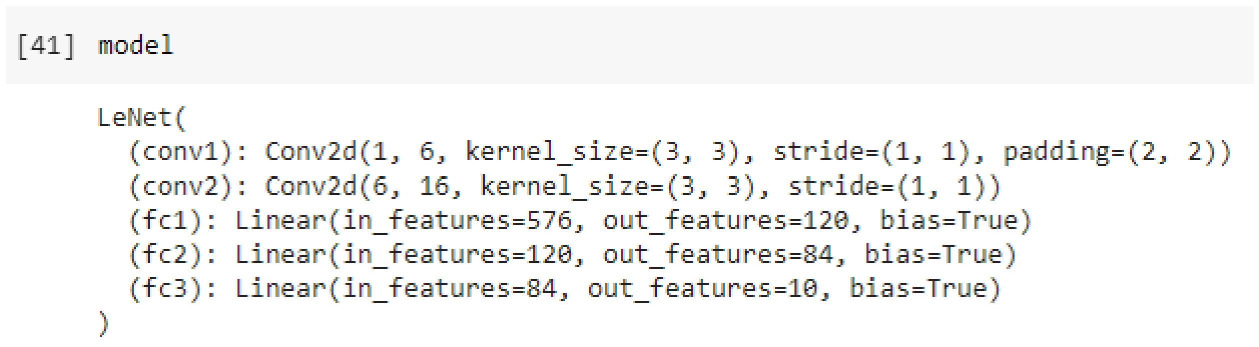
2018033_CV_HW15:

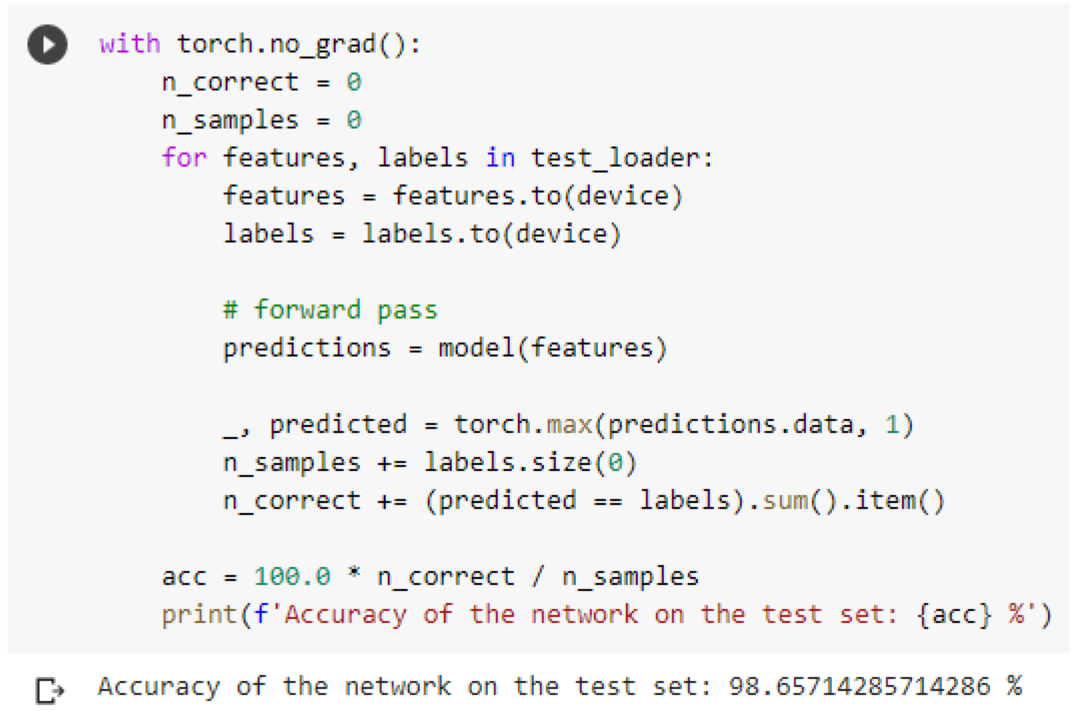
Original LeNet Architecture:

- **Source**: https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html)



```
[41] model

     LeNet(
       (conv1): Conv2d(1, 6, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
       (conv2): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1))
       (fc1): Linear(in_features=576, out_features=120, bias=True)
       (fc2): Linear(in_features=120, out_features=84, bias=True)
       (fc3): Linear(in_features=84, out_features=10, bias=True)
     )
```

- **Test Accuracy**:



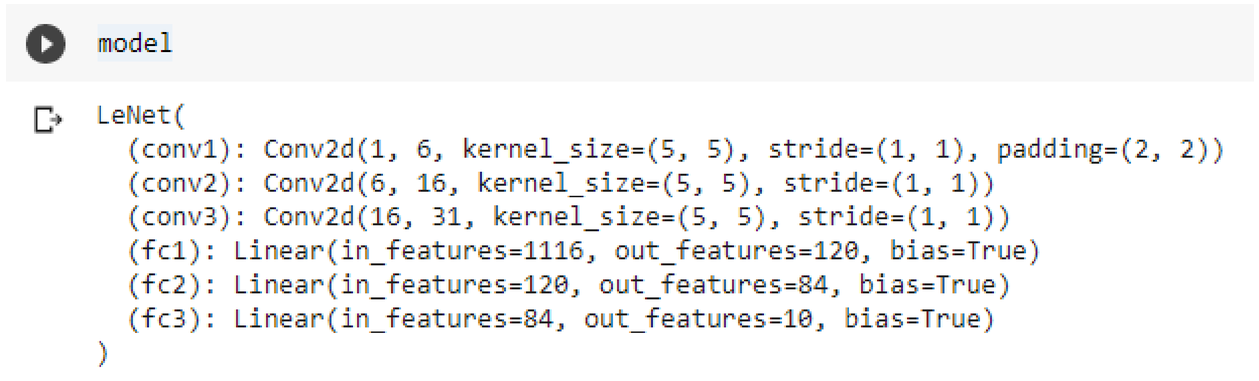
```python
with torch.no_grad():
    n_correct = 0
    n_samples = 0
    for features, labels in test_loader:
        features = features.to(device)
        labels = labels.to(device)

        # forward pass
        predictions = model(features)

        _, predicted = torch.max(predictions.data, 1)
        n_samples += labels.size(0)
        n_correct += (predicted == labels).sum().item()

    acc = 100.0 * n_correct / n_samples
    print(f'Accuracy of the network on the test set: {acc} %')
```

```
Accuracy of the network on the test set: 98.65714285714286 %
```

Modifications:

- Added a conv3 convolution layer having 16 input and 31 output channels.
- Changed the kernel size from (3,3) to (5,5) for all the convolution layers.
- Changed the pooling technique from max-pooling to avg-pooling.
- Pooling has only been applied to the first convolution layer.
- Consequently, the fc1 layer now has 31*6*6 = 1116 neurons.
- (I tried changing the number of filters but did not see much difference in performance.)

