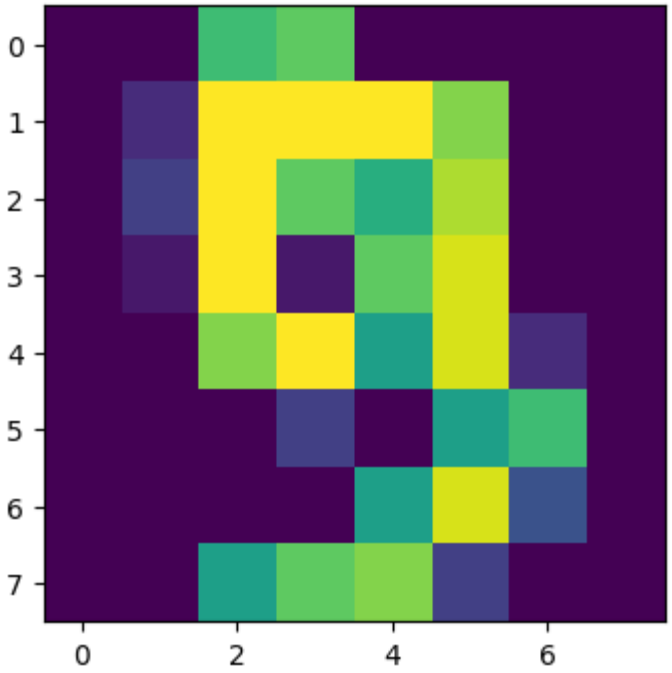
In [ ]:
```
# Install pytorch using pip
!pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu117
```

```
Looking in indexes: https://download.pytorch.org/whl/cu117
Requirement already satisfied: torch in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (2.0.0+cu117)
Requirement already satisfied: torchvision in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (0.15.1+cu117)
Requirement already satisfied: torchaudio in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (2.0.1+cu117)
Requirement already satisfied: filelock in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from torch) (3.9.0)
Requirement already satisfied: typing-extensions in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from torch) (1.11.1)
Requirement already satisfied: networkx in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from torch) (3.0)
Requirement already satisfied: jinja2 in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from torch) (3.1.2)
Requirement already satisfied: numpy in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from torchvision) (1.23.5)
Requirement already satisfied: requests in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from torchvision) (2.28.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from torchvision) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from jinja2->torch) (2.1.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from requests->torchvision) (3.1.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from requests->torchvision) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from requests->torchvision) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from requests->torchvision) (2022.12.7)
Requirement already satisfied: mpmath>=0.19 in c:\users\kalyan\win-virtualenvs\bigdata_ds\lib\site-packages (from sympy->torch) (1.2.1)
[notice] A new release of pip is available: 23.0.1 -> 23.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [ ]:
```
# import required libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt

# pytorch libraries
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# load the builtin digits dataset
digits = datasets.load_digits()
# no. of data samples
print(len(digits.data))
print(len(digits.target))

# visualize a number
plt.figure(figsize=(4,4))
plt.imshow(digits.images[9], interpolation='nearest', aspect='auto')
```

```
1797
1797
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1c888b89b50>
```



In [ ]:
```python
# count the frequency of each digit
digit_counts = {}
for digit in digits.target:
    if digit in digit_counts:
        digit_counts[digit] += 1
    else:
        digit_counts[digit] = 1

# print the counts for each digit
for digit, count in digit_counts.items():
    print(f"Digit {digit}: {count}")
```

```
Digit 0: 178
Digit 1: 182
Digit 2: 177
Digit 3: 183
Digit 4: 181
Digit 5: 182
Digit 6: 181
Digit 7: 179
Digit 8: 174
Digit 9: 180
```

In [ ]:
```
# data split
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2, random_state=42)

# convert the datasets to tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

In [ ]:
```
# Neural Network Class
class NeuralNetSklearnDigits(nn.Module):
    def __init__ (self):
        super().__init__()
        self.fc1 = nn.Linear(64, 32)
        self.fc2 = nn.Linear(32,10)

    def forward(self, X):
        X = X.view(-1, 64)
        X = torch.relu(self.fc1(X))
        X = self.fc2(X)
        return X

digits_model = NeuralNetSklearnDigits()
print (digits_model)
```

```
NeuralNetSklearnDigits(
  (fc1): Linear(in_features=64, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=10, bias=True)
)
```

