A Report
On
Assignment
CSE 465

By

Name: Fahmida Akter Nourin
Id: 2031741642

Question 1 (A):

```python
import torch
import torch.nn as nn
from torchvision.transforms import ToTensor

class task_A(nn.Module):
  def __init__(self):
    super(task_A, self).__init__()
    self.conv1x1 = nn.Conv2d(256, 128, kernel_size=1)
    self.conv3x3 = nn.Conv2d(256, 192, kernel_size=3, padding=1)
    self.conv5x5 = nn.Conv2d(256, 96, kernel_size=5, padding=2)
    self.pool3x3 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)

  def forward(self, x):
    x1 = nn.ReLU()(self.conv1x1(x))
    x2 = nn.ReLU()(self.conv3x3(x))
    x3 = nn.ReLU()(self.conv5x5(x))
    pooled_x = self.pool3x3(x)
    out = torch.cat((x1, x2, x3, pooled_x), dim=1)
    return out

input_tensor = torch.randn(1, 256, 28, 28)
model = task_A()
output_tensor = model(input_tensor)

print(model)
print(output_tensor.shape)
```

Output:

```python
Output:
task_A(
  (conv1x1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
  (conv3x3): Conv2d(256, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
  (conv5x5): Conv2d(256, 96, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (pool3x3): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=False)
)
torch.Size([1, 672, 28, 28])
```

Question 1 (B):

```python
class task_B(nn.Module):
  def __init__(self):
    super(task_B,self).__init__()
    self.branch1 = nn.Conv2d(256,128, kernel_size = 1)
    self.branch2 = nn.Sequential(
      nn.Conv2d(256,64,kernel_size = 1),
      nn.ReLU(),
      nn.Conv2d(64, 192, kernel_size = 3, padding =1),
    )
    self.branch3 = nn.Sequential(
      nn.Conv2d(256,64,kernel_size = 1),
      nn.ReLU(),
      nn.Conv2d(64, 96, kernel_size = 5, padding = 2),
    )
    self.branch4 = nn.Sequential(
      nn.MaxPool2d(kernel_size = 3, stride = 1 ,padding = 1),
      nn.Conv2d(256, 64, kernel_size = 1),
    )

  def forward(self,x):
    b1_out = self.branch1(x)
    b2_out = self.branch2(x)
    b3_out = self.branch3(x)
    b4_out = self.branch4(x)
    out = torch.cat((b1_out,b2_out,b3_out,b4_out), dim=1)
    return out

input_x = torch.randn(1, 256, 28, 28)
model = task_B()
output = model(input_x)
print(model)
print(output.shape)
```

Output:

```python
task_B(
 (branch1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
 (branch2): Sequential(
  (0): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1))
```

```
  (1): ReLU()
  (2): Conv2d(64, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
 )
 (branch3): Sequential(
  (0): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1))
  (1): ReLU()
  (2): Conv2d(64, 96, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
 )
 (branch4): Sequential(
  (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
  (1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1))
 )
)
torch.Size([1, 480, 28, 28])
```

Question 1 (C):

Python

```python
import torch
import torch.nn as nn

class RNN(nn.Module):
  def __init__(self, input_size, hidden_size, num_layers=1, dropout=0.1):
    super().__init__()
    self.hidden_size = hidden_size
    self.num_layers = num_layers
    self.rnn = nn.RNN(input_size, hidden_size, num_layers, dropout=dropout)

  def forward(self, input, hidden):
    output, hidden = self.rnn(input, hidden)
    return output


class Softmax(nn.Module):
  def __init__(self, input_size):
    super().__init__()
    self.linear = nn.Linear(input_size, input_size)

  def forward(self, input):
    output = self.linear(input)
```

```
    probs = torch.nn.functional.softmax(output, dim=1)
    return probs


# Convert the input text to word embeddings
input_text = "The quick brown fox"
word_to_index = {word: idx for idx, word in enumerate(input_text.lower().split())}
input_sequence = torch.tensor([[word_to_index[word] for word in
input_text.lower().split()]] , dtype=torch.float)



input_size = 4  #len(input_text.split(" "))
hidden_layers_size = 4   #len(input_text.split(" "))
num_layers = 2
dropout = 0.2



rnn_module = RNN(input_size, hidden_layers_size, num_layers, dropout)
softmax_module = Softmax(hidden_layers_size)


# Initialize hidden state with correct shape
hidden_layers = torch.zeros(num_layers, hidden_layers_size, dtype=torch.float)


# Pass input through modules:
output = rnn_module(input_sequence, hidden_layers)
prob = softmax_module(output)



print(prob)
```

