Problem 1

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
```

In [2]:

```
X = np.genfromtxt('data/X_train.txt', delimiter=None)
Y = np.genfromtxt('data/Y_train.txt', delimiter=None)
X,Y = ml.shuffleData(X,Y)
```

In [3]:

```
#1.1
print('minimum:', np.min(X,axis=0))
print('maximum:', np.max(X,axis=0))
print('mean:', np.mean(X,axis=0))
print('variance:', np.var(X,axis=0))
```

```
minimum: [ 1.9300e+02  1.9000e+02  2.1497e+02  2.0542e+02  1.0000e+01  0.0
000e+00
  0.0000e+00  0.0000e+00  6.8146e-01  0.0000e+00  0.0000e+00  0.0000e+00
  1.0074e+00 -9.9990e+02]
maximum: [2.5300e+02 2.5050e+02 2.5250e+02 2.5250e+02 1.7130e+04 1.2338e+0
4
 9.2380e+03 3.5796e+01 1.9899e+01 1.1368e+01 2.1466e+01 1.4745e+01
 2.7871e+02 7.8250e+02]
mean: [2.41797220e+02 2.28228260e+02 2.41796298e+02 2.33649299e+02
 2.86797959e+03 8.84073295e+02 1.73553355e+02 3.04719572e+00
 6.35196722e+00 1.92523232e+00 4.29379349e+00 2.80947178e+00
 1.03679146e+01 7.87334450e+00]
variance: [8.26945619e+01 9.09573945e+01 3.57255796e+01 9.52608539e+01
 1.06194180e+07 3.25702985e+06 7.40656134e+05 7.42244277e+00
 6.33229913e+00 4.28448703e+00 4.04684087e+00 1.98218303e+00
 1.66679252e+02 1.41079679e+03]
```

In [4]:

```
#1.2
Xtr, Xva, Ytr, Yva = ml.splitData(X, Y)
Xt, Yt = Xtr[:5000], Ytr[:5000] # subsample for efficiency (you can go higher)
XtS, params = ml.rescale(Xt) # Normalize the features
XvS, _ = ml.rescale(Xva, params) # Normalize the features
```

In [5]:

```
print("Xtrain:")
print("minimum: ",XtS.min(axis=0))
print("maximum: ",XtS.max(axis=0))
print("mean: ",XtS.mean(axis=0))
print("variance: ",XtS.var(axis=0))
```

```
Xtrain:
minimum:  [ -4.81279637  -3.99730114  -4.48020457  -2.88380268  -0.8763075
6
  -0.49059219  -0.20464878  -1.1040482   -1.88223587  -0.93699308
  -2.13222165  -1.99396338  -0.7595542  -23.75502203]
maximum:  [ 1.22052625  1.97496864  1.6435955   1.84378903  4.28248905  6.
17759004
 10.87231593  8.25546724  4.1591937   4.54752431  6.8442203   5.83209822
 22.57472839 17.24333273]
mean:  [ 2.50373333e-15  1.10094156e-15 -6.91087187e-14 -1.46315271e-13
  5.04041253e-18 -1.64523950e-16  6.33559871e-16  1.01132436e-15
 -5.53523893e-15 -2.30233610e-15 -8.98325858e-16 -7.51692042e-15
 -1.54722901e-15  1.55577773e-15]
variance:  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [6]:

```
print("Xval:")
print("minimum: ",XvS.min(axis=0))
print("maximum: ",XvS.max(axis=0))
print("mean: ",XvS.mean(axis=0))
print("variance: ",XvS.var(axis=0))
```

```
Xval:
minimum:  [ -5.30643185  -3.99730114  -4.48020457  -2.88380268  -0.8766089
1
  -0.49059219  -0.20464878  -1.1040482   -2.17482065  -0.93699308
  -2.13222165  -1.99396338  -0.7595542  -23.75502203]
maximum:  [ 1.22052625  2.18452196  1.79577076  1.93696877  4.28248905  6.
17759004
 10.87231593 11.64649278  5.35874338  4.54752431  8.21972332  8.43866281
 22.57472839 17.48853538]
mean:  [-0.0093118  -0.00039265 -0.00297315  0.00062205 -0.0182845  -0.006
52237
  0.01334365 -0.00899597  0.00321656  0.00178165  0.01576998  0.00290157
  0.01960524  0.00867894]
variance:  [1.00879736 1.02018885 1.02846987 1.01402694 0.97014596 0.98356
14
 1.13345537 0.97104384 1.0129362  1.02092571 1.01089869 1.00092762
 1.16890334 0.76150193]
```

