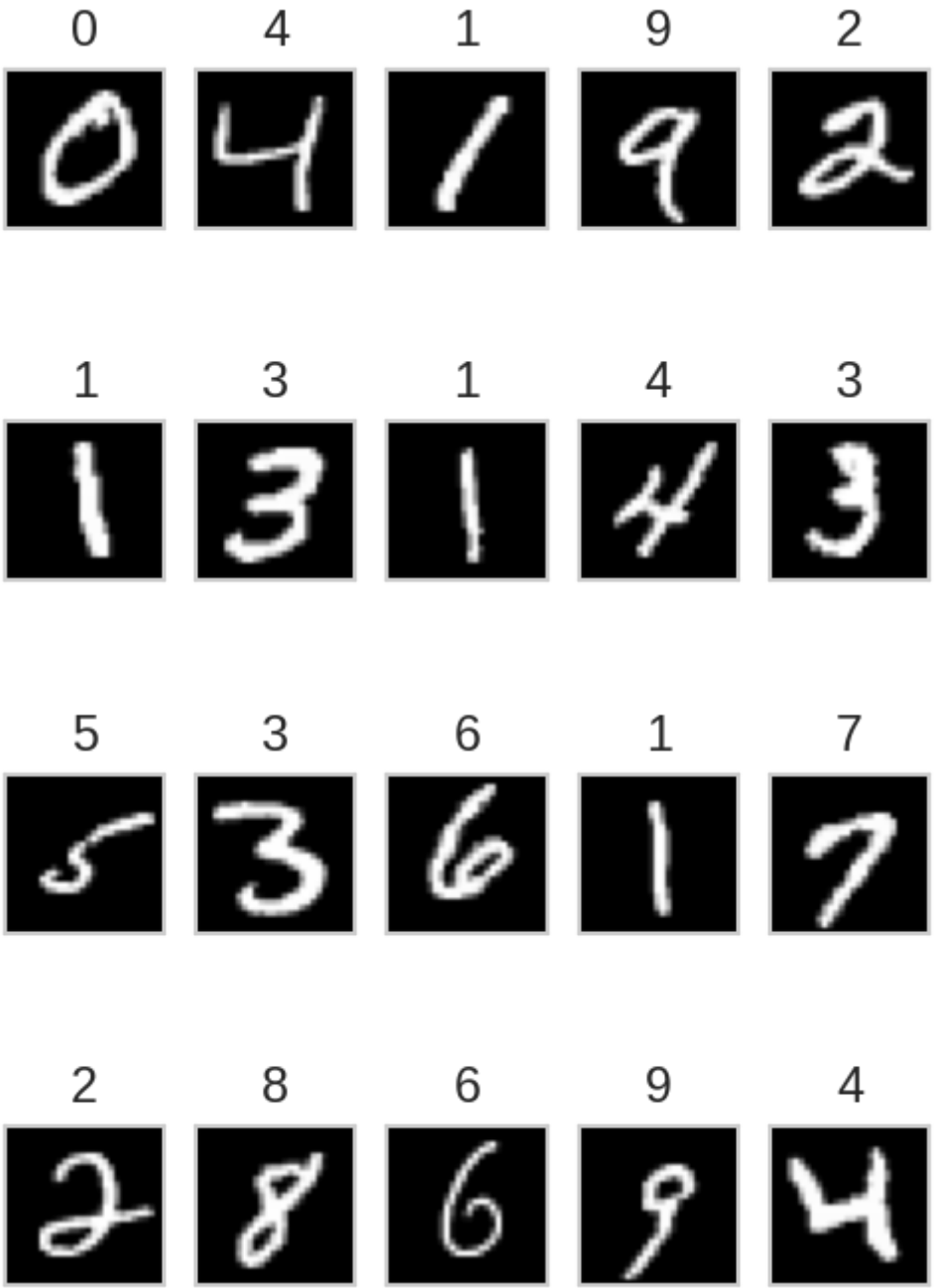
In Lab 5 we cleaned up image and extracted features (pixels values and borders, sort of) from the Sudoku image. Now we want to use the extracted features to recognize the numbers. But this time lets do it with unsupervised learning methods.

