In [ ]:
```
import pandas as pd
import numpy as np
import torch
from sklearn.preprocessing import StandardScaler
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from collections import OrderedDict
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from torch.utils.data import DataLoader, TensorDataset
import torchvision.transforms as transforms
import time
import torchvision
from torchvision import transforms
from torchvision import datasets
import seaborn as sns
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
```

In [ ]:
```
# Check for GPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
if torch.cuda.is_available():
    device_name = torch.cuda.get_device_name(0)
    print(device_name)
print(f"Using device: {device}")
```

```
NVIDIA GeForce RTX 4050 Laptop GPU
Using device: cuda:0
```

In [ ]:
```
file_id = '1yvScnvrecfBf0cSLq76kRENXdi2_N-lN'

url = f'https://drive.google.com/uc?id={file_id}'
```

In [ ]:
```
# Load your dataset
housing_data = pd.DataFrame(pd.read_csv(url))
#housing_data.head()
```

In [ ]:
```
# Replace 'yes' and 'no' with 1 and 0
housing_data.replace({'yes': 1, 'no': 0}, inplace=True)
#housing_data.head(10)
```

In [ ]:
```
# Checking Null values
#housing_data.isnull().sum()*100/housing_data.shape[0]
```

In [ ]:
```
# features for training
features = [col for col in housing_data.columns if col != 'price' and col != 'furni

# Extract input and output values for training
x = housing_data[features].values
y = housing_data['price'].values.reshape(-1,1)

print(features), y.shape
```

```
['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom', 'basement', 'h
otwaterheating', 'airconditioning', 'parking', 'prefarea']
```

Out[ ]: (None, (545, 1))

In [ ]:
```
# Split data: validation set = 20 percent
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2, random_state

# Standardize input features
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_val_scaled = scaler.transform(x_val)

# Convert to Tensor and move to device (gpu)
x_train_tensor = torch.tensor(x_train_scaled, dtype=torch.float32)
x_val_tensor = torch.tensor(x_val_scaled, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
y_val_tensor = torch.tensor(y_val, dtype=torch.float32)
```

In [ ]:
```
def train_model(n_epochs, optimizer, model, loss_fn, x_train, x_val, y_train, y_val
    train_losses = []
    val_losses = []

    for epoch in range(1, n_epochs + 1):
        model.train()
        optimizer.zero_grad()
        outputs = model(x_train)
        loss = loss_fn(outputs, y_train)
        loss.backward()
        optimizer.step()

        #train_preds = outputs.argmax(dim=1)

        model.eval()
        with torch.no_grad():
            val_outputs = model(x_val)
            val_loss = loss_fn(val_outputs, y_val)
            #val_preds = val_outputs.argmax(dim=1)

        # Calculate metrics for training data
        #train_accuracy = accuracy_score(y_train.cpu(), train_preds.cpu())

        # Calculate metrics for validation data
        #val_accuracy = accuracy_score(y_val.cpu(), val_preds.cpu())

        if epoch == 1 or epoch % 500 == 0:
            print(f"Epoch {epoch}/{n_epochs}, Training loss: {loss.item():.4f}, Val
            #print(f"Training Accuracy: {train_accuracy:.4f}")
            #print(f"Validation Accuracy: {val_accuracy:.4f}")

        train_losses.append(loss.detach().item())
        val_losses.append(val_loss.detach().item())

    return train_losses, val_losses
```