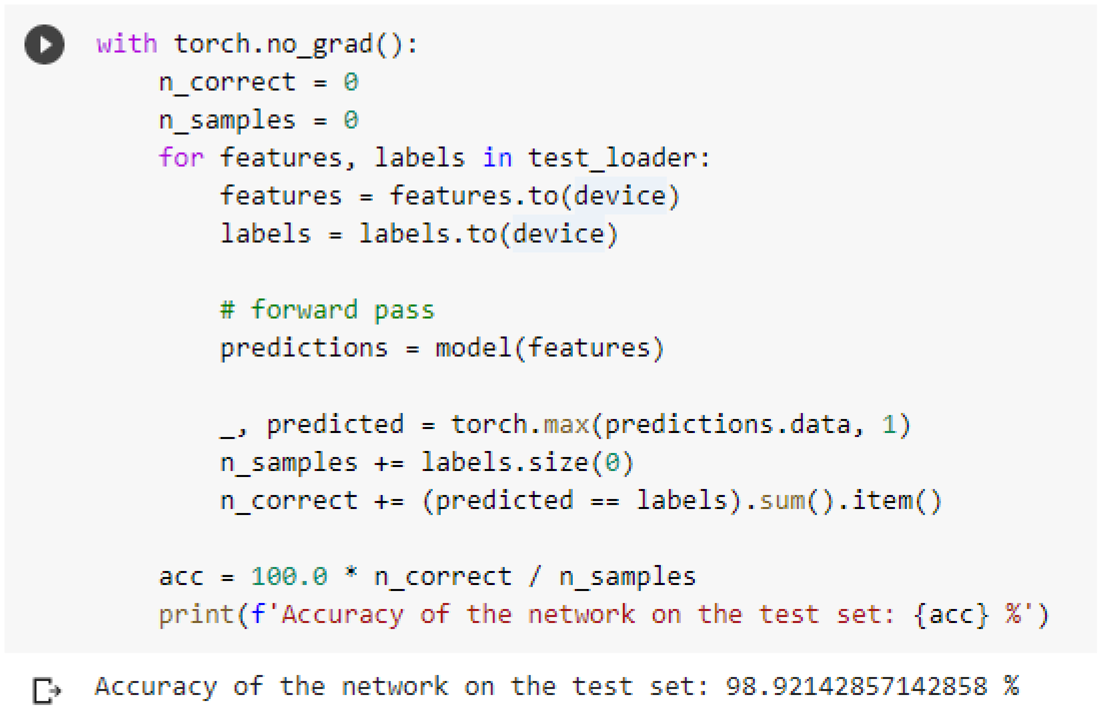
The Modified Architecture:



```
model
```

```
LeNet(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(16, 31, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=1116, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

- **Test Accuracy**:



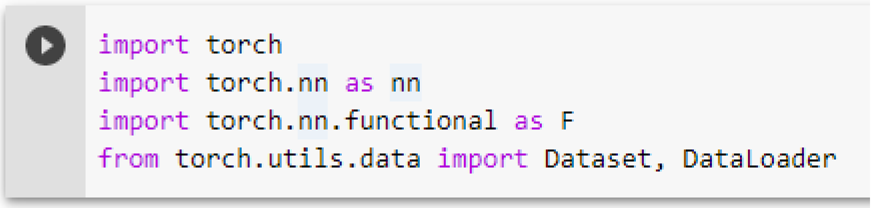
```
with torch.no_grad():
    n_correct = 0
    n_samples = 0
    for features, labels in test_loader:
        features = features.to(device)
        labels = labels.to(device)

        # forward pass
        predictions = model(features)

        _, predicted = torch.max(predictions.data, 1)
        n_samples += labels.size(0)
        n_correct += (predicted == labels).sum().item()

    acc = 100.0 * n_correct / n_samples
    print(f'Accuracy of the network on the test set: {acc} %')
```