1 Recognize the numbers (you can use the MNIST data to train and test the model).

1.1 MNIST Classification using Multilayer Perceptron (MLP).

```
#! pip install torch
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import torch

plt.rcParams['figure.figsize']=(5*1.78,5)
plt.rcParams['font.size']=12
plt.rcParams['font.family']='serif'
plt.rcParams['font.serif']='Georgia'
plt.rcParams['axes.labelsize']=10
plt.rcParams['axes.titlesize']=12

plt.style.use('seaborn-whitegrid')

#cuda = torch.cuda.is_available()
#print("GPU:",cuda)



from sklearn.datasets import fetch_openml
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

mnist=fetch_openml('mnist_784', version=1)
X,y= mnist["data"], mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]


#Split and normalize train dataset

X=X_train.iloc[:,:].values / 255
Y = y_train.iloc[:].values


r = 4
```

```
c = 5
fig = plt.figure(figsize=(4,6), dpi=150)
for i in range(1, r*c+1):
    img = X[i].reshape((28,28))
    ax = fig.add_subplot(r,c,i)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.title.set_text(Y[i])
    plt.imshow(img,cmap="gray")
plt.show()
```




```
X_train = np.array(X_train)
X_test = np.array(X_test)
```

```
#Converting our data into a torch object


X_train = torch.from_numpy(np.asarray(X_train).astype('float')).type(torch.FloatTensor
y_train = torch.from_numpy(np.asarray(y_train).astype('float')).type(torch.LongTensor)

X_test = torch.from_numpy(np.asarray(X_test).astype('float')).type(torch.FloatTensor)
y_test = torch.from_numpy(np.asarray(y_test).astype('float')).type(torch.LongTensor)

'''


X_train = torch.from_numpy(np.asarray(X_train).astype('float')).type(torch.FloatTensor
y_train = torch.from_numpy(np.asarray(y_train).astype('float')).type(torch.LongTensor)

X_test = torch.from_numpy(np.asarray(X_test).astype('float')).type(torch.FloatTensor).
y_test = torch.from_numpy(np.asarray(y_test).astype('float')).type(torch.LongTensor)

'''


Validation = pd.read_csv('./validation_images.csv')



y_validation = Validation['label'].values
X_validation = Validation.iloc[:,:-1].values / 255

X_validation = torch.from_numpy(np.asarray(X_validation).astype('float')).type(torch.F
y_validation = torch.from_numpy(np.asarray(y_validation).astype('float')).type(torch.F


from torch.utils.data import TensorDataset,DataLoader

train = TensorDataset(X_train, y_train)
test = TensorDataset(X_test, y_test)

train = DataLoader(train,batch_size=1000)
test = DataLoader(test,batch_size=1000)

validation=TensorDataset(X_validation,y_validation)
