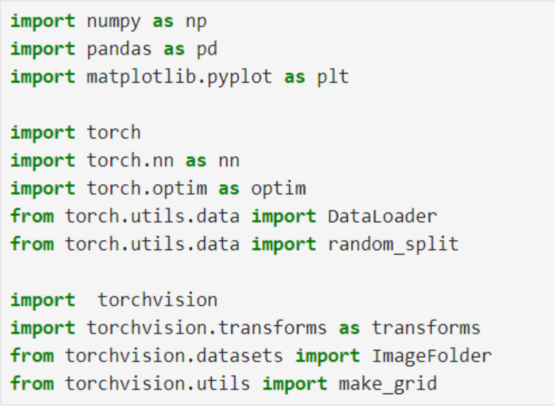
Model Development Phase Template

Date	15 March 2024
Team ID	SWTlD1720439521
Project Title	Covidvision: Advanced Covid-19 Detection From Lung X-Rays With Deep Learning
Maximum Marks	10 Marks

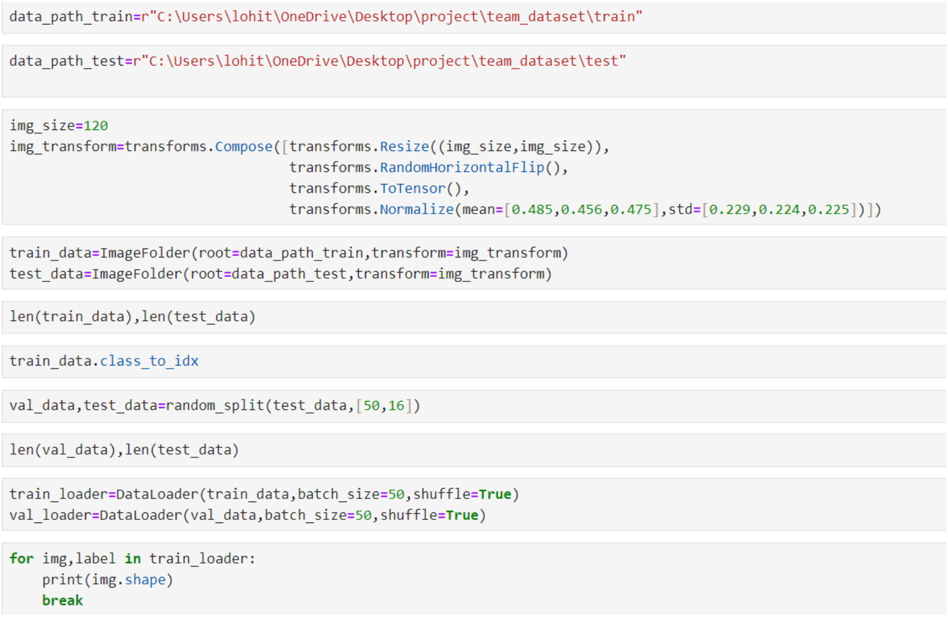
Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include a summary and training and validation performance metrics for multiple models, presented through respective screenshots.

Initial Model Training Code (5 marks):



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torch.utils.data import random_split

import  torchvision
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torchvision.utils import make_grid
```



```
data_path_train=r"C:\Users\lohit\OneDrive\Desktop\project\team_dataset\train"
```

```
data_path_test=r"C:\Users\lohit\OneDrive\Desktop\project\team_dataset\test"
```

```
img_size=120
img_transform=transforms.Compose([transforms.Resize((img_size,img_size)),
                                  transforms.RandomHorizontalFlip(),
                                  transforms.ToTensor(),
                                  transforms.Normalize(mean=[0.485,0.456,0.475],std=[0.229,0.224,0.225])])
```

```
train_data=ImageFolder(root=data_path_train,transform=img_transform)
test_data=ImageFolder(root=data_path_test,transform=img_transform)
```