In [ ]:
```
# Initialize the model
neuron_count = 32
input = x_train_tensor.shape[1]

# Instantiate the model
model1 = nn.Sequential(OrderedDict([
    ('hidden_linear', nn.Linear(input, neuron_count)),
    ('hidden_activation', nn.Tanh()),
    ('output_linear', nn.Linear(neuron_count, 1))
]))

optimizer1 = optim.SGD(model1.parameters(), lr=0.001)
n_epochs = 5000
loss_fn = nn.MSELoss()

train_losses1, val_losses1 = train_model(
    n_epochs = n_epochs,
    optimizer = optimizer1,
    model = model1.to(device=device, non_blocking=True),
    loss_fn = loss_fn,
    x_train = x_train_tensor.to(device=device, non_blocking=True),
    x_val = x_val_tensor.to(device=device, non_blocking=True),
    y_train = y_train_tensor.to(device=device, non_blocking=True),
    y_val = y_val_tensor.to(device=device, non_blocking=True))

total_params = sum(p.numel() for p in model1.parameters() if p.requires_grad)
print(f"Total number of trainable parameters: {total_params}")
```

```
Epoch 1/5000, Training loss: 25234793037824.0000, Validation loss: 30076301213696.00
00
Epoch 500/5000, Training loss: 1787692449792.0000, Validation loss: 2696574402560.00
00
Epoch 1000/5000, Training loss: 1578902355968.0000, Validation loss: 2526409916416.0
000
Epoch 1500/5000, Training loss: 1485081542656.0000, Validation loss: 2447122890752.0
000
Epoch 2000/5000, Training loss: 1435041398784.0000, Validation loss: 2405186142208.0
000
Epoch 2500/5000, Training loss: 1404018622464.0000, Validation loss: 2380270403584.0
000
Epoch 3000/5000, Training loss: 1382624526336.0000, Validation loss: 2363969765376.0
000
Epoch 3500/5000, Training loss: 1366860496896.0000, Validation loss: 2352475799552.0
000
Epoch 4000/5000, Training loss: 1354768187392.0000, Validation loss: 2343894253568.0
000
Epoch 4500/5000, Training loss: 1345246593024.0000, Validation loss: 2337199620096.0
000
Epoch 5000/5000, Training loss: 1337604833280.0000, Validation loss: 2331794735104.0
000
Total number of trainable parameters: 417
```

In [ ]:
```
# Instantiate the model
model2 = nn.Sequential(
    nn.Linear(x_train_tensor.shape[1], 32),
    nn.Tanh(),
```

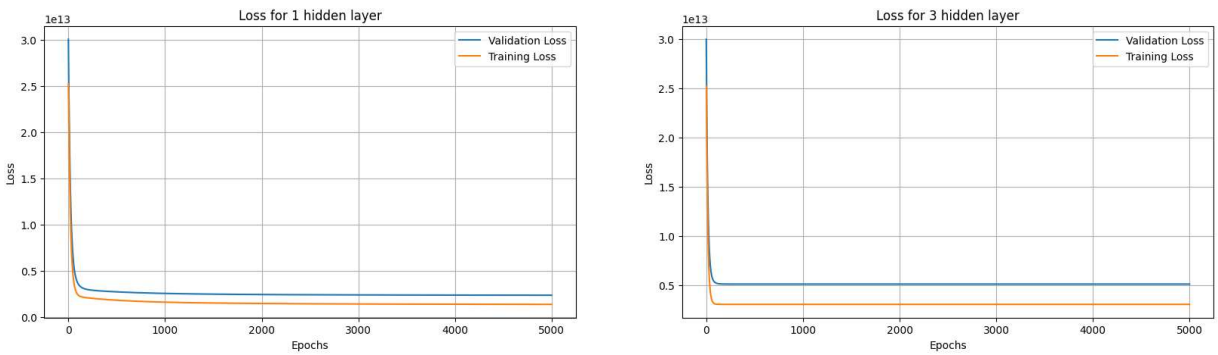
```
    nn.Linear(32, 64),
    nn.Tanh(),
    nn.Linear(64, 16),
    nn.Tanh(),
    nn.Linear(16, 1))

optimizer2 = optim.SGD(model2.parameters(), lr=0.001)
n_epochs = 5000
loss_fn = nn.MSELoss()

# Train the model
train_losses2, val_losses2 = train_model(
    n_epochs = n_epochs,
    optimizer = optimizer2,
    model = model2.to(device, non_blocking=True),
    loss_fn = loss_fn,
    x_train = x_train_tensor.to(device, non_blocking=True),
    x_val = x_val_tensor.to(device, non_blocking=True),
    y_train = y_train_tensor.to(device, non_blocking=True),
    y_val = y_val_tensor.to(device, non_blocking=True))

total_params = sum(p.numel() for p in model2.parameters() if p.requires_grad)
print(f"Total number of trainable parameters: {total_params}")
```

```
Epoch 1/5000, Training loss: 25234793037824.0000, Validation loss: 30002403868672.00
00
Epoch 500/5000, Training loss: 3083392253952.0000, Validation loss: 5145181028352.00
00
Epoch 1000/5000, Training loss: 3083392253952.0000, Validation loss: 5145181028352.0
000
Epoch 1500/5000, Training loss: 3083392253952.0000, Validation loss: 5145181028352.0
000
Epoch 2000/5000, Training loss: 3083392253952.0000, Validation loss: 5145181028352.0
000
Epoch 2500/5000, Training loss: 3083392253952.0000, Validation loss: 5145181028352.0
000
Epoch 3000/5000, Training loss: 3083392253952.0000, Validation loss: 5145181028352.0
000
Epoch 3500/5000, Training loss: 3083392253952.0000, Validation loss: 5145181028352.0
000
Epoch 4000/5000, Training loss: 3083392253952.0000, Validation loss: 5145181028352.0
000
Epoch 4500/5000, Training loss: 3083392253952.0000, Validation loss: 5145181028352.0
000
Epoch 5000/5000, Training loss: 3083392253952.0000, Validation loss: 5145181028352.0
000
Total number of trainable parameters: 3553
```