validation=DataLoader(validation,batch_size=1000)


import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.linear1 = nn.Linear(784,250)
        self.linear2 = nn.Linear(250,100)
        self.linear3 = nn.Linear(100,10)
```

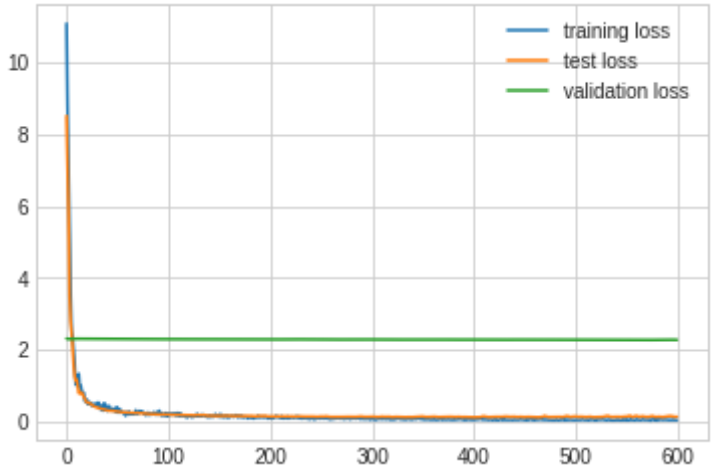
```
    def forward(self,X):
        X = F.relu(self.linear1(X))
        X = F.relu(self.linear2(X))
        X = self.linear3(X)
        return F.log_softmax(X,dim=1)

mlp=Model()
print(mlp)
```

```
 Model(
   (linear1): Linear(in_features=784, out_features=250, bias=True)
   (linear2): Linear(in_features=250, out_features=100, bias=True)
   (linear3): Linear(in_features=100, out_features=10, bias=True)
 )
```

```
from torch.optim import Adam
optimizer = Adam(mlp.parameters(),lr=1e-3)


def compute_test_loss(xtest,ytest,model):
    output=model(xtest)
    loss = F.cross_entropy(output,ytest)
    return loss


EPOCHS = 10

train_loss=[]
test_loss=[]
validation_loss=[]

mlp.train()

for epoch in range(EPOCHS):
    for batch_idx, (data, target) in enumerate(train):

        optimizer.zero_grad()

        y_pred=mlp(data)

        loss = F.cross_entropy(y_pred,target)
        train_loss.append(loss.cpu().data.item())

        loss.backward()
        optimizer.step()

        loss = compute_test_loss(X_test,y_test,mlp)
        test_loss.append(loss.cpu().data.item())

        loss=compute_test_loss(X_validation, y_validation, mlp)
        validation_loss.append(loss.cpu().data.item())
```

```
plt.plot(train_loss,label='training loss')
plt.plot(test_loss,label='test loss')
plt.plot(validation_loss,label='validation loss')
plt.legend(loc='upper right')
```

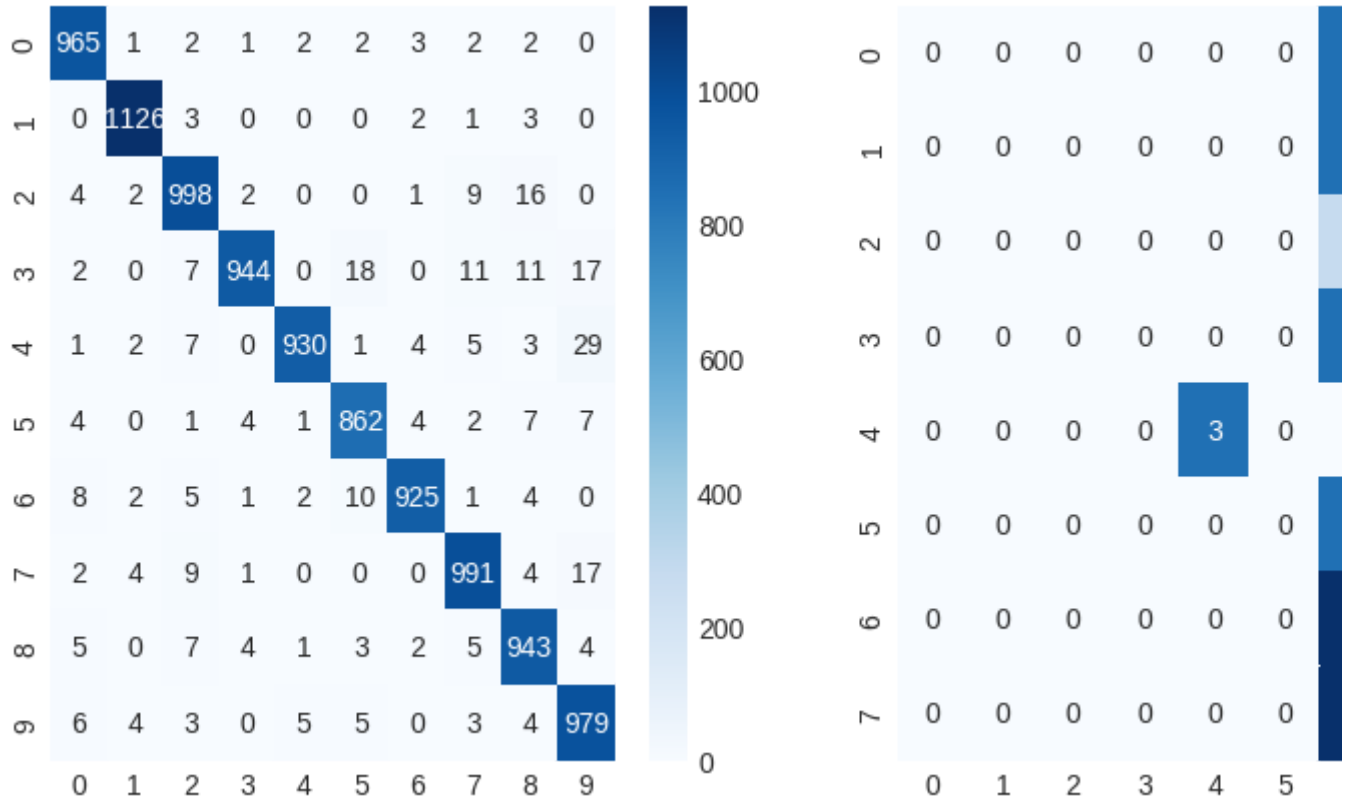
```
<matplotlib.legend.Legend at 0x7fab4f90ad50>
```




```
def predict_with_pytorch(model,val_x):

    y_preds=[]

    out=model(val_x)
    _, predicted = torch.max(out.data,1)

    for p in predicted:
        y_preds.append(p.detach().cpu().numpy().item())

    return y_preds


#Predicting validation set and plotting heatmap

pred=predict_with_pytorch(mlp,X_test)
pred_validation=predict_with_pytorch(mlp,X_validation)


X_test.numpy()
X_validation.numpy()
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test.numpy(),pred)
cm_pred=confusion_matrix(y_validation.numpy(),pred_validation)


names=['Test Confusion Matrix','Validation Confusion Matrix']
images=[cm,cm_pred]

fig, axes = plt.subplots(1,2,figsize=(10,5),dpi=100)

for ax, img, name in zip(axes.ravel(),images,names):
    sns.heatmap(img,annot=True,ax=ax,fmt='d',cmap="Blues")