```
len(train_data),len(test_data)
```

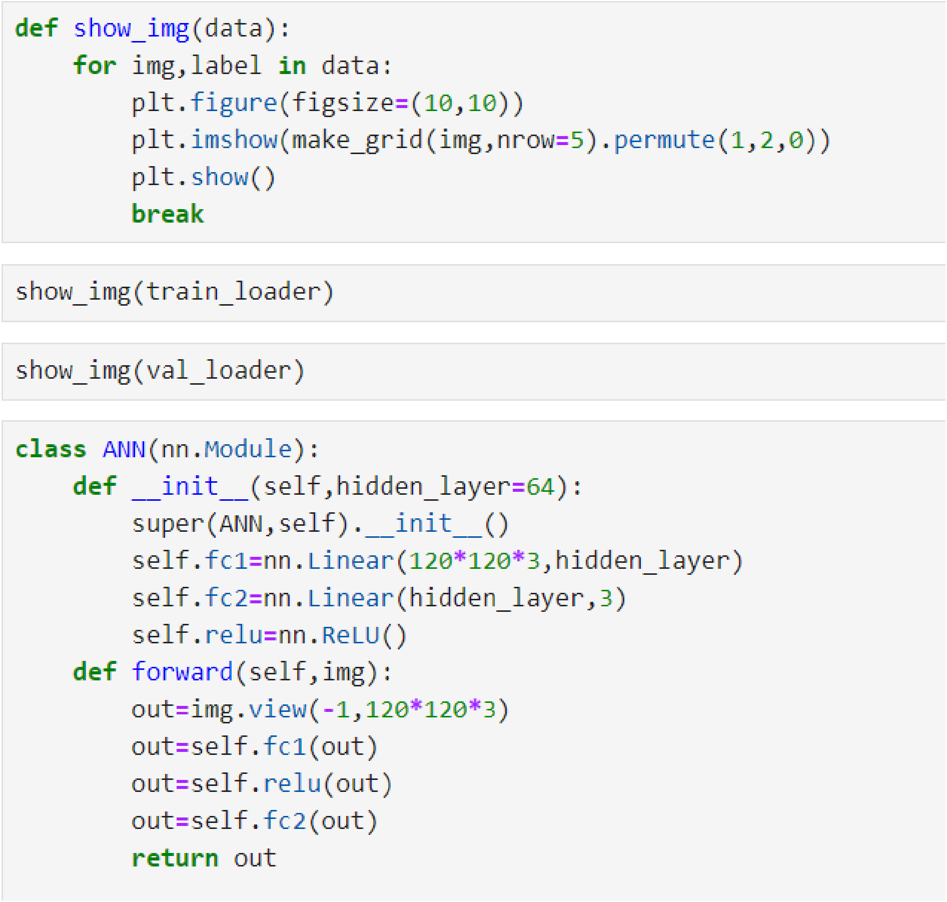
```
train_data.class_to_idx
```

```
val_data,test_data=random_split(test_data,[50,16])
```

```
len(val_data),len(test_data)
```

```
train_loader=DataLoader(train_data,batch_size=50,shuffle=True)
val_loader=DataLoader(val_data,batch_size=50,shuffle=True)
```

```
for img,label in train_loader:
    print(img.shape)
    break
```

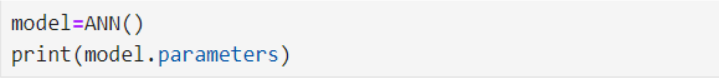


```
def show_img(data):
    for img,label in data:
        plt.figure(figsize=(10,10))
        plt.imshow(make_grid(img,nrow=5).permute(1,2,0))
        plt.show()
        break
```

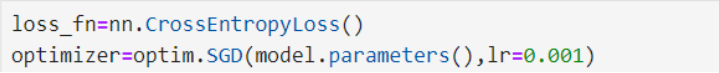
```
show_img(train_loader)
```