In [ ]:
```
# set the adam optimizer
optimizer = optim.Adam(digits_model.parameters(), lr=0.01)

# No. of iterations/epochs
for epoch in range(100):
    optimizer.zero_grad()
    output = digits_model(X_train)
    loss = nn.CrossEntropyLoss()(output, y_train)
    loss.backward()
    optimizer.step()
```

In [ ]:
```
# Model test
y_pred = digits_model(X_test)
accuracy = accuracy_score(y_test, torch.argmax(y_pred, axis=1))
print(f"Test accuracy: {accuracy}")
```

```
Test accuracy: 0.9666666666666667
```

Extra Credit Assignment

Burn Dataset Assumptions:

- First the dataset is not labeled properly, so I have assumed that the images from 1 to 62 as Burn and the other images as Not Burn.

In [ ]:
```
# required library imports
import os
from PIL import Image

# pytorch imports
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader, random_split
```

In [ ]:
```python
# create a new class by inheriting the Dataset class.
class CustomImgDataset(Dataset):
    # constructor
    def __init__(self, path, transform=None):
        # directory where images were present
        self.path = path
        # get all the files in the path dir
        self.image_paths = os.listdir(path)
        # remove the unnecessary DS_Store file
        self.image_paths.remove(".DS_Store")
        self.transform = transform

    # modifying the inherited methods
    def __getitem__(self, index):
        # open the image using PIL
        image = Image.open(os.path.join(self.path, self.image_paths[index]))
        # use a function to set the target class
        target = self.set_target_class(self.image_paths[index])
        # transform the image
        if self.transform is not None:
            image = self.transform(image)
        return image, target

    # total no. of samples
    def __len__(self):
        return len(self.image_paths)

    # Function to set target class as Burn(1) or Not Burn(0)
    # based on the filename.
    # filename = img1-62.jpg - Burn(1) else Not Burn(0)
    def set_target_class(self, filename):
        # Prepare a list of image names - img1-62.jpg
        burnList = [ f'img{idx}.jpg' for idx in range(1, 62+1)]
        # Check if the given filename is present to set Burn
        if filename in burnList:
            return 1
        # Else Not Burn
        else:
            return 0

# Image transformations
# resize and normalize the image
```

```
transform = transforms.Compose([
    # resize
    transforms.Resize((200,200)),
    # convert to tensor
    transforms.ToTensor(),
    # normalize
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

# DIR for the burnimages dataset
path = "BurnImages"

dataset = CustomImgDataset(path, transform)
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

In [ ]:

```
# set train and test sizes
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
# Split dataset into train and test sets
train_dataset, test_dataset = random_split(dataset, [train_size, test_size],
                                           generator=torch.Generator().manual_seed(42))

# Define the data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=0)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=True, num_workers=0)
```

In [ ]:
```
# CNN - Convolution neural network
class BurnCNN(nn.Module):
    def __init__(self):
        super(BurnCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.fc1 = nn.Linear(128*50*50, 256)
        self.fc2 = nn.Linear(256, 1)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.max_pool2d(F.relu(self.bn2(self.conv2(x))), 2)
        x = F.max_pool2d(F.relu(self.bn3(self.conv3(x))), 2)
        x = x.view(-1, 128*50*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x.squeeze()
```

In [ ]:
```
# instantiate the convolution neural network
cnn = BurnCNN()

# setup loss function and optimizer
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(cnn.parameters())

# train the neural network
num_epochs = 5
# no. of iterations/epochs to loop
for epoch in range(num_epochs):
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        # zero the param gradients
        optimizer.zero_grad()
        # forward -> backward -> optimize
        outputs = cnn(inputs)
        loss = criterion(outputs, labels.float())
        loss.backward()
        optimizer.step()

        # print some loss statistics
        running_loss += loss.item()
        if i % 4 == 2:
            print(f'[{epoch+1:5d}, {i+1:5d}] loss: {running_loss/10:.3f}')
            running_loss = 0.0
```

```
[    1,     3] loss: 14.605
[    2,     3] loss: 0.540
[    3,     3] loss: 0.971
[    4,     3] loss: 0.255
[    5,     3] loss: 0.183
```

In [ ]:
```
# test the neural network
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = cnn(images)
        predicted = torch.round(torch.sigmoid(outputs))
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the {len(test_dataset)} test images: {100*correct/total:.2f}%.')
```

```
Accuracy of the network on the 18 test images: 88.89%.
```