10) Implement and apply optimization methods for neural networks (AdaGrad, RMSProp, Adam) on any relevant dataset.

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

def get_data():
  transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,),
(0.5,))])
  trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=transform)
  testset = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=transform)
  trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
  testloader = DataLoader(testset, batch_size=1000, shuffle=False)
  return trainloader, testloader

class SimpleNN(nn.Module):
  def __init__(self):
    super(SimpleNN, self).__init__()
    self.fc1 = nn.Linear(28 * 28, 128)
    self.fc2 = nn.Linear(128, 64)
    self.fc3 = nn.Linear(64, 10)
    self.relu = nn.ReLU()

  def forward(self, x):
    x = x.view(-1, 28 * 28)
    x = self.relu(self.fc1(x))
    x = self.relu(self.fc2(x))
    x = self.fc3(x)
    return x

def train_model(optimizer_type, trainloader, testloader):
  model = SimpleNN()
```

```
    criterion = nn.CrossEntropyLoss()
    optimizer = optimizer_type(model.parameters(), lr=0.01)

    for epoch in range(5):
        model.train()
        for images, labels in trainloader:
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
        print(f"Epoch {epoch+1} completed")

    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in testloader:
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f"Accuracy with {optimizer_type.__name__}: {accuracy:.2f}%")

trainloader, testloader = get_data()
optimizers = [optim.Adagrad, optim.RMSprop, optim.Adam]

for opt in optimizers:
    train_model(opt, trainloader, testloader)
```

Output:

Epoch 1 completed

Epoch 2 completed

Epoch 3 completed

Epoch 4 completed

Epoch 5 completed

Accuracy with Adagrad: 94.78%

Epoch 1 completed

Epoch 2 completed

Epoch 3 completed

Epoch 4 completed

Epoch 5 completed

Accuracy with RMSprop: 93.17%

Epoch 1 completed

Epoch 2 completed

Epoch 3 completed

Epoch 4 completed

Epoch 5 completed

Accuracy with Adam: 94.48%

- **Data preprocessing**
- **Model definition (simple feedforward network)**
- **Training using different optimizers**
- **Performance comparison**

Apply, train and visualize Different deep CNN architectures like LeNet, AlexNet, VGG, PlacesNet, on MNIST datasets.

<u>This program uses LeNet, AlexNet on MNIST dataset</u>

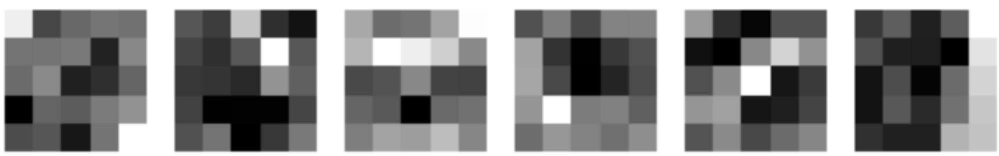
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np

# Load MNIST dataset
def get_data():
    transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
    trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
    testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
    trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
    testloader = DataLoader(testset, batch_size=1000, shuffle=False)
    return trainloader, testloader

# Define CNN Architectures
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16*4*4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(2, 2)

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
```

```
        x = x.view(-1, 16*4*4)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x

class AlexNet(nn.Module):
    def __init__(self):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Linear(256 * 3 * 3, 4096),
            nn.ReLU(),
            nn.Linear(4096, 4096),
            nn.ReLU(),
            nn.Linear(4096, 10),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x

# Train and evaluate model
def train_model(model, trainloader, testloader, optimizer_type):
    model = model()
    criterion = nn.CrossEntropyLoss()
    optimizer = optimizer_type(model.parameters(), lr=0.01)
```

```
  for epoch in range(5):
    model.train()
    for images, labels in trainloader:
      optimizer.zero_grad()
      outputs = model(images)
      loss = criterion(outputs, labels)
      loss.backward()
      optimizer.step()
    print(f"Epoch {epoch+1} completed")

  model.eval()
  correct = 0
  total = 0
  with torch.no_grad():
    for images, labels in testloader:
      outputs = model(images)
      _, predicted = torch.max(outputs, 1)
      total += labels.size(0)
      correct += (predicted == labels).sum().item()

  accuracy = 100 * correct / total
  print(f"Accuracy with {model.__class__.__name__}: {accuracy:.2f}%")
  return model

# Visualization Function
def visualize_filters(model):
  with torch.no_grad():
    for name, param in model.named_parameters():
      if 'conv' in name and param.requires_grad:
        filters = param.cpu().numpy()
        fig, axes = plt.subplots(1, min(6, filters.shape[0]))
        for i, ax in enumerate(axes):
          ax.imshow(filters[i, 0], cmap='gray')
          ax.axis('off')
        plt.show()
        break

# Load data
trainloader, testloader = get_data()

# Train models
models = [LeNet, AlexNet]
optimizers = [optim.Adam]
for model in models:
```

```
for opt in optimizers:
    trained_model = train_model(model, trainloader, testloader, opt)
    visualize_filters(trained_model)
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 55.8MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.70MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 14.0MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 6.21MB/s]
Epoch 1 completed
Epoch 2 completed
Epoch 3 completed
Epoch 4 completed
Epoch 5 completed
Accuracy with LeNet: 98.17%
```



```
Epoch 1 completed
Epoch 2 completed
Epoch 3 completed
Epoch 4 completed
Epoch 5 completed
Accuracy with AlexNet: 10.09%
```