In [ ]:

```
plt.figure(figsize=(20,5))

#plt.style.use('seaborn-darkgrid')

plt.subplot(1,2,1)
# Plot both training and validation loss
plt.plot(val_losses1, label='Validation Loss')
plt.plot(train_losses1, label='Training Loss')
```

```
#plt.grid(color='white', linestyle='-', linewidth=0.5)
plt.title('Loss for 1 hidden layer')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()

plt.subplot(1, 2, 2)
# Plot both training and validation loss
plt.plot(val_losses2, label='Validation Loss')
plt.plot(train_losses2, label='Training Loss')
plt.title('Loss for 3 hidden layer')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

#plt.tight_layout()
plt.show()
```



In [ ]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
if torch.cuda.is_available:
    device_name = torch.cuda.get_device_name(0)
    print(device_name)
print(f"Using device: {device}")
```

```
NVIDIA GeForce RTX 4050 Laptop GPU
Using device: cuda:0
```

In [ ]:
```
# Load and transform the CIFAR-10 dataset
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) # Normalize the images
])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,download=True, tr
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True, nu

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, t
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False, num
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [ ]:
```
# Define the model
model3 = nn.Sequential(
```

```
    nn.Linear(3 * 32 * 32, 512),
    nn.Tanh(),
    nn.Linear(512, 10),
    nn.LogSoftmax(dim=1)
)
model3.to(device, non_blocking=True)

# Define loss function and optimizer
loss_fn = nn.NLLLoss()
optimizer = optim.SGD(model3.parameters(), lr=1e-2)

# Training loop
n_epochs = 100
start_time = time.time()

all_preds = []
all_labels = []

for epoch in range(n_epochs):
    model3.train()
    total_loss = 0
    for imgs, labels in trainloader:
        imgs, labels = imgs.to(device, non_blocking=True), labels.to(device, non_bl
        #print("Batch Device:", imgs.device)
        #print("Model device:", next(model3.parameters()).device)
        optimizer.zero_grad()
        outputs = model3(imgs.view(imgs.size(0), -1))
        loss = loss_fn(outputs, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / len(trainloader)
    print(f"Epoch: {epoch}/{n_epochs}, Average Loss: {avg_loss}")

training_time = time.time() - start_time
print(f"Training completed in {training_time:.2f} seconds.")

# Evaluate the network
correct = 0
total = 0
model3.eval()
with torch.no_grad():
    for imgs, labels in testloader:
        imgs, labels = imgs.to(device, non_blocking=True), labels.to(device, non_bl
        outputs = model3(imgs.view(imgs.size(0), -1))
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        # Store predictions and actual labels
        all_preds.extend(predicted.view(-1).tolist())
        all_labels.extend(labels.view(-1).tolist())

accuracy = 100 * correct / total
```

```
print(f'Accuracy of the network on the test images: {accuracy:.2f} %')

# Calculate the confusion matrix
conf_matrix = confusion_matrix(all_labels, all_preds)

# Calculate the F1 score
f1Score = f1_score(all_labels, all_preds, average='weighted')

#print("Confusion Matrix:\n", conf_matrix)
print("F1 Score: ", f1Score)
```