```
show_img(val_loader)
```

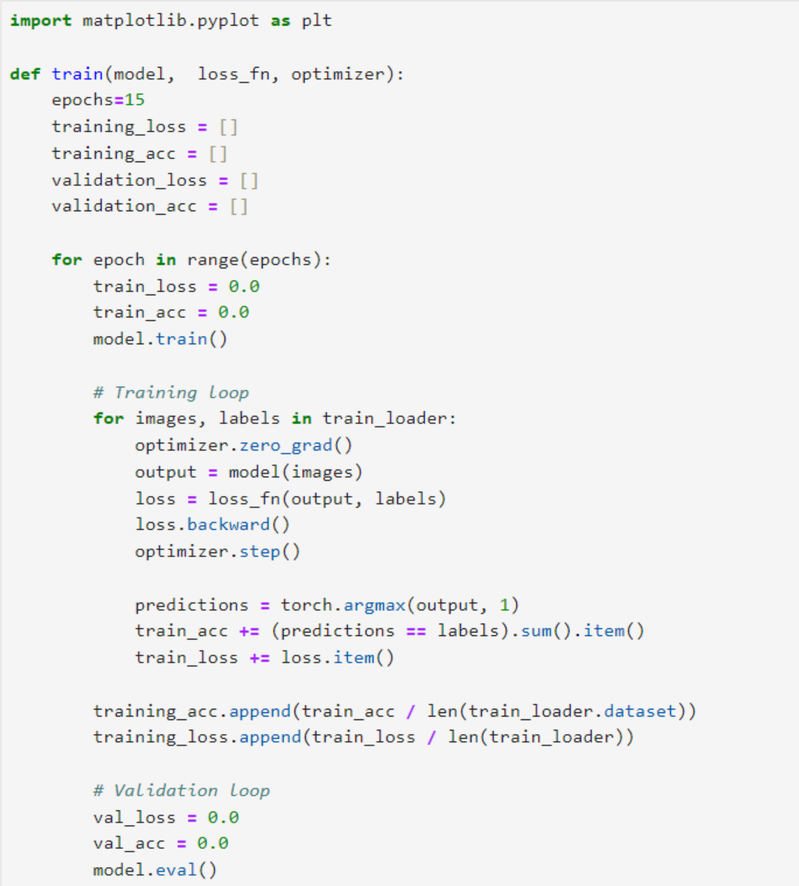
```
class ANN(nn.Module):
    def __init__(self,hidden_layer=64):
        super(ANN,self).__init__()
        self.fc1=nn.Linear(120*120*3,hidden_layer)
        self.fc2=nn.Linear(hidden_layer,3)
        self.relu=nn.ReLU()
    def forward(self,img):
        out=img.view(-1,120*120*3)
        out=self.fc1(out)
        out=self.relu(out)
        out=self.fc2(out)
        return out
```



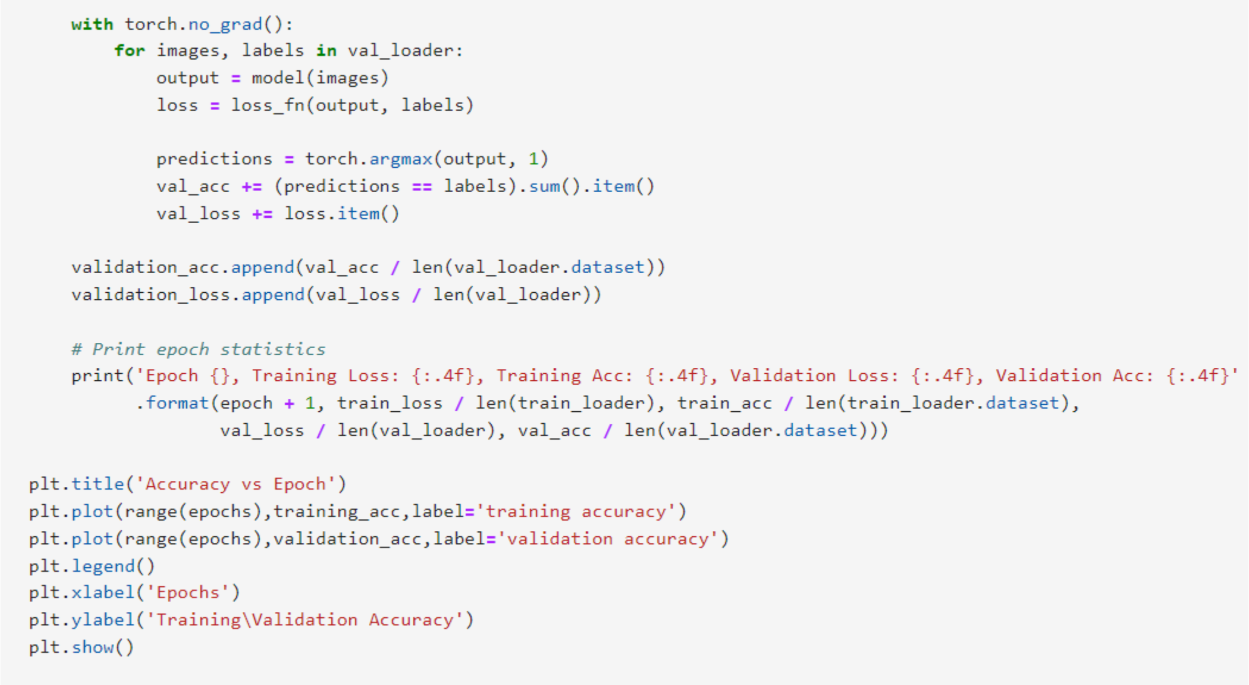
```
model=ANN()
print(model.parameters)
```