OutPut

```
Python
tensor([[0.3461, 0.2818, 0.1557, 0.2164]], grad_fn=<SoftmaxBackward0>)
```

Question 2

Python

```
import torch
import torch.nn as nn
import torchvision.models as models
from torchvision import transforms, datasets
from PIL import Image
import matplotlib.pyplot as plt
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


img_path = '/kaggle/input/vegetable-image-dataset/Vegetable
Images/train/Bean/0026.jpg'
img = Image.open(img_path).convert('RGB')

#transforming image
preprocess = transforms.Compose([
  transforms.Resize((224, 224)),
  transforms.ToTensor(),
  transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])


img_tensor = preprocess(img)
img_tensor = torch.unsqueeze(img_tensor, 0).to(device)


transform = transforms.Compose([
  transforms.Resize((224, 224)),
  transforms.ToTensor(),
  transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])


train_dataset =
datasets.ImageFolder(root='/kaggle/input/vegetable-image-dataset/Vegetable
Images/train', transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32,
shuffle=True)



num_classes = 15
```

```
# function of the feature map of the specified layer
def plot_feature_map(model,img_tensor,idx):
  feature_map_model = torch.nn.Sequential(*list(model.children())[:idx])
  feature_map_model.to(device)
  feature_map = feature_map_model(img_tensor)

  with torch.no_grad():
    feature_map = feature_map_model(img_tensor)

  # Visualize the feature map
  plt.imshow(feature_map[0, 0].cpu().detach().numpy(), cmap='viridis')
  plt.show()



def Simple_train(train_loader,model,criterion,optimizer):
  num_epochs = 3
  for epoch in range(num_epochs):
    for inputs, labels in train_loader:
      inputs, labels = inputs.to(device), labels.to(device)

      optimizer.zero_grad()
      outputs = model(inputs)
      loss = criterion(outputs, labels)
      loss.backward()
      optimizer.step()

    print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {loss.item()}')
  return model
```

Python

```
vgg16_model = models.vgg16(pretrained=True)
vgg16_model.classifier[-1] = nn.Linear(4096, num_classes)


# Train the model VGG16
vgg16_model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(vgg16_model.parameters(), lr=0.001, momentum=0.9)
```

```
vgg16_model.train()
vgg16_model = Simple_train(train_loader,vgg16_model,criterion,optimizer)
```

Python

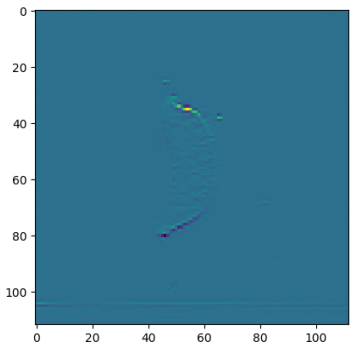
```
resnet101_model = models.resnet101(pretrained=True)
resnet101_model.fc = nn.Linear(resnet101_model.fc.in_features, num_classes)


# Train the model resnet10
resnet101_model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(resnet101_model.parameters(), lr=0.001, momentum=0.9)
resnet101_model.train()
resnet101_model = Simple_train(train_loader,resnet101_model,criterion,optimizer)
```