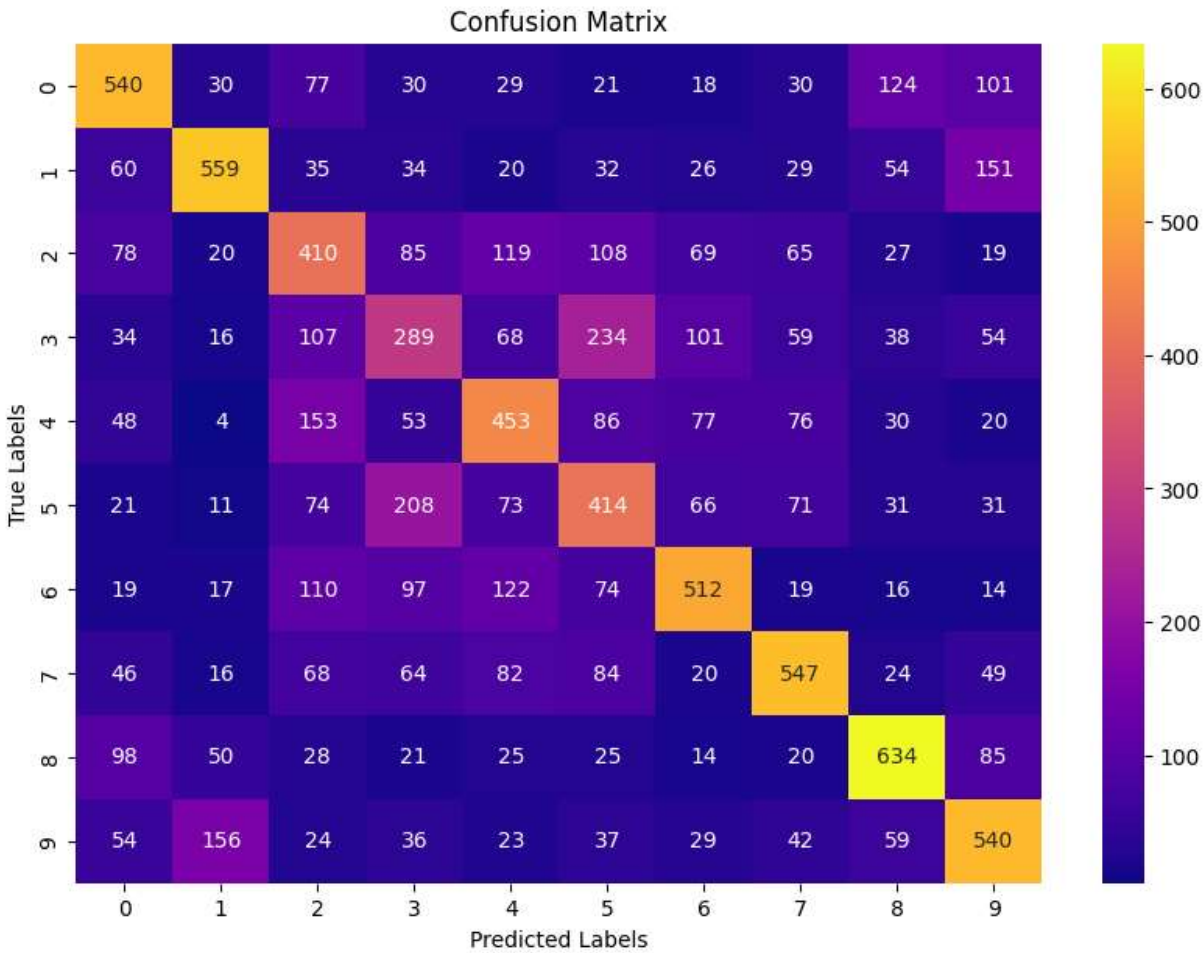
```
Epoch: 0, Average Loss: 1.8726207423393073
Epoch: 1, Average Loss: 1.7574138958435839
Epoch: 2, Average Loss: 1.7166181420121351
Epoch: 3, Average Loss: 1.687547467248824
Epoch: 4, Average Loss: 1.662034405009521
Epoch: 5, Average Loss: 1.6392102517435312
Epoch: 6, Average Loss: 1.6170653606314793
Epoch: 7, Average Loss: 1.5975598534354774
Epoch: 8, Average Loss: 1.5782244071326292
Epoch: 9, Average Loss: 1.5575243583725542
Epoch: 10, Average Loss: 1.538543203755108
Epoch: 11, Average Loss: 1.5186534027004486
Epoch: 12, Average Loss: 1.5002915818063194
Epoch: 13, Average Loss: 1.4803029414637925
Epoch: 14, Average Loss: 1.4615467233426125
Epoch: 15, Average Loss: 1.4420576298328311
Epoch: 16, Average Loss: 1.4229911712124526
Epoch: 17, Average Loss: 1.4033585814258935
Epoch: 18, Average Loss: 1.3849929612310952
Epoch: 19, Average Loss: 1.3668271665225553
Epoch: 20, Average Loss: 1.348460031135003
Epoch: 21, Average Loss: 1.3297849936253578
Epoch: 22, Average Loss: 1.3113670867422353
Epoch: 23, Average Loss: 1.2936730387113284
Epoch: 24, Average Loss: 1.2750172301021683
Epoch: 25, Average Loss: 1.2566372563162118
Epoch: 26, Average Loss: 1.238628781207687
Epoch: 27, Average Loss: 1.2200470063692468
Epoch: 28, Average Loss: 1.2028891451828314
Epoch: 29, Average Loss: 1.1846515701707367
Epoch: 30, Average Loss: 1.1659728701004897
Epoch: 31, Average Loss: 1.1489826711394904
Epoch: 32, Average Loss: 1.1317036337864674
Epoch: 33, Average Loss: 1.1134898498692476
Epoch: 34, Average Loss: 1.0969361942594924
Epoch: 35, Average Loss: 1.0798755533554976
Epoch: 36, Average Loss: 1.0616516061603565
Epoch: 37, Average Loss: 1.0451182152151757
Epoch: 38, Average Loss: 1.0269821934840258
Epoch: 39, Average Loss: 1.0099709428789672
Epoch: 40, Average Loss: 0.9932103772145098
Epoch: 41, Average Loss: 0.9762300607341025
Epoch: 42, Average Loss: 0.9588560007722177
Epoch: 43, Average Loss: 0.9419096090909466
Epoch: 44, Average Loss: 0.9257994730149388
Epoch: 45, Average Loss: 0.9079081539607718
Epoch: 46, Average Loss: 0.8911336087204916
Epoch: 47, Average Loss: 0.8753155202359495
Epoch: 48, Average Loss: 0.8585169611074736
Epoch: 49, Average Loss: 0.8444900539372583
Epoch: 50, Average Loss: 0.8265118095880885