```
loss_fn=nn.CrossEntropyLoss()
optimizer=optim.SGD(model.parameters(),lr=0.001)
```

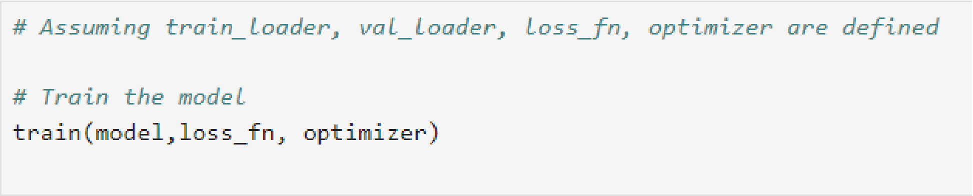


```
import matplotlib.pyplot as plt

def train(model,  loss_fn, optimizer):
    epochs=15
    training_loss = []
    training_acc = []
    validation_loss = []
    validation_acc = []

    for epoch in range(epochs):
        train_loss = 0.0
        train_acc = 0.0
        model.train()

        # Training Loop
        for images, labels in train_loader:
            optimizer.zero_grad()
            output = model(images)
            loss = loss_fn(output, labels)
            loss.backward()
            optimizer.step()

            predictions = torch.argmax(output, 1)
            train_acc += (predictions == labels).sum().item()
            train_loss += loss.item()

        training_acc.append(train_acc / len(train_loader.dataset))
        training_loss.append(train_loss / len(train_loader))

        # Validation Loop
        val_loss = 0.0
        val_acc = 0.0
        model.eval()
```

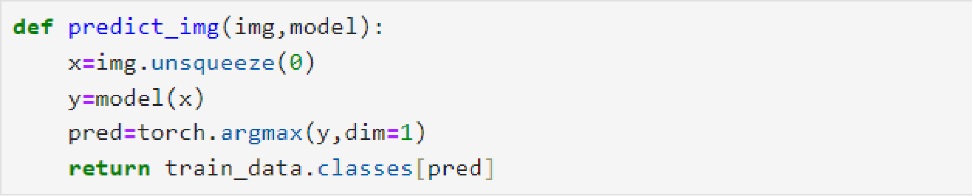


```
    with torch.no_grad():
        for images, labels in val_loader:
            output = model(images)
            loss = loss_fn(output, labels)

            predictions = torch.argmax(output, 1)
            val_acc += (predictions == labels).sum().item()
            val_loss += loss.item()

    validation_acc.append(val_acc / len(val_loader.dataset))
    validation_loss.append(val_loss / len(val_loader))

    # Print epoch statistics
    print('Epoch {}, Training Loss: {:.4f}, Training Acc: {:.4f}, Validation Loss: {:.4f}, Validation Acc: {:.4f}'
          .format(epoch + 1, train_loss / len(train_loader), train_acc / len(train_loader.dataset),
                  val_loss / len(val_loader), val_acc / len(val_loader.dataset)))