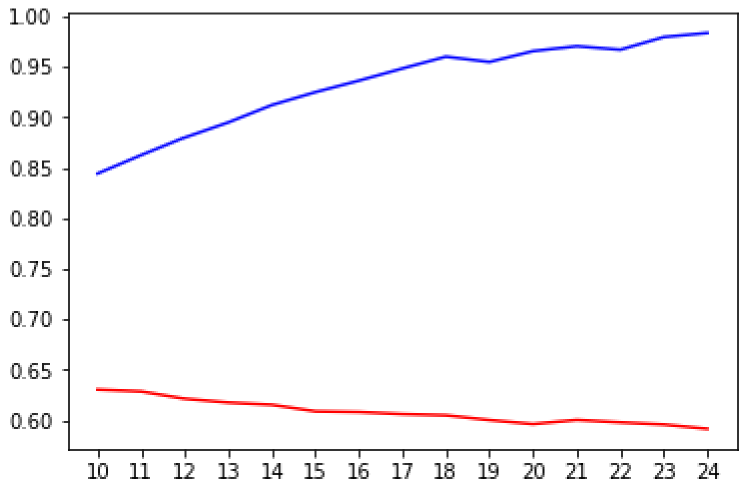
Problem 2

In [7]:

```
#2.1
def train_tree(md=15, mp=2,mlf=1):
    learner = ml.dtree.treeClassify(XtS, Yt, maxDepth=md, minParent=mp, minLeaf=mlf)
    probs = learner.predictSoft(XvS)
    t_auc = learner.auc(XtS, Yt)
    v_auc = learner.auc(XvS, Yva)
    return (t_auc, v_auc, learner.sz)

def draw_line(line1, line2, r1,r2):
    plt.figure(1)
    plt.xticks(list(range(0,r2-r1)), list(range(r1,r2)))
    plt.plot(line1,'b-',line2,'r-')
    plt.draw()
```

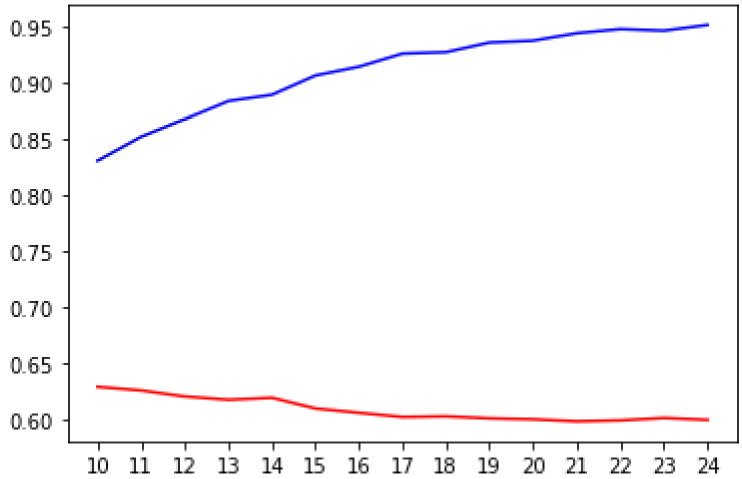
In [8]:

```
Train_AUC = []
Val_AUC = []
sz1 = []
for i in range(10,25):
    t,v,s = train_tree(md=i)
    Train_AUC.append(t)
    Val_AUC.append(v)
    sz1.append(s)
draw_line(Train_AUC,Val_AUC,10,25)
```