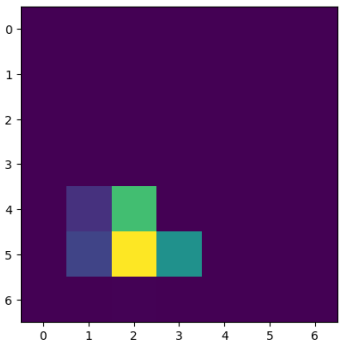
Python

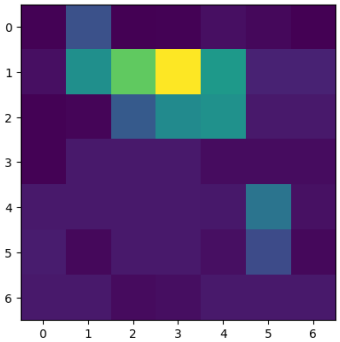
```
mobilenetv3_model = models.mobilenet_v3_small(pretrained=True)
mobilenetv3_model.classifier[-1] =
nn.Linear(mobilenetv3_model.classifier[-1].in_features, num_classes)


# Train the model mobilen
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(mobilenetv3_model.parameters(), lr=0.001,
momentum=0.9)



mobilenetv3_model.to(device)
mobilenetv3_model.train()
mobilenetv3_model =
Simple_train(train_loader,mobilenetv3_model,criterion,optimizer)
```

(a)

```python
print("feature map of the first level features using Resnet-101")
plot_feature_map(resnet101_model,img_tensor,1)
print("feature map of the first level features using VGG16")
plot_feature_map(vgg16_model,img_tensor,1)
print("feature map of the first level features using MobilenetV3")
plot_feature_map(mobilenetv3_model,img_tensor,1)
```



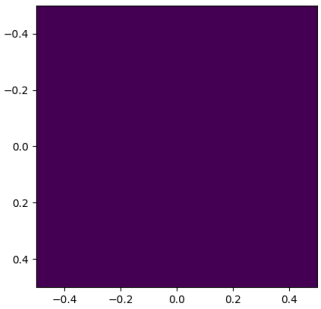


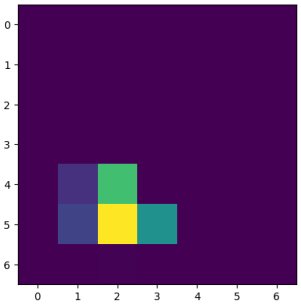


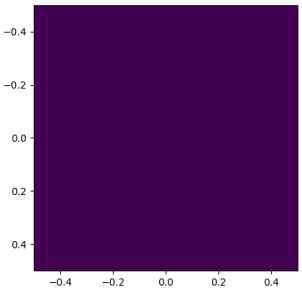
Actual Image

(b)

```python
print("feature map of the final level features using Resnet-101")
plot_feature_map(resnet101_model,img_tensor,-1)
print("feature map of the first level features using VGG16")
plot_feature_map(vgg16_model,img_tensor,-1)
print("feature map of the final level features using MobilenetV3")
plot_feature_map(mobilenetv3_model,img_tensor,-1)
```







Actual Image

2(c)