plt.title('Accuracy vs Epoch')
plt.plot(range(epochs),training_acc,label='training accuracy')
plt.plot(range(epochs),validation_acc,label='validation accuracy')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Training\Validation Accuracy')
plt.show()
```

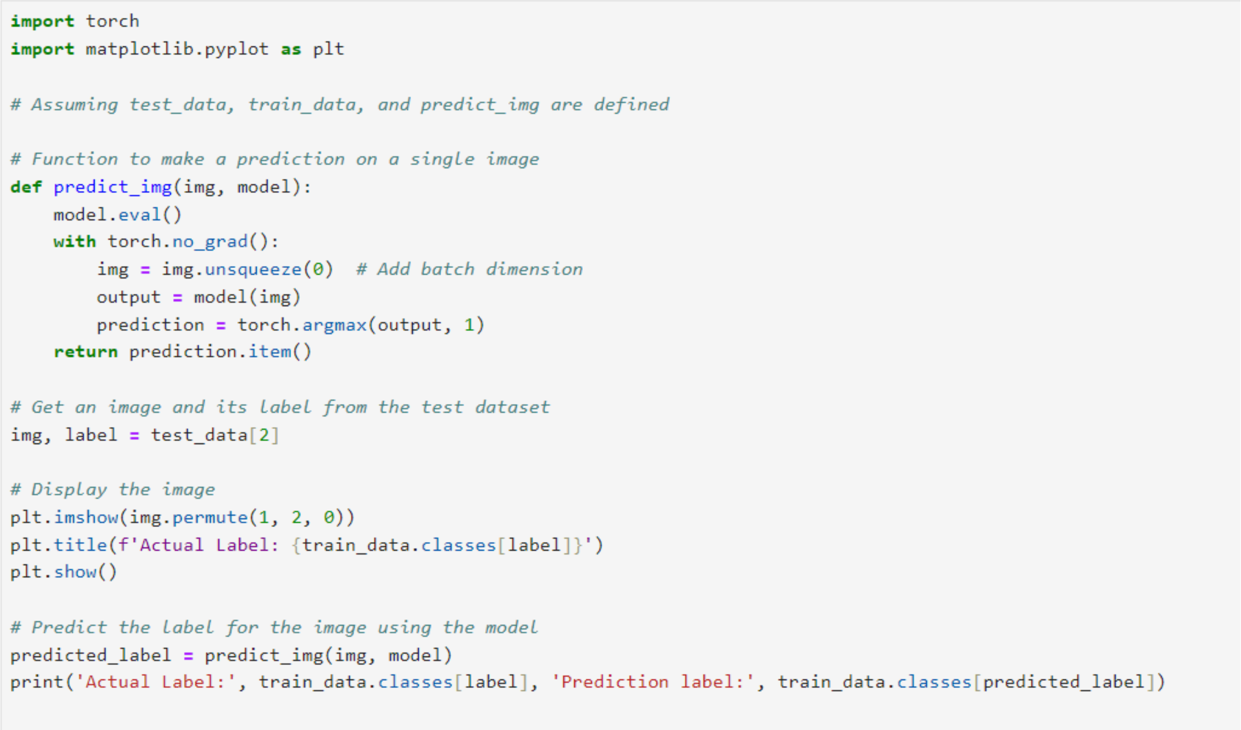
```
    with torch.no_grad():
        for images, labels in val_loader:
            output = model(images)
            loss = loss_fn(output, labels)

            predictions = torch.argmax(output, 1)
            val_acc += (predictions == labels).sum().item()
            val_loss += loss.item()

    validation_acc.append(val_acc / len(val_loader.dataset))
    validation_loss.append(val_loss / len(val_loader))

    # Print epoch statistics
    print('Epoch {}, Training Loss: {:.4f}, Training Acc: {:.4f}, Validation Loss: {:.4f}, Validation Acc: {:.4f}'
          .format(epoch + 1, train_loss / len(train_loader), train_acc / len(train_loader.dataset),
                  val_loss / len(val_loader), val_acc / len(val_loader.dataset)))

plt.title('Accuracy vs Epoch')
plt.plot(range(epochs),training_acc,label='training accuracy')
plt.plot(range(epochs),validation_acc,label='validation accuracy')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Training\Validation Accuracy')
plt.show()
```



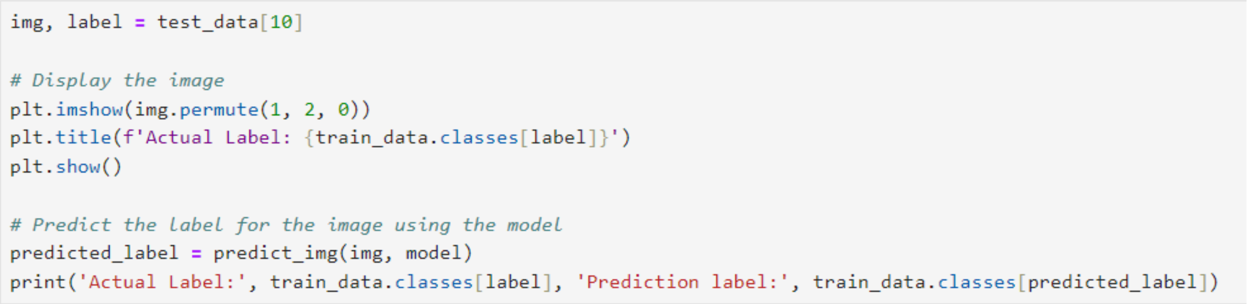
```
# Assuming train_loader, val_loader, loss_fn, optimizer are defined