This program uses VGG, PlacesNet on MNIST dataset

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
```

```
import cv2

# Load MNIST Dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Preprocess Data
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)

# Convert grayscale to 3 channels for pretrained models
x_train = np.repeat(x_train, 3, axis=-1)
x_test = np.repeat(x_test, 3, axis=-1)

# Resize images to 32x32 for VGG-16 compatibility
x_train_resized = np.array([cv2.resize(img, (32, 32)) for img in x_train])
x_test_resized = np.array([cv2.resize(img, (32, 32)) for img in x_test])

# Normalize data
x_train_resized, x_test_resized = x_train_resized / 255.0, x_test_resized / 255.0
x_train, x_test = x_train / 255.0, x_test / 255.0

# Convert labels to categorical
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

# Define VGG-16 Model
base_model_vgg = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
for layer in base_model_vgg.layers:
    layer.trainable = False

x = Flatten()(base_model_vgg.output)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)

vgg_model = Model(inputs=base_model_vgg.input, outputs=x)
vgg_model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train VGG-16
vgg_model.fit(x_train_resized, y_train_cat, epochs=5, batch_size=64,
validation_data=(x_test_resized, y_test_cat))
```

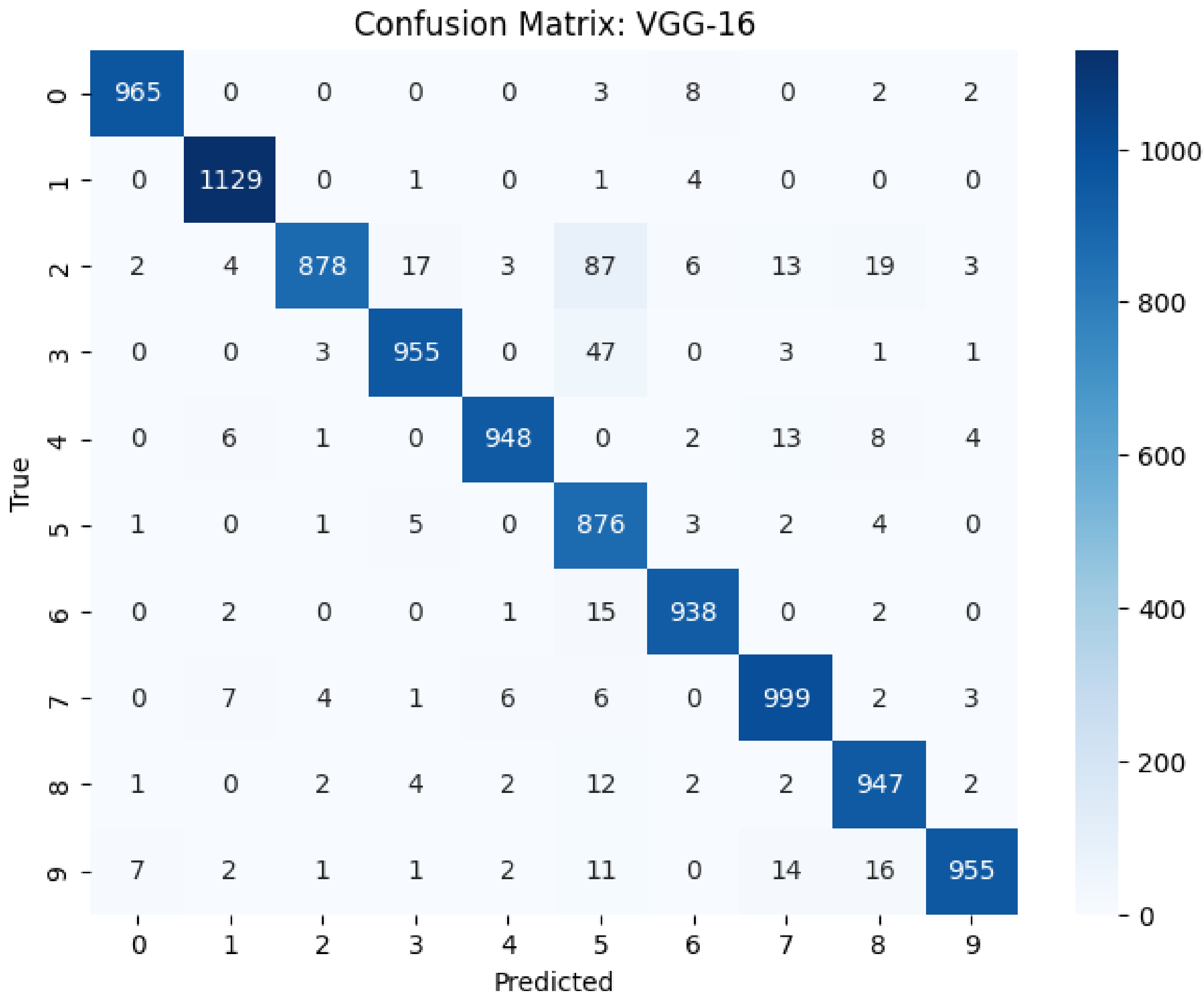
```
# Define PlacesNet-like CNN
input_layer = Input(shape=(28, 28, 3))
x = Conv2D(64, (3,3), activation='relu', padding='same')(input_layer)
x = MaxPooling2D((2,2))(x)
x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
x = MaxPooling2D((2,2))(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)