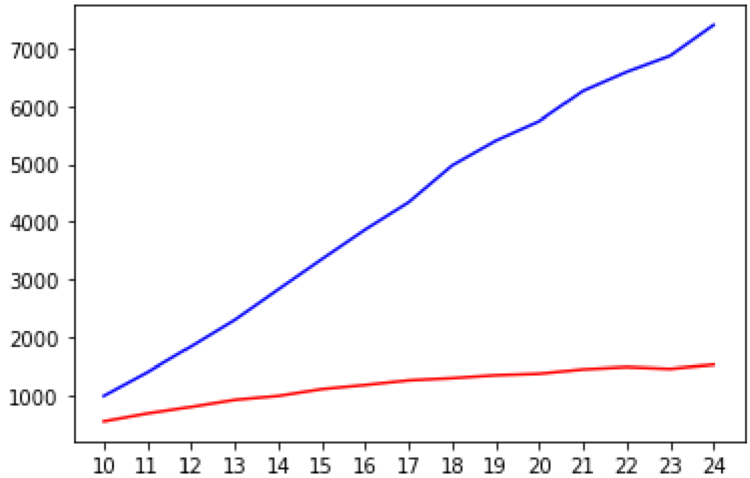
In [9]:

```
#2.2
Train_AUC = []
Val_AUC = []
sz2 = []
for i in range(10,25):
    t,v,s = train_tree(mlf=5,md=i)
    Train_AUC.append(t)
    Val_AUC.append(v)
    sz2.append(s)
draw_line(Train_AUC,Val_AUC,10,25)
```



In [10]:

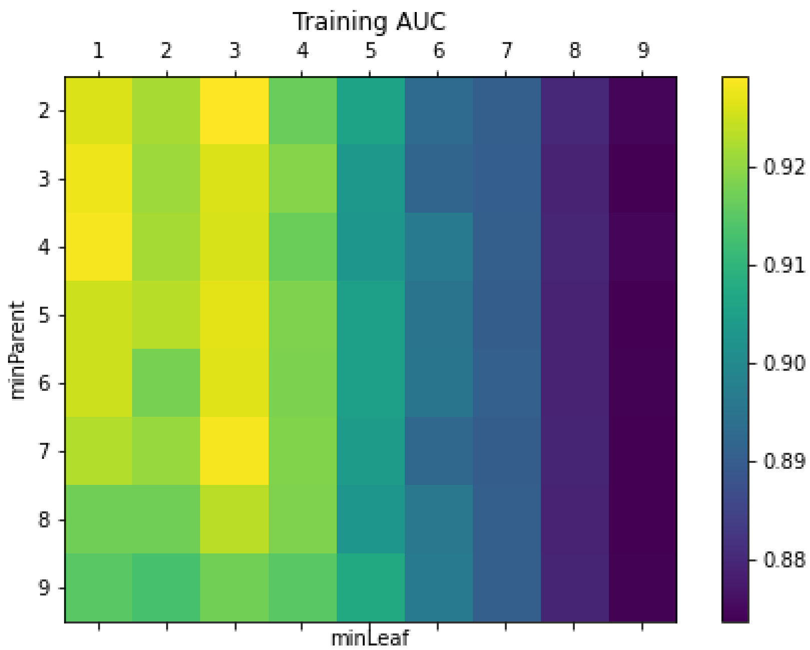
```
draw_line(sz1,sz2,10,25)
```



In [11]:

```
#Training one
K = range(2,10,1)
A = range(1,10,1)

tr_auc = np.zeros((len(K),len(A)))
va_auc = np.zeros((len(K),len(A)))

for i,k in enumerate(K):
    for j,a in enumerate(A):
        learner = ml.dtree.treeClassify()
        learner.train(Xt, Yt, maxDepth = 15, minParent = k, minLeaf = a)
        tr_auc[i][j] = learner.auc(Xt, Yt)
        va_auc[i][j] = learner.auc(Xva, Yva)

f, ax = plt.subplots(1, 1, figsize=(8, 5))
caxtr4 = ax.matshow(tr_auc, interpolation='nearest')
f.colorbar(caxtr4)
ax.set_xticklabels(['']+[*A])
ax.set_yticklabels(['']+[*K])
ax.set_xlabel("minLeaf")
ax.set_ylabel("minParent")
ax.set_title("Training AUC")
plt.show()
```