```




```
from sklearn.metrics import accuracy_score

test_acc = accuracy_score(y_test.numpy(),pred)
val_acc = accuracy_score(y_validation.numpy(),pred_validation)

print("Test Accuracy:",test_acc*100,"%")
print("Validation Accuracy:",val_acc*100,"%")
```

```
Test Accuracy: 96.63000000000001 %
Validation Accuracy: 29.166666666666668 %
```

1.2 MNIST classification using Convolutional neural network(CNN)

```
X,y = mnist["data"], mnist["target"]

X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
X_train = np.array(X_train)
X_test = np.array(X_test)

print(X_test.shape)

CNN_X_train = torch.from_numpy(np.asarray(X_train).astype('float')).type(torch.FloatTe
CNN_y_train = torch.from_numpy(np.asarray(y_train).astype('float')).type(torch.LongTer

CNN_X_test = torch.from_numpy(np.asarray(X_test).astype('float')).type(torch.FloatTens
CNN_y_test = torch.from_numpy(np.asarray(y_test).astype('float')).type(torch.LongTensc

y_validation = Validation['label'].values
X_validation = Validation.iloc[:,:-1].values / 255

CNN_X_validation = torch.from_numpy(np.asarray(X_validation).astype('float')).type(tor
CNN_y_validation = torch.from_numpy(np.asarray(y_validation).astype('float')).type(tor
```

```
(10000, 784)
```

```
print('CNN_X_train shape:', CNN_X_train.shape)
print('CNN_X_test shape:', CNN_X_test.shape)
print('CNN_y_train shape:', CNN_y_train.shape)
print('CNN_y_test shape:', CNN_y_test.shape)
```

```
CNN_X_train shape: torch.Size([60000, 1, 28, 28])
CNN_X_test shape: torch.Size([0, 1, 28, 28])
CNN_y_train shape: torch.Size([60000])
CNN_y_test shape: torch.Size([10000])
```

```
from torch.utils.data import TensorDataset,DataLoader
train = TensorDataset(CNN_X_train, CNN_y_train)
test = TensorDataset(CNN_X_test, CNN_y_test)
validation = TensorDataset(CNN_X_validation, CNN_y_validation)

train = DataLoader(train,batch_size=1000)
test = DataLoader(test,batch_size=1000)
validation = DataLoader(validation,batch_size=1000)


class Model(nn.Module):
    def __init__(self):
        super(Model,self).__init__()
        self.conv1 = nn.Conv2d(1,16,5)
        self.mxp1 = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(16,24,5)
```

```
            self.mxp2 = nn.MaxPool2d(2)
            self.linear1 = nn.Linear(24 * 4 * 4, 100)
            self.linear2 = nn.Linear(100,10)

        def forward(self,x):
            X=self.mxp1(F.relu(self.conv1(x)))
            X=self.mxp2(F.relu(self.conv2(X)))
            X=X.view(-1,24*4*4)
            X=F.relu(self.linear1(X))
            X=self.linear2(X)