placesnet_model = Model(inputs=input_layer, outputs=x)
placesnet_model.compile(optimizer=Adam(), loss='categorical_crossentropy',
metrics=['accuracy'])

# Train PlacesNet
placesnet_model.fit(x_train, y_train_cat, epochs=5, batch_size=64, validation_data=(x_test,
y_test_cat))

# Evaluate and Visualize Results
def plot_confusion_matrix(model, x_test, y_test, title):
    y_pred = np.argmax(model.predict(x_test), axis=1)
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title(f'Confusion Matrix: {title}')
    plt.show()

# Confusion Matrices
plot_confusion_matrix(vgg_model, x_test_resized, y_test, "VGG-16")
plot_confusion_matrix(placesnet_model, x_test, y_test, "PlacesNet")

# Display Feature Maps
def visualize_feature_maps(model, x_sample):
    layer_outputs = [layer.output for layer in model.layers if isinstance(layer, Conv2D)]
    activation_model = Model(inputs=model.input, outputs=layer_outputs)
    activations = activation_model.predict(np.expand_dims(x_sample, axis=0))
    for i, activation in enumerate(activations[:3]):  # Show first 3 layers
```

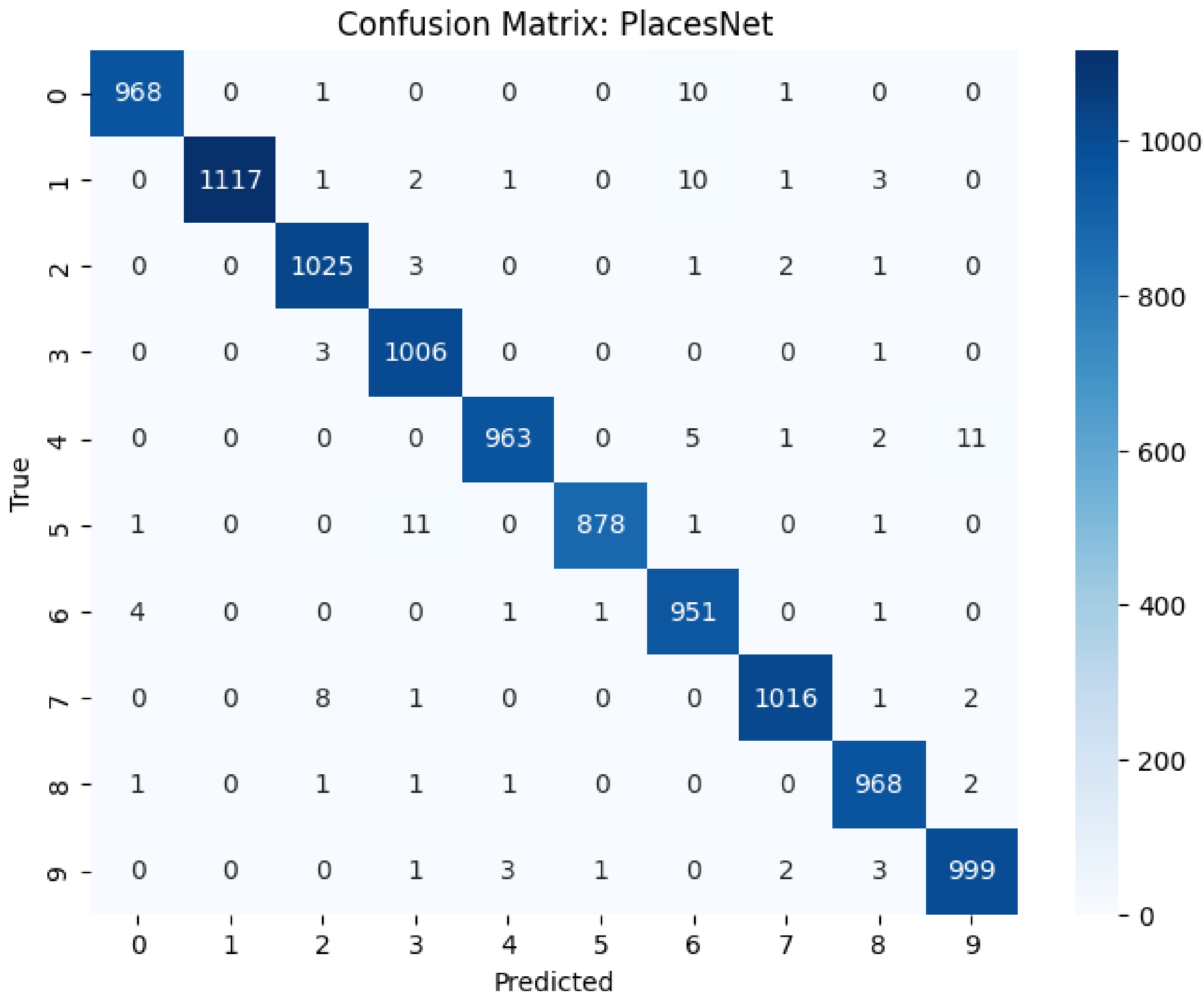
```