# Train the model
train(model,loss_fn, optimizer)
```



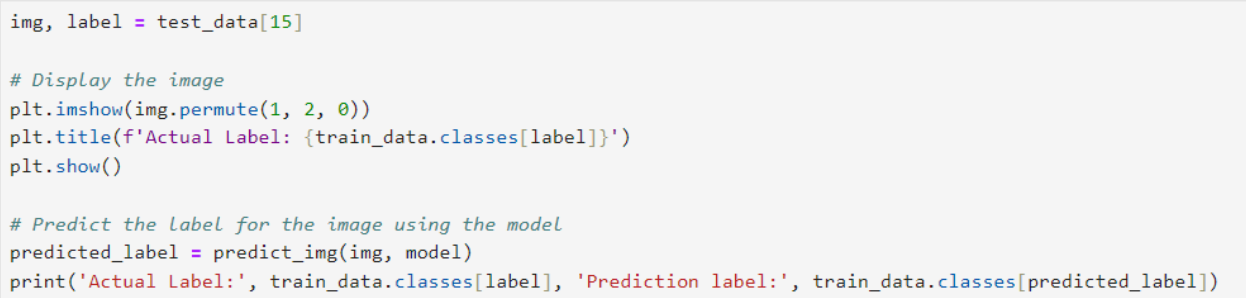
```
def predict_img(img,model):
    x=img.unsqueeze(0)
    y=model(x)
    pred=torch.argmax(y,dim=1)
    return train_data.classes[pred]
```



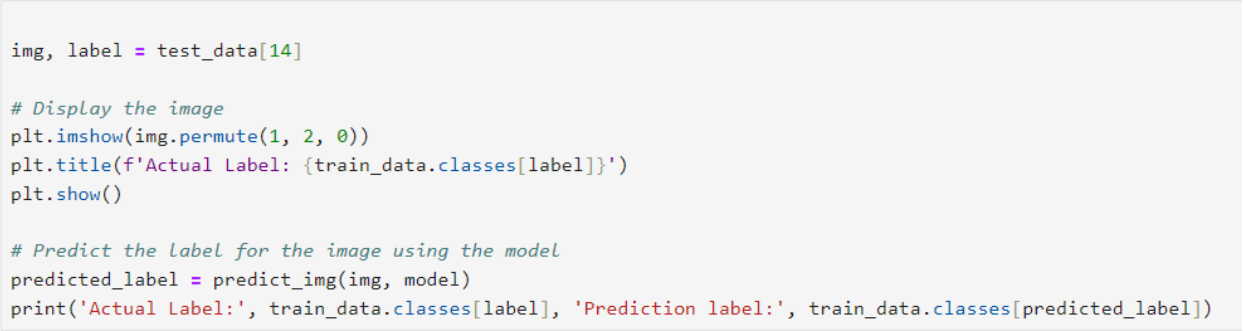
```
import torch
import matplotlib.pyplot as plt

# Assuming test_data, train_data, and predict_img are defined

# Function to make a prediction on a single image
def predict_img(img, model):
    model.eval()
    with torch.no_grad():
        img = img.unsqueeze(0)  # Add batch dimension
        output = model(img)
        prediction = torch.argmax(output, 1)
    return prediction.item()

# Get an image and its label from the test dataset
img, label = test_data[2]

# Display the image
plt.imshow(img.permute(1, 2, 0))
plt.title(f'Actual Label: {train_data.classes[label]}')
plt.show()

# Predict the label for the image using the model
predicted_label = predict_img(img, model)
print('Actual Label:', train_data.classes[label], 'Prediction label:', train_data.classes[predicted_label])
```



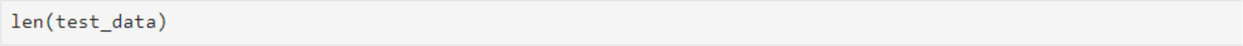
```
img, label = test_data[10]

# Display the image
plt.imshow(img.permute(1, 2, 0))
plt.title(f'Actual Label: {train_data.classes[label]}')
plt.show()

# Predict the label for the image using the model
predicted_label = predict_img(img, model)
print('Actual Label:', train_data.classes[label], 'Prediction label:', train_data.classes[predicted_label])
```



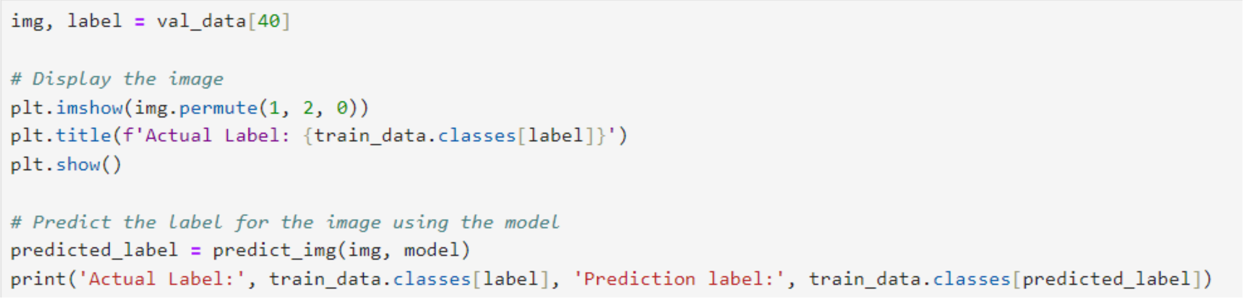
```
img, label = test_data[15]