Epoch: 51, Average Loss: 0.811617812308509
Epoch: 52, Average Loss: 0.7959503612250013
Epoch: 53, Average Loss: 0.7801992637879404
Epoch: 54, Average Loss: 0.7645836778156593
Epoch: 55, Average Loss: 0.7491541687408676
```

```
Epoch: 56, Average Loss: 0.733862681271475
Epoch: 57, Average Loss: 0.7199405754161308
Epoch: 58, Average Loss: 0.7043724438875837
Epoch: 59, Average Loss: 0.6899133931721568
Epoch: 60, Average Loss: 0.6759810645485778
Epoch: 61, Average Loss: 0.6606448741672594
Epoch: 62, Average Loss: 0.6476702530060887
Epoch: 63, Average Loss: 0.6333924770126562
Epoch: 64, Average Loss: 0.6206535829607484
Epoch: 65, Average Loss: 0.606469107139141
Epoch: 66, Average Loss: 0.5933684698684746
Epoch: 67, Average Loss: 0.5807639098990603
Epoch: 68, Average Loss: 0.5672137731938716
Epoch: 69, Average Loss: 0.5551803540009672
Epoch: 70, Average Loss: 0.54299872896403
Epoch: 71, Average Loss: 0.5312868138332196
Epoch: 72, Average Loss: 0.5181551314985661
Epoch: 73, Average Loss: 0.5065667027478937
Epoch: 74, Average Loss: 0.4944095587562722
Epoch: 75, Average Loss: 0.48266399238268126
Epoch: 76, Average Loss: 0.47202274531049804
Epoch: 77, Average Loss: 0.46043043143456547
Epoch: 78, Average Loss: 0.44976506273612343
Epoch: 79, Average Loss: 0.4405104212291405
Epoch: 80, Average Loss: 0.42849425289332105
Epoch: 81, Average Loss: 0.4190553815849602
Epoch: 82, Average Loss: 0.4091433201299604
Epoch: 83, Average Loss: 0.39974508512660367
Epoch: 84, Average Loss: 0.3894022068442286
Epoch: 85, Average Loss: 0.37989303376287453
Epoch: 86, Average Loss: 0.37146276763409297
Epoch: 87, Average Loss: 0.3618564001091606
Epoch: 88, Average Loss: 0.3529305958267673
Epoch: 89, Average Loss: 0.34520097124530835
Epoch: 90, Average Loss: 0.3350210589597292
Epoch: 91, Average Loss: 0.3276068205418794
Epoch: 92, Average Loss: 0.3200825403261063
Epoch: 93, Average Loss: 0.3120011741014393
Epoch: 94, Average Loss: 0.3042693255311998
Epoch: 95, Average Loss: 0.2967721684014096
Epoch: 96, Average Loss: 0.2894299868351358
Epoch: 97, Average Loss: 0.28137250076931763
Epoch: 98, Average Loss: 0.2752238547672396
Epoch: 99, Average Loss: 0.2685746366677382
Training completed in 490.09 seconds.
Accuracy of the network on the test images: 48.98 %
F1 Score:  0.4905981864871472
```