    plt.figure(figsize=(10, 5))
    for j in range(min(activation.shape[-1], 6)):  # Show first 6 filters
        plt.subplot(1, 6, j+1)
        plt.imshow(activation[0, :, :, j], cmap='viridis')
        plt.axis('off')
    plt.show()

# Show feature maps of first test image
visualize_feature_maps(vgg_model, x_test_resized[0])
visualize_feature_maps(placesnet_model, x_test[0])
```

Output:

938/938 ━━━━━━━━━━━━━━━━━━━━ 204s

205ms/step - accuracy: 0.9961 - loss: 0.0109 - val_accuracy: 0.9891 - val_loss: 0.0339


Confusion Matrix: VGG-16
True
Predicted
0 1 2 3 4 5 6 7 8 9
0 200 400 600 800 1000
965 0 0 0 0 3 8 0 2 2
0 1129 0 1 0 1 4 0 0 0
2 4 878 17 3 87 6 13 19 3
0 0 3 955 0 47 0 3 1 1
0 6 1 0 948 0 2 13 8 4
1 0 1 5 0 876 3 2 4 0
0 2 0 0 1 15 938 0 2 0
0 7 4 1 6 6 0 999 2 3
1 0 2 4 2 12 2 2 947 2
7 2 1 1 2 11 0 14 16 955


Confusion Matrix: PlacesNet
True
Predicted
0 1 2 3 4 5 6 7 8 9
968 0 1 0 0 0 10 1 0 0
0 1117 1 2 1 0 10 1 3 0
0 0 1025 3 0 0 1 2 1 0
0 0 3 1006 0 0 0 0 1 0
0 0 0 0 963 0 5 1 2 11
1 0 0 11 0 878 1 0 1 0
4 0 0 0 1 1 951 0 1 0
0 0 8 1 0 0 0 1016 1 2
1 0 1 1 1 0 0 0 968 2
0 0 0 1 3 1 0 2 3 999
0
200
400
600
800
1000