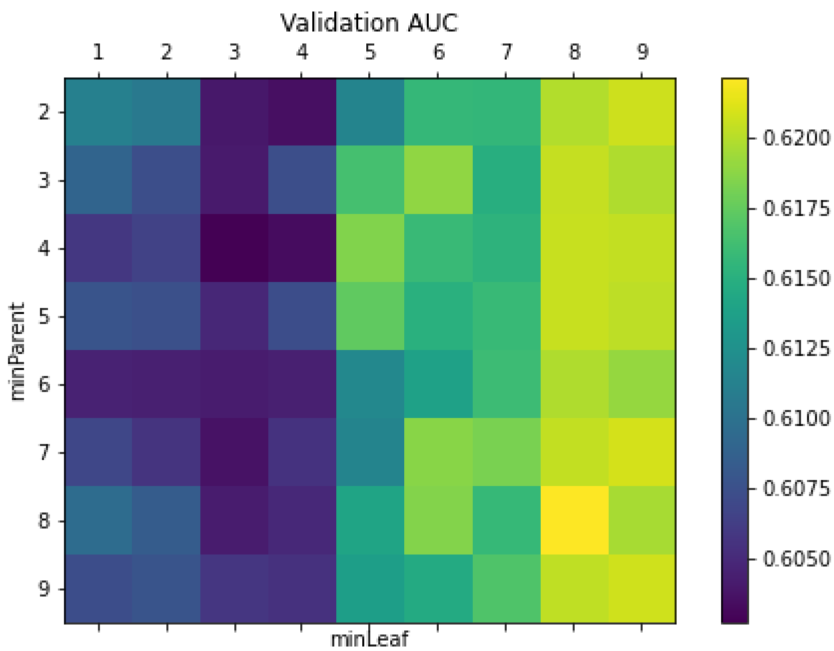
```
C:\Users\10630\AppData\Local\Temp/ipykernel_13980/380761089.py:18: UserWar
ning: FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(['']+[*A])
C:\Users\10630\AppData\Local\Temp/ipykernel_13980/380761089.py:19: UserWar
ning: FixedFormatter should only be used together with FixedLocator
  ax.set_yticklabels(['']+[*K])
```



In [12]:

```
#validation one
f, ax = plt.subplots(1, 1, figsize=(8, 5))
caxva4 = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(caxva4)
ax.set_xticklabels(['']+[*A])
ax.set_yticklabels(['']+[*K])
ax.set_xlabel("minLeaf")
ax.set_ylabel("minParent")
ax.set_title("Validation AUC")
plt.show()
```

```
C:\Users\10630\AppData\Local\Temp/ipykernel_13980/2370792878.py:5: UserWar
ning: FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(['']+[*A])
C:\Users\10630\AppData\Local\Temp/ipykernel_13980/2370792878.py:6: UserWar
ning: FixedFormatter should only be used together with FixedLocator
  ax.set_yticklabels(['']+[*K])
```



In [13]:

```
#As a result, I recommend that minileaf = 4, miniparent = 5
```

Problem 3

In [14]:

```
import numpy as np
from datetime import datetime

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader

from torchvision import datasets, transforms

%matplotlib inline
import matplotlib.pyplot as plt

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [15]:

```
# define transforms
transforms = transforms.Compose([transforms.Resize((32, 32)),
                                 transforms.ToTensor()])

# download and create datasets
train_dataset = datasets.MNIST(root='mnist_data',
                               train=True,
                               transform=transforms,
                               download=True)

valid_dataset = datasets.MNIST(root='mnist_data',
                               train=False,
                               transform=transforms)
```

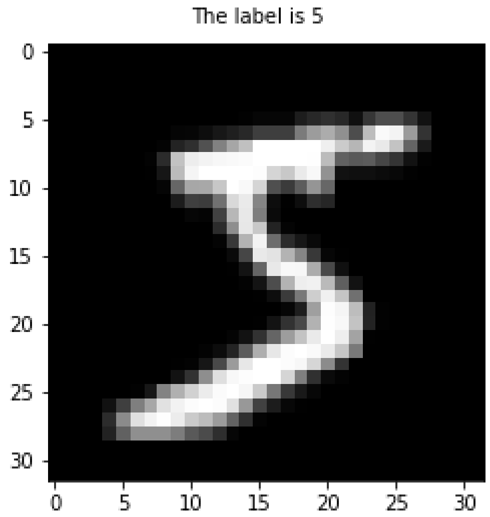
3.1.1

In [16]:

```
plt.imshow(train_dataset[0][0][0], cmap='gray')
plt.text(10, -2, 'The label is ' + str(train_dataset[0][1]))
```

Out[16]:

```
Text(10, -2, 'The label is 5')
```



In [17]:

```
# hyper parameters
RANDOM_SEED = 42
LEARNING_RATE = 0.001
BATCH_SIZE = 32
N_EPOCHS = 15

IMG_SIZE = 32
N_CLASSES = 10
```

3.1.2

In [18]:

```
# define the data loaders
train_loader = DataLoader(dataset=train_dataset,
                          batch_size=BATCH_SIZE,
                          shuffle=True)

valid_loader = DataLoader(dataset=valid_dataset,
                          batch_size=BATCH_SIZE,
                          shuffle=True)
```

3.1.3

In [19]:

```
def train(train_loader, model, criterion, optimizer):
    '''
    Train one epoch.
    '''

    model.train()
    running_loss = 0

    for X, y_true in train_loader:
        X = X.to(device)
        y_true = y_true.to(device)
        optimizer.zero_grad()


        # Forward pass
        y_hat, _ = model(X)
        loss = criterion(y_hat, y_true)
        running_loss += loss.item() * X.size(0)

        # Backward pass
        loss.backward()
        optimizer.step()

    epoch_loss = running_loss / len(train_loader.dataset)
    return model, optimizer, epoch_loss
```

3.1.4

In [20]:

```
def validate(valid_loader, model, criterion):
    '''
    Function for the validation step of the training loop.
    Returns the model and the loss on the test set.
    '''

    model.eval()
    running_loss = 0

    for X, y_true in valid_loader:
        X = X.to(device)
        y_true = y_true.to(device)
        # Forward pass and record loss
        y_hat, _ = model(X)
        loss = criterion(y_hat, y_true)

        running_loss += loss.item() * X.size(0)

    epoch_loss = running_loss / len(valid_loader.dataset)

    return model, epoch_loss
```

In [21]:

```
def training_loop(model, criterion, optimizer, train_loader, valid_loader, epochs, print
    '''
    Function defining the entire training loop
    '''

    # set objects for storing metrics
    best_loss = 1e10
    train_losses = []
    valid_losses = []
    train_accs = []
    valid_accs = []

    # Train model
    for epoch in range(0, epochs):

        # training
        model, optimizer, train_loss = train(train_loader, model, criterion, optimizer)
        train_losses.append(train_loss)

        # validation
        with torch.no_grad():
            model, valid_loss = validate(valid_loader, model, criterion)
            valid_losses.append(valid_loss)

        if epoch % print_every == (print_every - 1):

            train_acc = get_accuracy(model, train_loader,)
            train_accs.append(train_acc)
            valid_acc = get_accuracy(model, valid_loader)
            valid_accs.append(valid_acc)

            print(f'{datetime.now().time().replace(microsecond=0)} '
                  f'Epoch: {epoch}\t'
                  f'Train loss: {train_loss:.4f}\t'
                  f'Valid loss: {valid_loss:.4f}\t'
                  f'Train accuracy: {100 * train_acc:.2f}\t'
                  f'Valid accuracy: {100 * valid_acc:.2f}')

    performance = {
        'train_losses':train_losses,
        'valid_losses': valid_losses,
        'train_acc': train_accs,
        'valid_acc':valid_accs
    }

    return model, optimizer, performance
```

3.1.5

In [22]:

```
def get_accuracy(model, data_loader):
    '''
    Function for computing the accuracy of the predictions over the entire data_loader
    '''

    correct_pred = 0
    n = 0

    with torch.no_grad():
        model.eval()
        for X, y_true in data_loader:
            X = X.to(device)
            y_true = y_true.to(device)
            y_hat, y_prob = model(X)
            predicted_labels = torch.argmax(y_prob, 1)
            n += y_true.size(0)
            correct_pred += torch.eq(predicted_labels, y_true).sum()

    return correct_pred.float() / n



def plot_performance(performance):
    '''
    Function for plotting training and validation losses
    '''

    # temporarily change the style of the plots to seaborn
    plt.style.use('seaborn')

    fig, ax = plt.subplots(1, 2, figsize = (16, 4.5))
    for key, value in performance.items():
        if 'loss' in key:
            ax[0].plot(value, label=key)
        else:
            ax[1].plot(value, label=key)
    ax[0].set(title="Loss  over epochs",
            xlabel='Epoch',
            ylabel='Loss')
    ax[1].set(title="accuracy over epochs",
            xlabel='Epoch',
            ylabel='Loss')
    ax[0].legend()
    ax[1].legend()
    plt.show()