```
Accuracy of the network on the test set: 98.92142857142858 %
```

With the above modifications, the test accuracy was improved from 98.66% to 98.92%. The PyTorch trained model has been pickle-dumped to Google Drive.

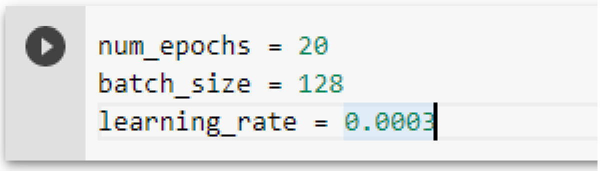
Code: (All work was done in Google Colab)

IMPORTING REQUIRED MODULES



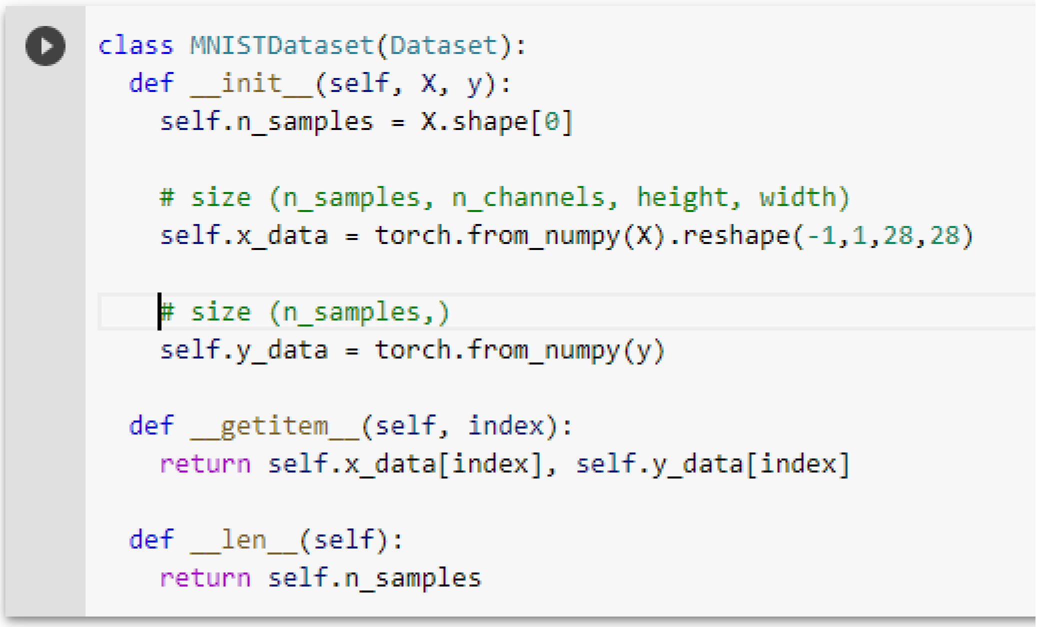
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
```

HYPERPARAMETERS



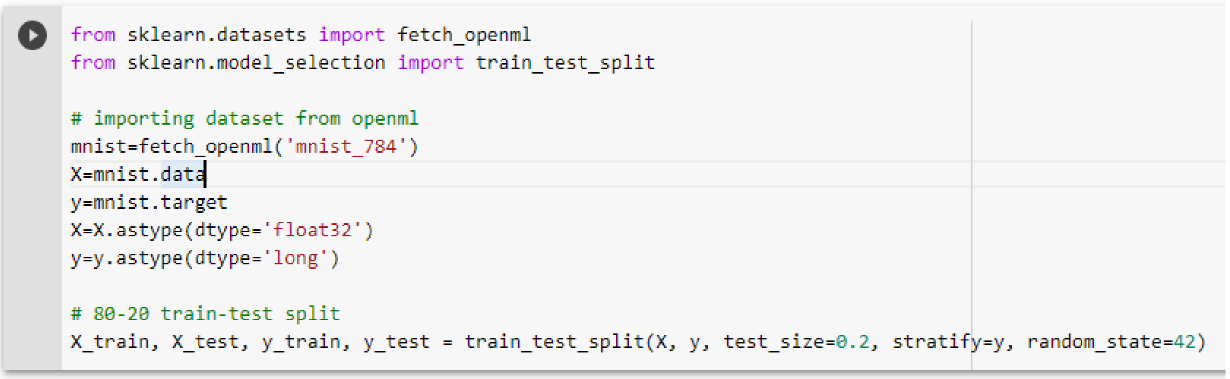
```
num_epochs = 20
batch_size = 128
learning_rate = 0.0003
```

