```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical



# Load data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Reshape and normalize
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0


Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

# One-hot encode labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)


# Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
1)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes
])


/usr/local/lib/python3.11/dist-packages/keras/src/layers/
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

# Compile
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
```

```
              metrics=['accuracy'])


# Train
model.fit(x_train, y_train, epochs=5, batch_size=64,
validation_split=0.1)
```

```
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 44s 50ms/step - accuracy: 0.8666 - loss:
0.4577 - val_accuracy: 0.9838 - val_loss: 0.0557
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 83s 51ms/step - accuracy: 0.9801 - loss:
0.0665 - val_accuracy: 0.9845 - val_loss: 0.0516
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 80s 49ms/step - accuracy: 0.9870 - loss:
0.0409 - val_accuracy: 0.9852 - val_loss: 0.0528
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 88s 56ms/step - accuracy: 0.9903 - loss:
0.0308 - val_accuracy: 0.9835 - val_loss: 0.0568
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 78s 52ms/step - accuracy: 0.9928 - loss:
0.0213 - val_accuracy: 0.9888 - val_loss: 0.0408

<keras.src.callbacks.history.History at 0x7bf4a9c2bb50>
```

```
# Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9851 - loss:
0.0477
Test accuracy: 0.9884
```

pytorch

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



# Transform: normalize and convert to tensor
transform = transforms.Compose([
    transforms.ToTensor(),
```

```
    transforms.Normalize((0.5,), (0.5,))
])

# Load dataset
train_data = datasets.MNIST(root='./data', train=True, download=True,
transform=transform)
test_data = datasets.MNIST(root='./data', train=False, download=True,
transform=transform)

train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=1000)


100%|██████████| 9.91M/9.91M [00:00<00:00, 122MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 23.5MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 91.6MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 4.51MB/s]

# Define CNN model
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.fc1 = nn.Linear(64 * 5 * 5, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(-1, 64 * 5 * 5)
        x = torch.relu(self.fc1(x))
        return self.fc2(x)

model = CNN()


# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)


# Training loop
for epoch in range(5):
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

```
    print(f"Epoch {epoch+1} complete")


Epoch 1 complete
Epoch 2 complete
Epoch 3 complete
Epoch 4 complete
Epoch 5 complete

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

print(f'Test Accuracy: {100 * correct / total:.2f}%')

Test Accuracy: 98.95%
```