In [ ]:

```
conf_matrix = confusion_matrix(all_labels, all_preds)

# Plotting the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='plasma')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
```

```
plt.ylabel('True Labels')
plt.show()
```



In [ ]:

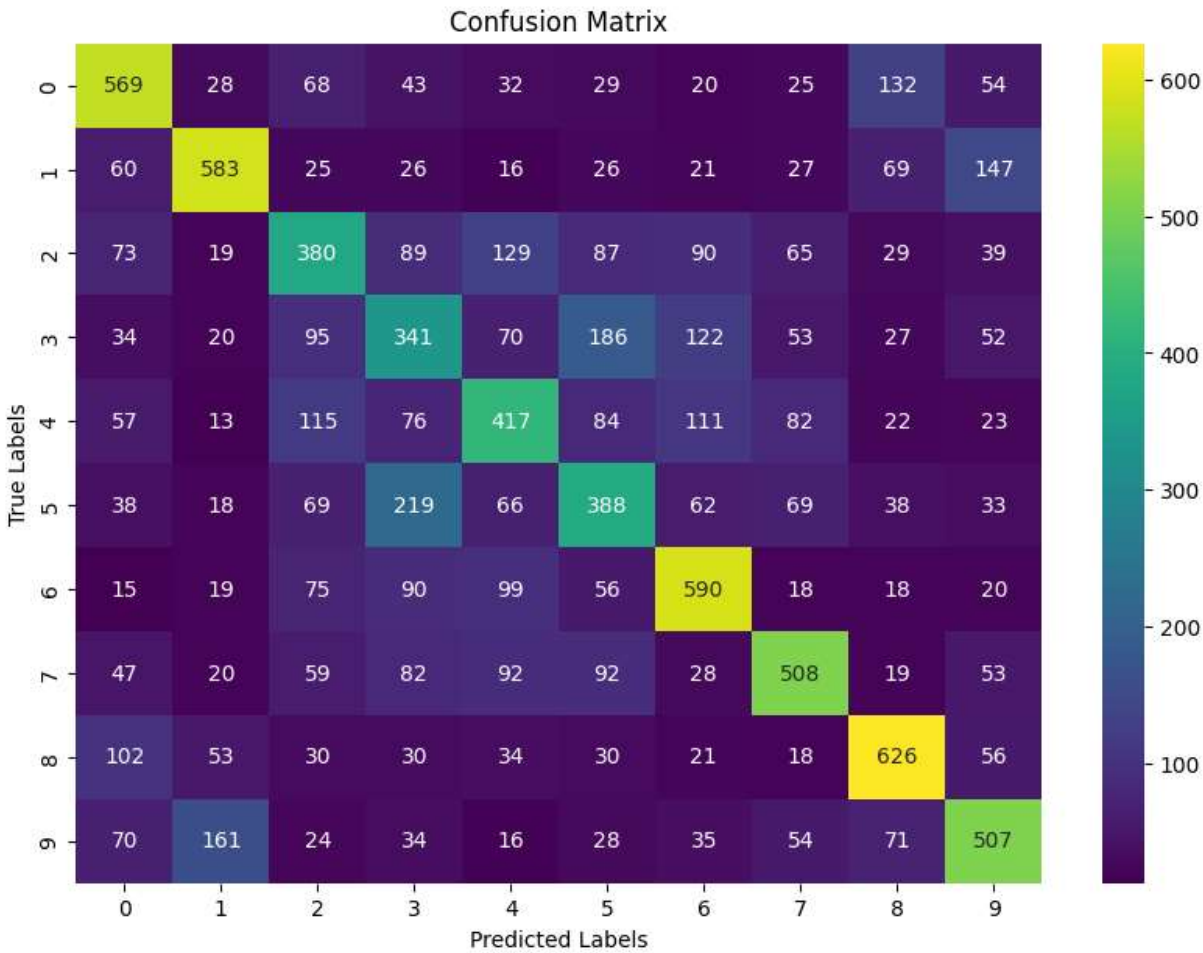
```
# Define the model
model4 = nn.Sequential(
    nn.Linear(3 * 32 * 32, 512),
    nn.Tanh(),
    nn.Linear(512, 256),
    nn.Tanh(),
    nn.Linear(256, 128),
    nn.Tanh(),
    nn.Linear(128, 10),
    nn.LogSoftmax(dim=1)
)
model4.to(device, non_blocking=True)

# Define loss function and optimizer
loss_fn = nn.NLLLoss()
optimizer = optim.SGD(model4.parameters(), lr=0.01)

# Training loop
n_epochs = 100
start_time = time.time()

all_preds = []
all_labels = []
```

```
for epoch in range(n_epochs):
    model3.train()
    #print("Model device:", next(model4.parameters()).device)
    total_loss = 0
    for imgs, labels in trainloader:
        imgs, labels = imgs.to(device, non_blocking=True), labels.to(device, non_bl
        #print("Batch Device:", imgs.device)
        optimizer.zero_grad()
        outputs = model4(imgs.view(imgs.size(0), -1))
        loss = loss_fn(outputs, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / len(trainloader)
    print(f"Epoch: {epoch}, Average Loss: {avg_loss}")

training_time = time.time() - start_time
print(f"Training completed in {training_time:.2f} seconds.")

# Evaluate the network
correct = 0
total = 0
model3.eval()
with torch.no_grad():
    for imgs, labels in testloader:
        imgs, labels = imgs.to(device, non_blocking=True), labels.to(device, non_bl
        outputs = model4(imgs.view(imgs.size(0), -1))
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        # Store predictions and actual labels
        all_preds.extend(predicted.view(-1).tolist())
        all_labels.extend(labels.view(-1).tolist())

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

# Calculate the confusion matrix
conf_matrix = confusion_matrix(all_labels, all_preds)

# Calculate the F1 score
f1Score = f1_score(all_labels, all_preds, average='weighted')

#print("Confusion Matrix:\n", conf_matrix)
print("F1 Score: ", f1Score)
```

```
Epoch: 0, Average Loss: 1.9959205279264913
Epoch: 1, Average Loss: 1.8200943498965114
Epoch: 2, Average Loss: 1.754783681743895