DATASET LOADER CLASS



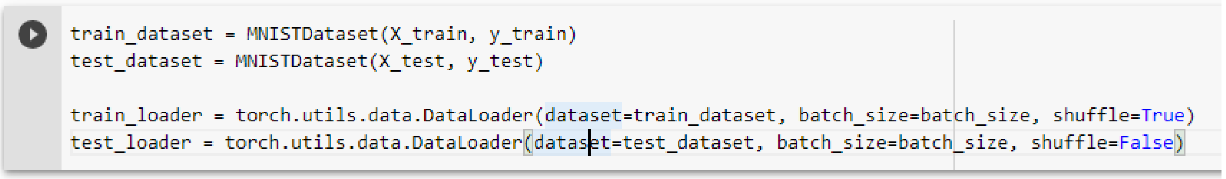
```
class MNISTDataset(Dataset):
  def __init__(self, X, y):
    self.n_samples = X.shape[0]

    # size (n_samples, n_channels, height, width)
    self.x_data = torch.from_numpy(X).reshape(-1,1,28,28)

    # size (n_samples,)
    self.y_data = torch.from_numpy(y)

  def __getitem__(self, index):
    return self.x_data[index], self.y_data[index]

  def __len__(self):
    return self.n_samples
```

DOWNLOADING DATASET



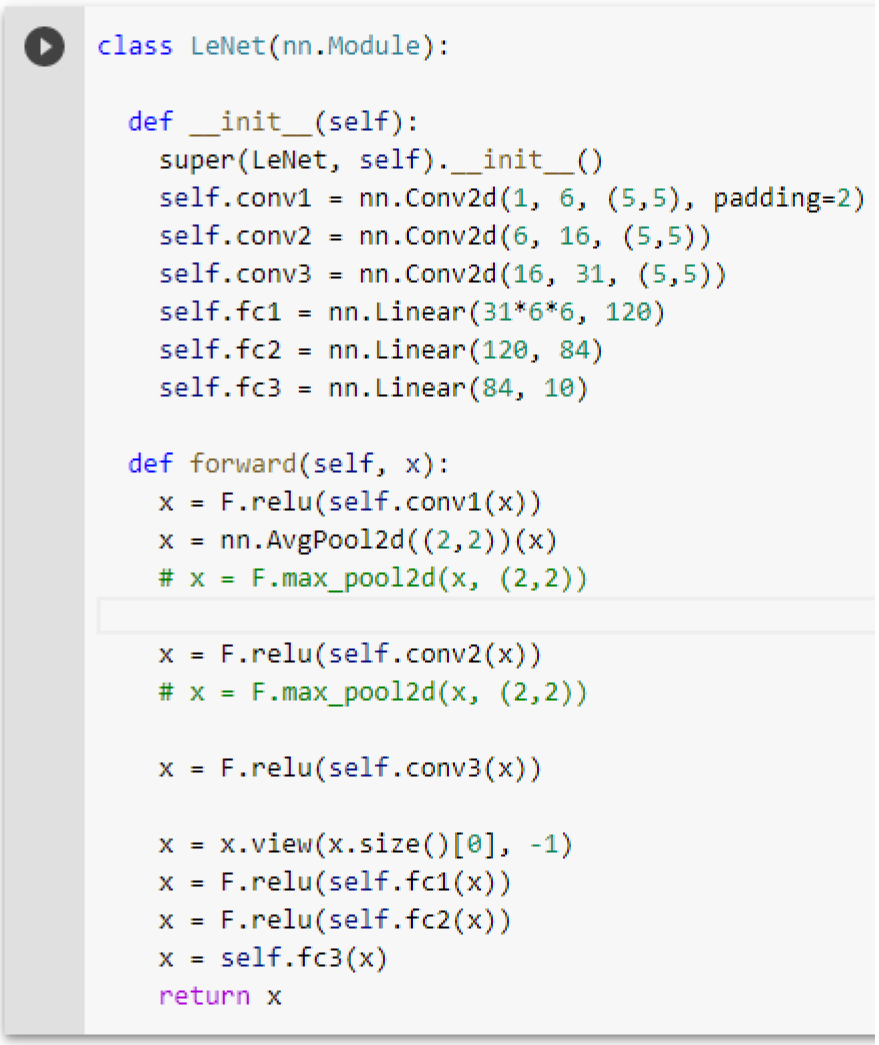