# Display the image
plt.imshow(img.permute(1, 2, 0))
plt.title(f'Actual Label: {train_data.classes[label]}')
plt.show()

# Predict the label for the image using the model
predicted_label = predict_img(img, model)
print('Actual Label:', train_data.classes[label], 'Prediction label:', train_data.classes[predicted_label])
```



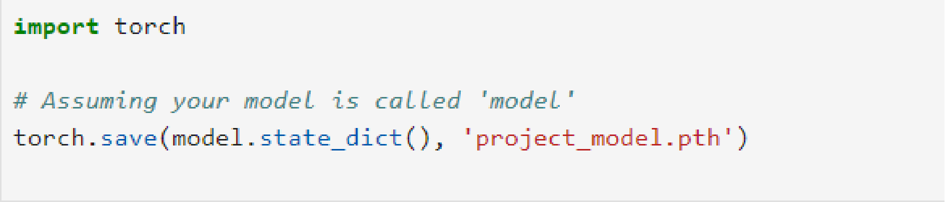
```
img, label = test_data[14]

# Display the image
plt.imshow(img.permute(1, 2, 0))
plt.title(f'Actual Label: {train_data.classes[label]}')
plt.show()

# Predict the label for the image using the model
predicted_label = predict_img(img, model)
print('Actual Label:', train_data.classes[label], 'Prediction label:', train_data.classes[predicted_label])
```



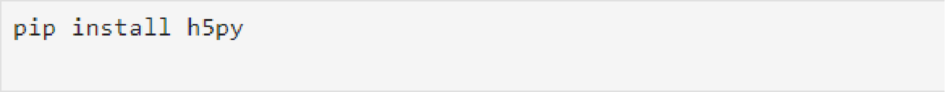
```
len(test_data)
```



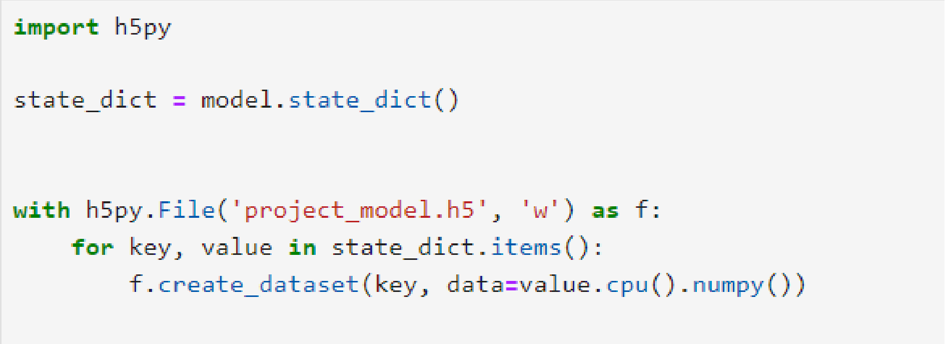
```
img, label = val_data[40]

# Display the image
plt.imshow(img.permute(1, 2, 0))
plt.title(f'Actual Label: {train_data.classes[label]}')
plt.show()

# Predict the label for the image using the model
predicted_label = predict_img(img, model)
print('Actual Label:', train_data.classes[label], 'Prediction label:', train_data.classes[predicted_label])
```



```
import torch

# Assuming your model is called 'model'
torch.save(model.state_dict(), 'project_model.pth')
```



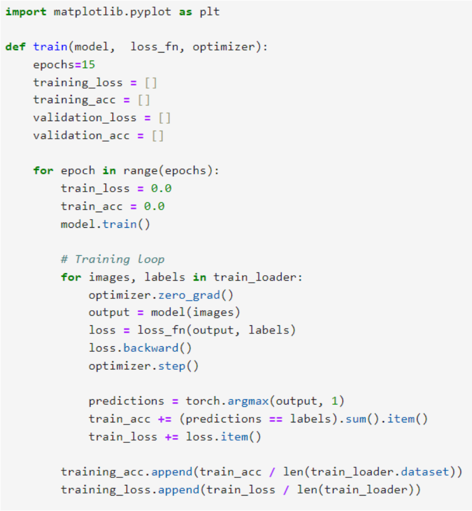
```
pip install h5py
```