Epoch: 3, Average Loss: 1.7128593874404499
Epoch: 4, Average Loss: 1.6819902591388245
Epoch: 5, Average Loss: 1.655642751385184
Epoch: 6, Average Loss: 1.6304742146635909
Epoch: 7, Average Loss: 1.604979267053287
Epoch: 8, Average Loss: 1.5798129687833664
Epoch: 9, Average Loss: 1.5549722194976514
Epoch: 10, Average Loss: 1.5290899596860648
Epoch: 11, Average Loss: 1.5048662994218909
Epoch: 12, Average Loss: 1.478872036842434
Epoch: 13, Average Loss: 1.4548326797802429
Epoch: 14, Average Loss: 1.4299234994841963
Epoch: 15, Average Loss: 1.403617142411449
Epoch: 16, Average Loss: 1.378280603519791
Epoch: 17, Average Loss: 1.353744968733824
Epoch: 18, Average Loss: 1.326872205337905
Epoch: 19, Average Loss: 1.3007517422709014
Epoch: 20, Average Loss: 1.2737942378386817
Epoch: 21, Average Loss: 1.248310415412459
Epoch: 22, Average Loss: 1.22143248668717
Epoch: 23, Average Loss: 1.1924778229135382
Epoch: 24, Average Loss: 1.166308185252387
Epoch: 25, Average Loss: 1.1397658747327908
Epoch: 26, Average Loss: 1.1111723838559806
Epoch: 27, Average Loss: 1.0862032865624294
Epoch: 28, Average Loss: 1.060364114277808
Epoch: 29, Average Loss: 1.0338041209198934
Epoch: 30, Average Loss: 1.0080324290201181
Epoch: 31, Average Loss: 0.9804229538154114
Epoch: 32, Average Loss: 0.9529009579545091
Epoch: 33, Average Loss: 0.9273771024725931
Epoch: 34, Average Loss: 0.9006961895071942
Epoch: 35, Average Loss: 0.8702801844805402
Epoch: 36, Average Loss: 0.8499708480542273
Epoch: 37, Average Loss: 0.8171367969964166
Epoch: 38, Average Loss: 0.7962218171647747
Epoch: 39, Average Loss: 0.7682093635696889
Epoch: 40, Average Loss: 0.7485380262670005
Epoch: 41, Average Loss: 0.7173417386267801
Epoch: 42, Average Loss: 0.6921574650213237
Epoch: 43, Average Loss: 0.6677987672331388
Epoch: 44, Average Loss: 0.6433317054568044
Epoch: 45, Average Loss: 0.6190364348614003
Epoch: 46, Average Loss: 0.6015778809709622
Epoch: 47, Average Loss: 0.5686111842732295
Epoch: 48, Average Loss: 0.547579688908499
Epoch: 49, Average Loss: 0.5268544212288564
Epoch: 50, Average Loss: 0.5030825748429884
Epoch: 51, Average Loss: 0.4821916318229397
Epoch: 52, Average Loss: 0.45776770115279786
Epoch: 53, Average Loss: 0.43288025446712514
Epoch: 54, Average Loss: 0.41633310503400195
Epoch: 55, Average Loss: 0.3992628745944299
```

```
Epoch: 56, Average Loss: 0.3750001923621768