```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split

# importing dataset from openml
mnist=fetch_openml('mnist_784')
X=mnist.data
y=mnist.target
X=X.astype(dtype='float32')
y=y.astype(dtype='long')

# 80-20 train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

TRAIN & TEST DATASETS



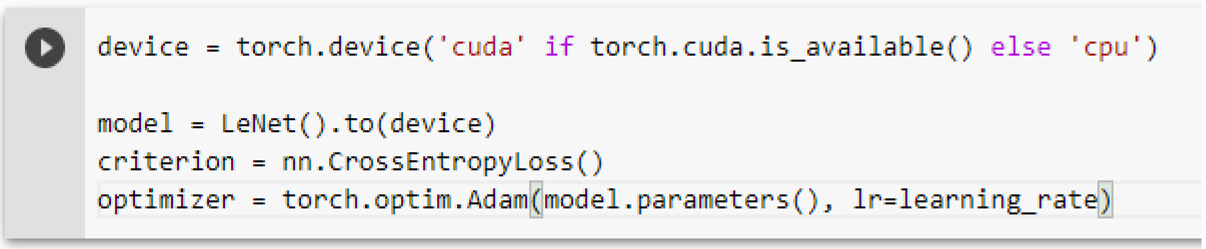
```
train_dataset = MNISTDataset(X_train, y_train)
test_dataset = MNISTDataset(X_test, y_test)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
```

NEURAL NETWORK ARCHITECTURE

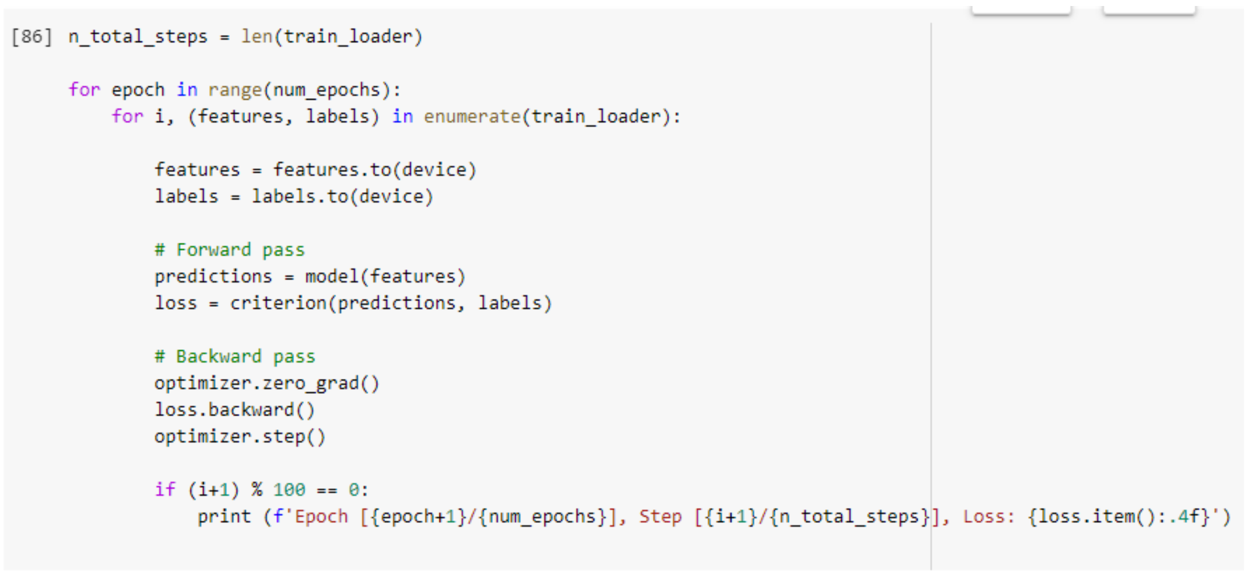