```
import h5py

state_dict = model.state_dict()


with h5py.File('project_model.h5', 'w') as f:
    for key, value in state_dict.items():
        f.create_dataset(key, data=value.cpu().numpy())
```

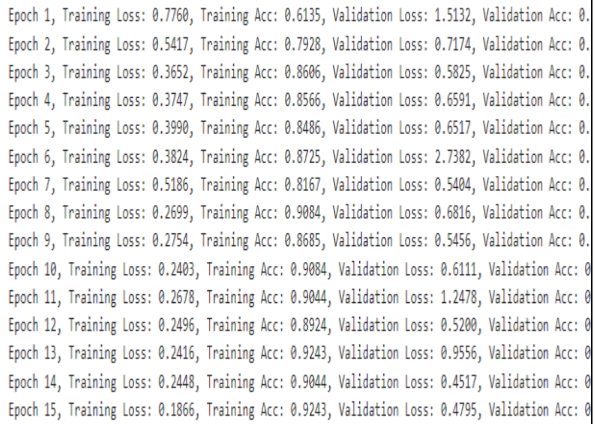
Model Validation and Evaluation Report (5 marks):

<table>
<tr><th>Model</th><th>Summary</th><th>Training and Validation Performance Metrics</th></tr>
<tr><td>Model 1</td><td>



```
import matplotlib.pyplot as plt

def train(model,  loss_fn, optimizer):
    epochs=15
    training_loss = []
    training_acc = []
    validation_loss = []
    validation_acc = []

    for epoch in range(epochs):
        train_loss = 0.0
        train_acc = 0.0
        model.train()

        # Training loop
        for images, labels in train_loader:
            optimizer.zero_grad()
            output = model(images)
            loss = loss_fn(output, labels)
            loss.backward()
            optimizer.step()

            predictions = torch.argmax(output, 1)
            train_acc += (predictions == labels).sum().item()
            train_loss += loss.item()

        training_acc.append(train_acc / len(train_loader.dataset))
        training_loss.append(train_loss / len(train_loader))
```

</td><td>



```
Epoch 1, Training Loss: 0.7760, Training Acc: 0.6135, Validation Loss: 1.5132, Validation Acc: 0.
Epoch 2, Training Loss: 0.5417, Training Acc: 0.7928, Validation Loss: 0.7174, Validation Acc: 0.
Epoch 3, Training Loss: 0.3652, Training Acc: 0.8606, Validation Loss: 0.5825, Validation Acc: 0.
Epoch 4, Training Loss: 0.3747, Training Acc: 0.8566, Validation Loss: 0.6591, Validation Acc: 0.
Epoch 5, Training Loss: 0.3990, Training Acc: 0.8486, Validation Loss: 0.6517, Validation Acc: 0.
Epoch 6, Training Loss: 0.3824, Training Acc: 0.8725, Validation Loss: 2.7382, Validation Acc: 0.
Epoch 7, Training Loss: 0.5186, Training Acc: 0.8167, Validation Loss: 0.5404, Validation Acc: 0.
Epoch 8, Training Loss: 0.2699, Training Acc: 0.9084, Validation Loss: 0.6816, Validation Acc: 0.
Epoch 9, Training Loss: 0.2754, Training Acc: 0.8685, Validation Loss: 0.5456, Validation Acc: 0.
Epoch 10, Training Loss: 0.2403, Training Acc: 0.9084, Validation Loss: 0.6111, Validation Acc: 0
Epoch 11, Training Loss: 0.2678, Training Acc: 0.9044, Validation Loss: 1.2478, Validation Acc: 0
Epoch 12, Training Loss: 0.2496, Training Acc: 0.8924, Validation Loss: 0.5200, Validation Acc: 0
Epoch 13, Training Loss: 0.2416, Training Acc: 0.9243, Validation Loss: 0.9556, Validation Acc: 0
Epoch 14, Training Loss: 0.2448, Training Acc: 0.9044, Validation Loss: 0.4517, Validation Acc: 0
Epoch 15, Training Loss: 0.1866, Training Acc: 0.9243, Validation Loss: 0.4795, Validation Acc: 0
```

</td></tr>
</table>