            return F.log_softmax(X, dim=1)

    cnn=Model()
    print(cnn)
```

```
     Model(
       (conv1): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1))
       (mxp1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fa
       (conv2): Conv2d(16, 24, kernel_size=(5, 5), stride=(1, 1))
       (mxp2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fa
       (linear1): Linear(in_features=384, out_features=100, bias=True)
       (linear2): Linear(in_features=100, out_features=10, bias=True)
     )
```

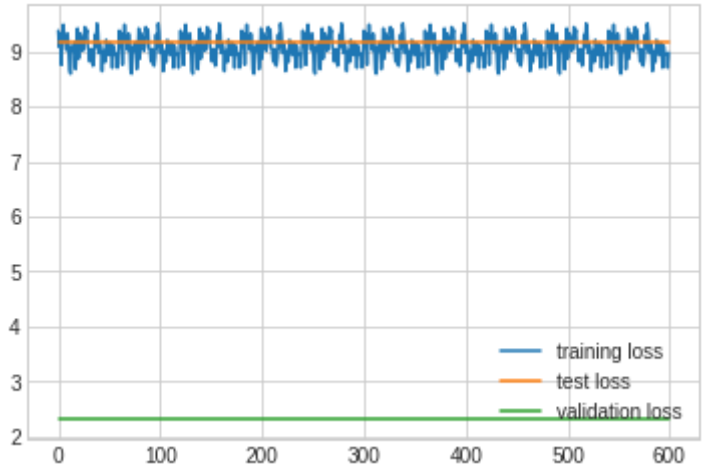
```
    EPOCHS = 10

    train_loss=[]
    test_loss=[]
    validation_loss=[]

    cnn.train()

    for epoch in range(EPOCHS):
        for batch_idx, (data, target) in enumerate(train):

            optimizer.zero_grad()

            y_pred=cnn(data)

            loss = F.cross_entropy(y_pred,target)
            train_loss.append(loss.cpu().data.item())

            loss.backward()
            optimizer.step()

            loss = compute_test_loss(CNN_X_test,CNN_y_test,cnn)
            test_loss.append(loss.cpu().data.item())

            loss=compute_test_loss(CNN_X_validation, CNN_y_validation, cnn)
            validation_loss.append(loss.cpu().data.item())
```

```
    print("Epoch: {} | train_loss: {} | test_loss: {}".format(epoch+1,train_loss[-1],t
```

```
Epoch: 1 | train_loss: 8.979212760925293 | test_loss: 9.171875953674316
Epoch: 2 | train_loss: 8.979212760925293 | test_loss: 9.171875953674316
Epoch: 3 | train_loss: 8.979212760925293 | test_loss: 9.171875953674316
Epoch: 4 | train_loss: 8.979212760925293 | test_loss: 9.171875953674316
Epoch: 5 | train_loss: 8.979212760925293 | test_loss: 9.171875953674316
Epoch: 6 | train_loss: 8.979212760925293 | test_loss: 9.171875953674316
Epoch: 7 | train_loss: 8.979212760925293 | test_loss: 9.171875953674316
Epoch: 8 | train_loss: 8.979212760925293 | test_loss: 9.171875953674316
Epoch: 9 | train_loss: 8.979212760925293 | test_loss: 9.171875953674316
Epoch: 10 | train_loss: 8.979212760925293 | test_loss: 9.171875953674316
```

```
plt.plot(train_loss,label='training loss')
plt.plot(test_loss,label='test loss')
plt.plot(validation_loss,label='validation loss')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7fab4fc67990>
```



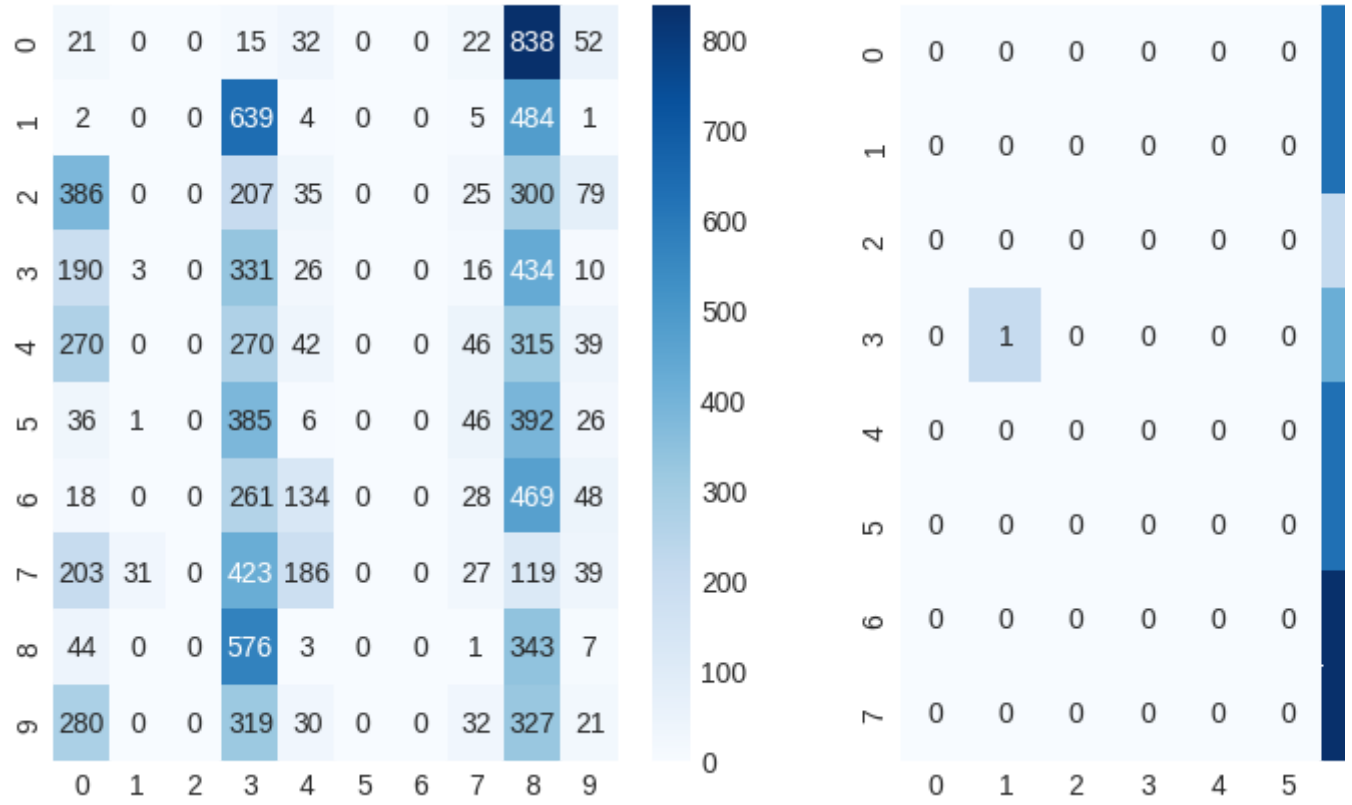

```
def predict_with_pytorch(model,val_x):

    y_preds=[]

    out=model(val_x)
    _, predicted = torch.max(out.data,1)

    for p in predicted:
        y_preds.append(p.detach().cpu().numpy().item())

    return y_preds