```python
class LeNet(nn.Module):

  def __init__(self):
    super(LeNet, self).__init__()
    self.conv1 = nn.Conv2d(1, 6, (5,5), padding=2)
    self.conv2 = nn.Conv2d(6, 16, (5,5))
    self.conv3 = nn.Conv2d(16, 31, (5,5))
    self.fc1 = nn.Linear(31*6*6, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 10)

  def forward(self, x):
    x = F.relu(self.conv1(x))
    x = nn.AvgPool2d((2,2))(x)
    # x = F.max_pool2d(x, (2,2))

    x = F.relu(self.conv2(x))
    # x = F.max_pool2d(x, (2,2))

    x = F.relu(self.conv3(x))

    x = x.view(x.size()[0], -1)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

TRAINING THE MODEL

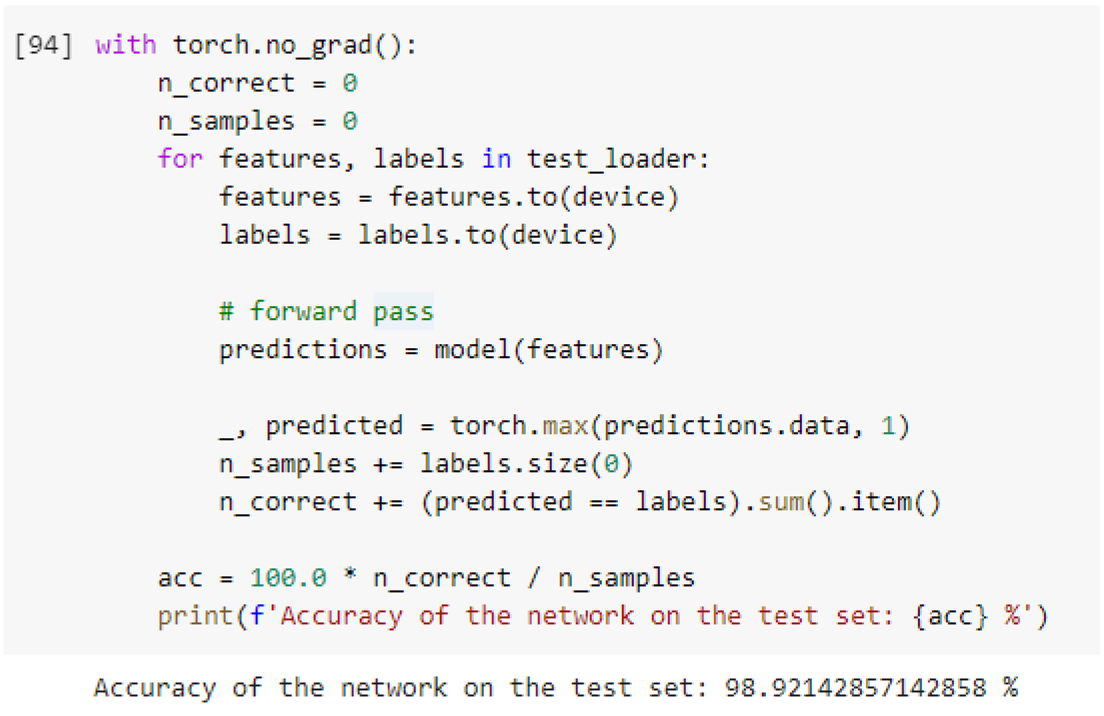


```python
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = LeNet().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```



```
[86] n_total_steps = len(train_loader)

     for epoch in range(num_epochs):
         for i, (features, labels) in enumerate(train_loader):

             features = features.to(device)
             labels = labels.to(device)

             # Forward pass
             predictions = model(features)
             loss = criterion(predictions, labels)

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

             if (i+1) % 100 == 0:
                 print (f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{n_total_steps}], Loss: {loss.item():.4f}')
```

TESTING THE MODEL



```
[94] with torch.no_grad():
         n_correct = 0
         n_samples = 0
         for features, labels in test_loader:
             features = features.to(device)
             labels = labels.to(device)

             # forward pass
             predictions = model(features)

             _, predicted = torch.max(predictions.data, 1)
             n_samples += labels.size(0)
             n_correct += (predicted == labels).sum().item()

         acc = 100.0 * n_correct / n_samples
         print(f'Accuracy of the network on the test set: {acc} %')

     Accuracy of the network on the test set: 98.92142857142858 %
```