Epoch: 57, Average Loss: 0.35462665765562934
Epoch: 58, Average Loss: 0.34448375265159265
Epoch: 59, Average Loss: 0.32225798337203465
Epoch: 60, Average Loss: 0.3008011099417954
Epoch: 61, Average Loss: 0.2933751741124084
Epoch: 62, Average Loss: 0.27291520443909306
Epoch: 63, Average Loss: 0.2525483853066974
Epoch: 64, Average Loss: 0.2368041184418799
Epoch: 65, Average Loss: 0.22362575450402392
Epoch: 66, Average Loss: 0.21320605057927652
Epoch: 67, Average Loss: 0.20255053766986444
Epoch: 68, Average Loss: 0.18512453741448767
Epoch: 69, Average Loss: 0.17219860349183955
Epoch: 70, Average Loss: 0.15997676873851158
Epoch: 71, Average Loss: 0.14861887417581227
Epoch: 72, Average Loss: 0.1485922390933308
Epoch: 73, Average Loss: 0.12801851191655603
Epoch: 74, Average Loss: 0.12552688531863415
Epoch: 75, Average Loss: 0.11350230247858921
Epoch: 76, Average Loss: 0.1063140023575948
Epoch: 77, Average Loss: 0.09178855674831993
Epoch: 78, Average Loss: 0.08463760495871839
Epoch: 79, Average Loss: 0.08312489899099254
Epoch: 80, Average Loss: 0.07535811240696694
Epoch: 81, Average Loss: 0.07643193951891282
Epoch: 82, Average Loss: 0.07169054599497895
Epoch: 83, Average Loss: 0.08139787683777912
Epoch: 84, Average Loss: 0.049608503607675776
Epoch: 85, Average Loss: 0.047144838060964554
Epoch: 86, Average Loss: 0.045632249431546465
Epoch: 87, Average Loss: 0.0532580888365655
Epoch: 88, Average Loss: 0.05138926833625073
Epoch: 89, Average Loss: 0.047183402484078006
Epoch: 90, Average Loss: 0.04674201830447463
Epoch: 91, Average Loss: 0.03026482152164249
Epoch: 92, Average Loss: 0.02757039474016127
Epoch: 93, Average Loss: 0.021817701060058134
Epoch: 94, Average Loss: 0.03415354337993428
Epoch: 95, Average Loss: 0.049676357521949446
Epoch: 96, Average Loss: 0.021193363095807563
Epoch: 97, Average Loss: 0.018918649686495665
Epoch: 98, Average Loss: 0.01684103309012511
Epoch: 99, Average Loss: 0.014082461886126977
Training completed in 430.78 seconds.
Accuracy of the network on the test images: 49.09 %
F1 Score:  0.4904543080048704
```

In [ ]:

```
conf_matrix = confusion_matrix(all_labels, all_preds)

# Plotting the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='viridis')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
```

```
plt.ylabel('True Labels')
plt.show()
```