#Predicting validation set and plotting heatmap

pred=predict_with_pytorch(cnn,CNN_X_test)
pred_validation=predict_with_pytorch(cnn,CNN_X_validation)
```

```
CNN_X_test.numpy()
CNN_X_validation.numpy()
```

```
array([[[[0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         ...,
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.]]],


       [[[0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         ...,
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.]]],


       [[[0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         ...,
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.]]],


       ...,


       [[[0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         ...,
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.]]],


       [[[0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         ...,
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.]]],


       [[[0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         [0., 0., 0., ..., 0., 0., 0.],
         ...,
         [0., 0., 0., ..., 0., 0., 0.],
```

```
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]]]], dtype=float32)
```

```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(CNN_y_test.numpy(),pred)
cm_pred=confusion_matrix(CNN_y_validation.numpy(),pred_validation)


names=['Test Confusion Matrix','Validation Confusion Matrix']
images=[cm,cm_pred]

fig, axes = plt.subplots(1,2,figsize=(10,5),dpi=100)

for ax, img, name in zip(axes.ravel(),images,names):
    sns.heatmap(img,annot=True,ax=ax,fmt='d',cmap="Blues")
```




```
from sklearn.metrics import accuracy_score

test_acc = accuracy_score(CNN_y_test.numpy(),pred)
val_acc = accuracy_score(CNN_y_validation.numpy(),pred_validation)

print("Test Accuracy:",test_acc*100,"%")
print("Validation Accuracy:",val_acc*100,"%")
```

```
    Test Accuracy: 7.85 %
    Validation Accuracy: 16.666666666666664 %
```

1.3 MNIST classification using Recurrent neural network(RNN)

```
X,y = mnist["data"], mnist["target"]

X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
X_train = np.array(X_train)
X_test = np.array(X_test)

print(X_test.shape)

RNN_X_train = torch.from_numpy(np.asarray(X_train).astype('float')).type(torch.FloatTe
RNN_y_train = torch.from_numpy(np.asarray(y_train).astype('float')).type(torch.LongTer

RNN_X_test = torch.from_numpy(np.asarray(X_test).astype('float')).type(torch.FloatTens
RNN_y_test = torch.from_numpy(np.asarray(y_test).astype('float')).type(torch.LongTensc

y_validation = Validation['label'].values
