014321e-04 1.56228460e-04 1.54771027e-04
 1.54012334e-04 1.52931578e-04 1.51372631e-04 1.50428503e-04
 1.48973122e-04 1.47787912e-04 1.47088140e-04 1.46460618e-04
 1.45594284e-04 1.44041071e-04 1.42949662e-04 1.41578348e-04
 1.40432123e-04 1.39461365e-04 1.39012482e-04 1.37021198e-04
 1.36324568e-04 1.35848270e-04 1.35118389e-04 1.33469090e-04
 1.32784655e-04 1.31961511e-04 1.30061933e-04 1.29774649e-04
 1.29255495e-04 1.28871048e-04 1.27914376e-04 1.26687257e-04
 1.25648046e-04 1.25426319e-04 1.24657527e-04 1.24295868e-04
 1.22727957e-04 1.21666148e-04 1.20526594e-04 1.20033088e-04
 1.18735312e-04 1.17331518e-04 1.17153380e-04 1.16029682e-04
 1.15102019e-04 1.14119212e-04 1.13415910e-04 1.12918649e-04
 1.12564812e-04 1.11340771e-04 1.10759007e-04 1.10078247e-04
 1.08631721e-04 1.08166671e-04 1.07132830e-04 1.06647676e-04
 1.06483429e-04 1.05186438e-04 1.04421750e-04 1.03581981e-04
 1.03103011e-04 1.02796927e-04 1.01982449e-04 1.01053745e-04
 1.00835307e-04 9.94072325e-05 9.86729501e-05 9.84095459e-05
 9.79208344e-05 9.70133769e-05])
shape of cov_mat: (784, 784)
```



2 Problem 2:

[ ]:
```
import torch
ten1 = torch.rand((2,3))
ten2 = torch.rand((2,3))
tenA = third_tensor = torch.cat((ten1, ten2), 1)
tenA.shape
```

[ ]:
```
torch.Size([2, 6])
```

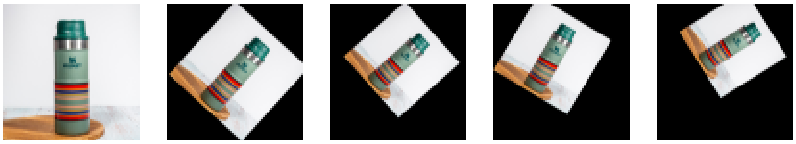
[ ]:
```
from PIL import Image
from pathlib import Path
import matplotlib.pyplot as plt

import torch
from torchvision.transforms import v2

plt.rcParams["savefig.bbox"] = 'tight'

# if you change the seed, make sure that the randomly-applied transforms
# properly show that the image can be both transformed and *not* transformed!
torch.manual_seed(0)

# If you're trying to run that on collab, you can download the assets and the
# helpers from https://github.com/pytorch/vision/tree/main/gallery/
# from helpers import plot
orig_img = Image.open('cup.jpg')

def plot(images, rows=1):
    for num, img in enumerate(images):
        plt.subplot(rows,6,num+1)

        plt.axis('off')
        plt.imshow(img)
```

2.1 Random Rotation

[ ]:
```
rotater = v2.RandomRotation(degrees=(0, 180))
rotated_imgs = [rotater(orig_img) for _ in range(4)]

plot([orig_img]+rotated_imgs)
```

3 Random Translation-and-Resizing

[ ]:
```
affine_transfomer = v2.RandomAffine(degrees=(30, 70), translate=(0.1, 0.3),␣
    ↪scale=(0.5, 0.75))
affine_imgs = [affine_transfomer(orig_img) for _ in range(4)]
plot([orig_img] + affine_imgs)
```



3.1 Both Augmentation

[ ]:
```
rotater = v2.RandomRotation(degrees=(0, 180))
affine_transfomer = v2.RandomAffine(degrees=(50, 90), translate=(0.1, 0.15),␣
    ↪scale=(0.9, 1))

rotated_imgs = [affine_transfomer(rotater(orig_img)) for _ in range(4)]

plot([orig_img] + rotated_imgs)
```