    # change the plot style to default
    plt.style.use('default')
```

3.2.1

In [23]:

```
class LeNet5(nn.Module):

    def __init__(self, n_classes):
        super(LeNet5, self).__init__()
        self.layer1 = nn.Sequential( # use nn.Sequential to build several mini-models
            # in_channels, out_channels, kernel_size, stride
            nn.Conv2d(1, 6, (5,5), 1),
            nn.Tanh(),
            #kernel size, stride
            nn.AvgPool2d(2, 2)
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(6, 16, (5,5), 1),
            nn.Tanh(),
            nn.AvgPool2d(2, 2)
        )
        self.layer3 = nn.Sequential(
            nn.Conv2d(16,120,(5,5),1),
            nn.Tanh()
        )
        self.fc = nn.Sequential(
            nn.Linear(120, 84),
            nn.Tanh(),
            nn.Linear(84, n_classes)
        )
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = torch.flatten(x, 1)
        logits = self.fc(x)

        probs = F.softmax(logits, dim=1)
        return logits, probs
```

3.2.2

In [24]:

```
class MLP(nn.Module):

    def __init__(self, layers):
        super(MLP, self).__init__()
        self.layers = nn.ModuleList()
        for i in range(len(layers) - 1):
            self.layers.append(nn.Linear(layers[i], layers[i+1]))

    def forward(self, x):
        x = x.view(x.size(0), -1)
        for layer in self.layers[:-1]:
            x = F.relu(layer(x))
        logits = self.layers[-1](x)
        probs = F.softmax(logits, dim=1)
        return logits, probs
```

3.3.1

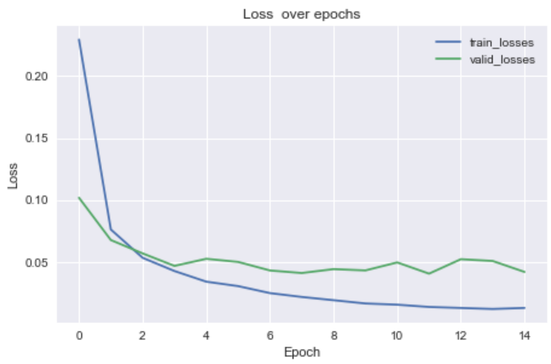
In [25]:

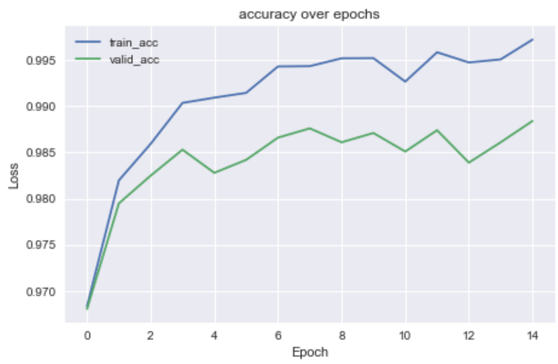
```
torch.manual_seed(RANDOM_SEED)

ln_model = LeNet5(N_CLASSES)
optimizer = torch.optim.Adam(ln_model.parameters(), lr=LEARNING_RATE)
criterion = nn.CrossEntropyLoss()
```

In [26]:

```
ln_model, optimizer, performance_1 = training_loop(ln_model, criterion, optimizer, train_
plot_performance(performance_1)
```

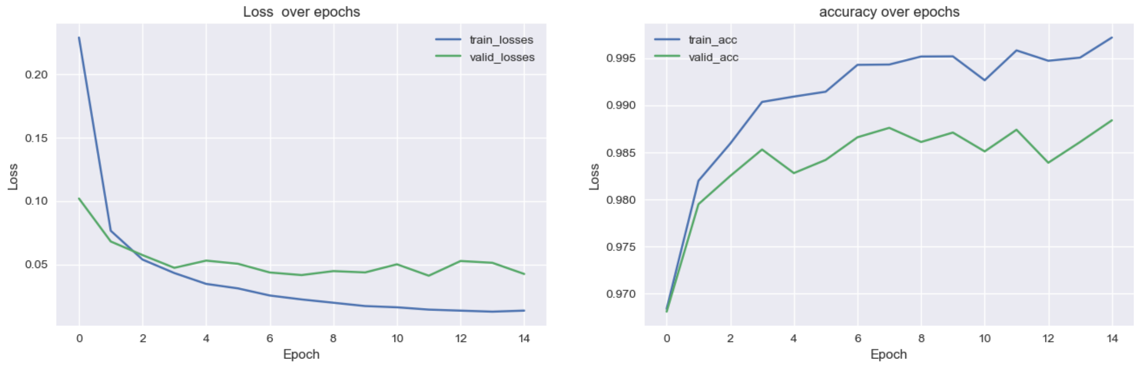
```
17:36:24 Epoch: 0       Train loss: 0.2290      Valid loss: 0.1020      Tr
ain accuracy: 96.84     Valid accuracy: 96.81
17:36:57 Epoch: 1       Train loss: 0.0766      Valid loss: 0.0681      Tr
ain accuracy: 98.20     Valid accuracy: 97.95
17:37:31 Epoch: 2       Train loss: 0.0538      Valid loss: 0.0573      Tr
ain accuracy: 98.59     Valid accuracy: 98.25
17:38:03 Epoch: 3       Train loss: 0.0432      Valid loss: 0.0473      Tr
ain accuracy: 99.04     Valid accuracy: 98.53
17:38:35 Epoch: 4       Train loss: 0.0346      Valid loss: 0.0530      Tr
ain accuracy: 99.09     Valid accuracy: 98.28
17:39:07 Epoch: 5       Train loss: 0.0311      Valid loss: 0.0505      Tr
ain accuracy: 99.14     Valid accuracy: 98.42
17:39:39 Epoch: 6       Train loss: 0.0255      Valid loss: 0.0436      Tr
ain accuracy: 99.43     Valid accuracy: 98.66
17:40:11 Epoch: 7       Train loss: 0.0224      Valid loss: 0.0416      Tr
ain accuracy: 99.43     Valid accuracy: 98.76
17:40:44 Epoch: 8       Train loss: 0.0198      Valid loss: 0.0447      Tr
ain accuracy: 99.52     Valid accuracy: 98.61
17:41:16 Epoch: 9       Train loss: 0.0171      Valid loss: 0.0437      Tr
ain accuracy: 99.52     Valid accuracy: 98.71
17:41:49 Epoch: 10      Train loss: 0.0162      Valid loss: 0.0501      Tr
ain accuracy: 99.26     Valid accuracy: 98.51
17:42:23 Epoch: 11      Train loss: 0.0143      Valid loss: 0.0411      Tr
ain accuracy: 99.58     Valid accuracy: 98.74
17:42:56 Epoch: 12      Train loss: 0.0135      Valid loss: 0.0527      Tr
ain accuracy: 99.47     Valid accuracy: 98.39
17:43:29 Epoch: 13      Train loss: 0.0127      Valid loss: 0.0513      Tr
ain accuracy: 99.51     Valid accuracy: 98.61
17:44:02 Epoch: 14      Train loss: 0.0136      Valid loss: 0.0425      Tr
ain accuracy: 99.72     Valid accuracy: 98.84
```





In [27]:

```
plot_performance(performance_1)
```



In [28]:

```
# We can see that Lenet5 works well here.
# The graph shows that there is a huge decrease of loss, and also there are both high ra
# on training and testing data accuracy
```

3.3.2

In [29]:

```
torch.manual_seed(RANDOM_SEED)
layers = [1024, 256, 64, 16, N_CLASSES]
MLP_model = MLP(layers)
print(MLP_model)
optimizer = torch.optim.Adam(MLP_model.parameters(), lr=LEARNING_RATE)
criterion = nn.CrossEntropyLoss()
```

```
MLP(
  (layers): ModuleList(
    (0): Linear(in_features=1024, out_features=256, bias=True)
    (1): Linear(in_features=256, out_features=64, bias=True)
    (2): Linear(in_features=64, out_features=16, bias=True)
    (3): Linear(in_features=16, out_features=10, bias=True)
  )
)
```

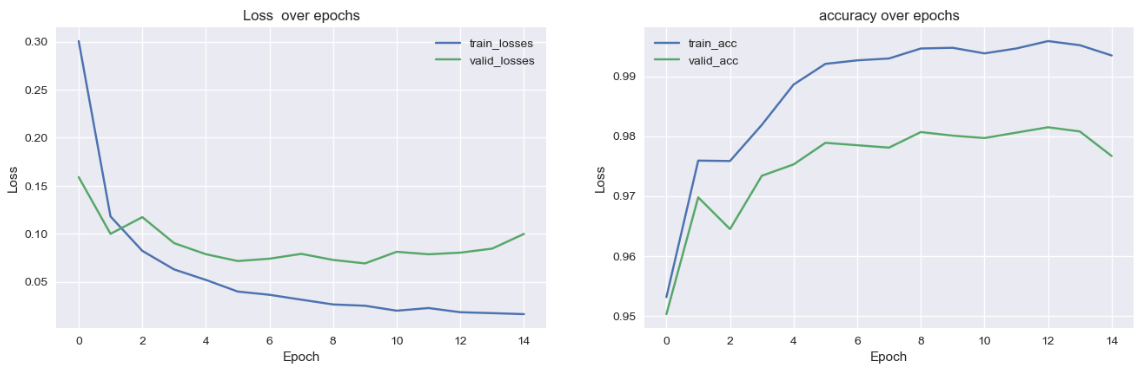
In [30]:

```
MLP_model, optimizer, performance_2 = training_loop(MLP_model, criterion, optimizer, tra
```

```
17:44:33 Epoch: 0       Train loss: 0.3006      Valid loss: 0.1590      Tr
ain accuracy: 95.31     Valid accuracy: 95.03
17:45:04 Epoch: 1       Train loss: 0.1181      Valid loss: 0.1000      Tr
ain accuracy: 97.59     Valid accuracy: 96.98
17:45:35 Epoch: 2       Train loss: 0.0823      Valid loss: 0.1174      Tr
ain accuracy: 97.59     Valid accuracy: 96.45
17:46:06 Epoch: 3       Train loss: 0.0629      Valid loss: 0.0903      Tr
ain accuracy: 98.19     Valid accuracy: 97.34
17:46:38 Epoch: 4       Train loss: 0.0520      Valid loss: 0.0788      Tr
ain accuracy: 98.86     Valid accuracy: 97.53
17:47:10 Epoch: 5       Train loss: 0.0399      Valid loss: 0.0717      Tr
ain accuracy: 99.21     Valid accuracy: 97.89
17:47:41 Epoch: 6       Train loss: 0.0365      Valid loss: 0.0741      Tr
ain accuracy: 99.27     Valid accuracy: 97.85
17:48:12 Epoch: 7       Train loss: 0.0314      Valid loss: 0.0791      Tr
ain accuracy: 99.30     Valid accuracy: 97.81
17:48:43 Epoch: 8       Train loss: 0.0265      Valid loss: 0.0728      Tr
ain accuracy: 99.47     Valid accuracy: 98.07
17:49:14 Epoch: 9       Train loss: 0.0251      Valid loss: 0.0692      Tr
ain accuracy: 99.48     Valid accuracy: 98.01
17:49:45 Epoch: 10      Train loss: 0.0200      Valid loss: 0.0813      Tr
ain accuracy: 99.38     Valid accuracy: 97.97
17:50:16 Epoch: 11      Train loss: 0.0227      Valid loss: 0.0787      Tr
ain accuracy: 99.47     Valid accuracy: 98.06
17:50:47 Epoch: 12      Train loss: 0.0184      Valid loss: 0.0804      Tr
ain accuracy: 99.59     Valid accuracy: 98.15
17:51:19 Epoch: 13      Train loss: 0.0174      Valid loss: 0.0846      Tr
ain accuracy: 99.52     Valid accuracy: 98.08
17:51:50 Epoch: 14      Train loss: 0.0164      Valid loss: 0.0999      Tr
ain accuracy: 99.35     Valid accuracy: 97.67
```

In [31]:

```
plot_performance(performance_2)
```



In [32]:

```
# We can see that MLP also works well here.
# The graph shows that there is a huge decrease of loss, and also there are both high ra
# on training and testing data accuracy
```

In [33]:

```
#3.4.1
```

In [34]:

```
#3.4.2
def find_trainable_parameter(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
```

In [35]:

```
print("number of trainable parameters of LeNet:", find_trainable_parameter(ln_model))
print("number of trainable parameters of MLP:",find_trainable_parameter(MLP_model))
```

```
number of trainable parameters of LeNet: 61706
number of trainable parameters of MLP: 280058
```

In [36]:

```
#3.4.3
#In my opinion, I think LeNet5 is the better model for predict accuracy on the test data
#We can see that as the epoches become bigger, the LeNet5 did better than MLP.
#The reason why I think LeNet5 is the better model is that
#the difference of the two methods. LeNet5 uses different small areas to work.
#However, MLP uses much more neurons to work out the problem.
```

In [37]:

```
#Statement of Collaboration
# I asked about the use of AUC with Yanghan Deng, and I also checked with him about the
# I also asked him about the count of the training parameters. I was not sure how to do
```

In [ ]:

```
```