X_validation = Validation.iloc[:,:-1].values / 255

RNN_X_validation = torch.from_numpy(np.asarray(X_validation).astype('float')).type(tor
RNN_y_validation = torch.from_numpy(np.asarray(y_validation).astype('float')).type(tor
```

```
(10000, 784)
```

```
print('RNN_X_train shape:', RNN_X_train.shape)
print('RNN_X_test shape:', RNN_X_test.shape)
print('RNN_y_train shape:', RNN_y_train.shape)
print('RNN_y_test shape:', RNN_y_test.shape)
```

```
RNN_X_train shape: torch.Size([60000, 28, 28])
RNN_X_test shape: torch.Size([10000, 28, 28])
RNN_y_train shape: torch.Size([60000])
RNN_y_test shape: torch.Size([10000])
```

```
from torch.utils.data import TensorDataset,DataLoader

train = TensorDataset(RNN_X_train, RNN_y_train)
test = TensorDataset(RNN_X_test, RNN_y_test)

train = DataLoader(train,batch_size=1000)
test = DataLoader(test,batch_size=1000)

validation=TensorDataset(RNN_X_validation,RNN_y_validation)
validation=DataLoader(validation,batch_size=1000)


class RNN(nn.Module):
```

```
    def __init__(self):
        super(RNN,self).__init__()

        self.rnn = nn.LSTM(
            input_size=28,
            hidden_size=64,
            num_layers=1,
            batch_first=True,
        )

        self.out = nn.Linear(64,10)

    def forward(self,x):
        r_out, (h_n,h_c) = self.rnn(x, None)

        out=self.out(r_out[:,-1,:])

        return out

rnn=RNN()
print(rnn)
```

```
RNN(
  (rnn): LSTM(28, 64, batch_first=True)
  (out): Linear(in_features=64, out_features=10, bias=True)
)
```

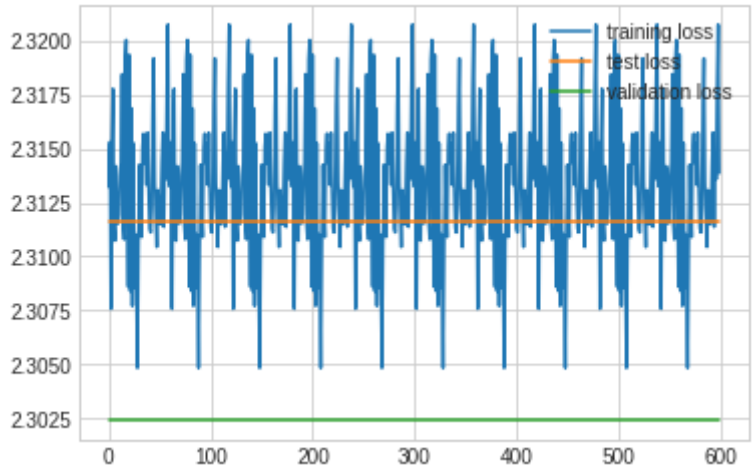
```
EPOCHS = 10

train_loss=[]
test_loss=[]
validation_loss=[]

rnn.train()

for epoch in range(EPOCHS):
    for batch_idx, (data, target) in enumerate(train):

        optimizer.zero_grad()

        y_pred=rnn(data)

        loss = F.cross_entropy(y_pred,target)
        train_loss.append(loss.cpu().data.item())

        loss.backward()
        optimizer.step()

        loss = compute_test_loss(RNN_X_test,RNN_y_test,rnn)
        test_loss.append(loss.cpu().data.item())
```

```
        loss=compute_test_loss(RNN_X_validation, RNN_y_validation, rnn)
        validation_loss.append(loss.cpu().data.item())

    print("Epoch: {} | train_loss: {} | test_loss: {}".format(epoch+1,train_loss[-1],t
```

```
Epoch: 1 | train_loss: 2.3138678073883057 | test_loss: 2.311629056930542
Epoch: 2 | train_loss: 2.3138678073883057 | test_loss: 2.311629056930542
Epoch: 3 | train_loss: 2.3138678073883057 | test_loss: 2.311629056930542
Epoch: 4 | train_loss: 2.3138678073883057 | test_loss: 2.311629056930542
Epoch: 5 | train_loss: 2.3138678073883057 | test_loss: 2.311629056930542
Epoch: 6 | train_loss: 2.3138678073883057 | test_loss: 2.311629056930542
Epoch: 7 | train_loss: 2.3138678073883057 | test_loss: 2.311629056930542
Epoch: 8 | train_loss: 2.3138678073883057 | test_loss: 2.311629056930542
Epoch: 9 | train_loss: 2.3138678073883057 | test_loss: 2.311629056930542
Epoch: 10 | train_loss: 2.3138678073883057 | test_loss: 2.311629056930542
```

```
plt.plot(train_loss,label='training loss')
plt.plot(test_loss,label='test loss')
plt.plot(validation_loss,label='validation loss')
plt.legend(loc='upper right')
```

```
<matplotlib.legend.Legend at 0x7fab4fc37c10>
```




```
def predict_with_pytorch(model,val_x):

    y_preds=[]

    out=model(val_x)
    _, predicted = torch.max(out.data,1)

    for p in predicted:
        y_preds.append(p.detach().cpu().numpy().item())

    return y_preds


#Predicting validation set and plotting heatmap
```

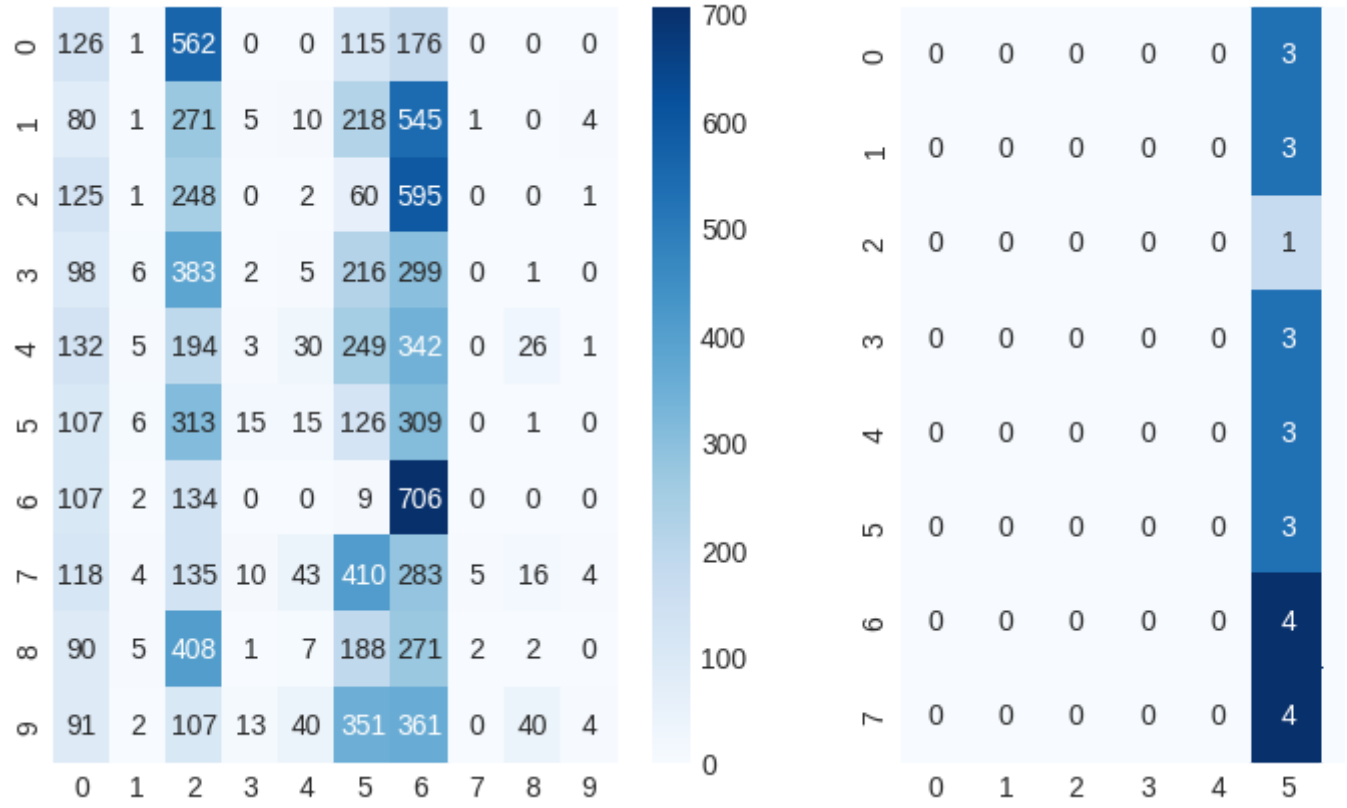
```
pred=predict_with_pytorch(rnn,RNN_X_test)
pred_validation=predict_with_pytorch(rnn,RNN_X_validation)
```

```
RNN_X_test.numpy()
RNN_X_validation.numpy()
```

```
array([[[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]],

       [[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]],

       [[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]],

       ...,

       [[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]],

       [[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]],

       [[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
```

```
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]]], dtype=float32)
```

```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(RNN_y_test.numpy(),pred)
cm_pred=confusion_matrix(RNN_y_validation.numpy(),pred_validation)


names=['Test Confusion Matrix','Validation Confusion Matrix']
images=[cm,cm_pred]

fig, axes = plt.subplots(1,2,figsize=(10,5),dpi=100)

for ax, img, name in zip(axes.ravel(),images,names):
    sns.heatmap(img,annot=True,ax=ax,fmt='d',cmap="Blues")
```




```
from sklearn.metrics import accuracy_score

test_acc = accuracy_score(RNN_y_test.numpy(),pred)
val_acc = accuracy_score(RNN_y_validation.numpy(),pred_validation)

print("Test Accuracy:",test_acc*100,"%")
print("Validation Accuracy:",val_acc*100,"%")
```

```
Test Accuracy: 12.5 %
Validation Accuracy: 12.5 %
```

Extra You can Submit it with Lab 8. - Extra Credit lab of the Semester (20 points, optional).

Solve the Sudoku puzzle. Use any Neural Network.