Python

```python
import numpy as np
from sklearn.neighbors import NearestNeighbors
from pathlib import Path
from PIL import Image
from torchvision import transforms, models
import torch
from tqdm.notebook import tqdm
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")



def plot_clg(image_paths):
  images = [Image.open(path) for path in image_paths]
  fig, axs = plt.subplots(1, 5, figsize=(10, 3))
  for i in range(len(images)):
    img_array = np.array(images[i])
    axs[i].imshow(img_array)
    axs[i].axis('off')
  plt.tight_layout()
  plt.show()



def extract_features(image_path, model):
  model.eval()
  preprocess = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
  ])

  image = Image.open(image_path).convert('RGB')
  input_image = preprocess(image)
  input_batch = torch.unsqueeze(input_image, 0).to(device)

  with torch.no_grad():
    output = model(input_batch)
```

```
  feature_vector = output.squeeze().cpu().numpy()

  return feature_vector



vgg16_model = models.vgg16(pretrained=True).to(device)



test_image_dir = Path('/kaggle/input/vegetable-image-dataset/Vegetable
Images/test')
validation_image_dir = Path('/kaggle/input/vegetable-image-dataset/Vegetable
Images/validation')



print("Finding the lowest number image from each category in the test
set.............")
lowest_number_images = {}
for category_dir in tqdm(test_image_dir.iterdir()):
  if category_dir.is_dir():
    images_in_category = list(category_dir.iterdir())
    lowest_number_image = min(images_in_category, key=lambda x: int(x.stem))
    lowest_number_images[category_dir.name] = lowest_number_image


print("Extracting features from the lowest number
images.....................")
query_features = {}
for category, image_path in tqdm(lowest_number_images.items()):
  features = extract_features(image_path, vgg16_model)
  query_features[category] = features


validation_features = []
validation_image_paths = []


print("Extracting features from all images in the validation
set.............")
for image_path in tqdm(validation_image_dir.glob('**/*.jpg')):
  features = extract_features(image_path, vgg16_model)
  validation_features.append(features)
  validation_image_paths.append(image_path)
```

```
query_features_array = np.array(list(query_features.values()))
validation_features_array = np.array(validation_features)

#find 5 nearest neighbors for each query
k_neighbors = 5
neigh = NearestNeighbors(n_neighbors=k_neighbors)
neigh.fit(validation_features_array)


nearest_neighbors_indices = []
for query_feature in query_features_array:
  _, indices = neigh.kneighbors([query_feature])
  nearest_neighbors_indices.append(indices.flatten())


for category, indices in zip(lowest_number_images.keys(),
nearest_neighbors_indices):
  print(f"Category: {category}")
  print("Nearest Neighbors:")
  temp=[]
  for index in indices:
    #print(validation_image_paths[index])
    temp.append(validation_image_paths[index])
  plot_clg(temp)
  print("\n")
```

Category: Broccoli
Nearest Neighbors:

Centered Image

Category: Capsicum
Nearest Neighbors:

Centered Image

Category: Bottle_Gourd
Nearest Neighbors:

Centered Image

Category: Radish
Nearest Neighbors:





Centered Image

Category: Tomato
Nearest Neighbors:

Centered Image

Category: Brinjal
Nearest Neighbors:

Category: Pumpkin
Nearest Neighbors:





Category: Carrot
Nearest Neighbors:





Centered Image

Category: Papaya

Nearest Neighbors:

Category: Cabbage
Nearest Neighbors:

Centered Image

Category: Bitter_Gourd

Nearest Neighbors:

Actual Image

Category: Cauliflower
Nearest Neighbors:

Centered Image

Category: Bean
Nearest Neighbors:

Centered Image

Category: Cucumber
Nearest Neighbors:

Category: Potato
Nearest Neighbors:

Centered Image

Question 3

Python

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from sklearn.metrics import confusion_matrix, accuracy_score
import time



# Defining data transformation
transform = transforms.Compose([
  transforms.Resize((224, 224)),
  transforms.ToTensor(),
])


# Loading and validation datasets
train_dataset =
torchvision.datasets.ImageFolder(root='/kaggle/input/vegetable-image-dataset/Ve
getable Images/test', transform=transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True,
num_workers=4)

val_dataset =
torchvision.datasets.ImageFolder(root='/kaggle/input/vegetable-image-dataset/Ve
getable Images/validation', transform=transform)
```

```
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=4)


num_classes = 15



# Define VGG16 model
vgg_model = torchvision.models.vgg16(pretrained=True)
# Modifing the last layer
vgg_model.classifier[6] = nn.Linear(4096, num_classes)


# Define ResNet-101 model
resnet_model = torchvision.models.resnet101(pretrained=True)
#Modifing the last layer
resnet_model.fc = nn.Linear(2048, num_classes)
```

Python

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
vgg_model = vgg_model.to(device)
resnet_model = resnet_model.to(device)
```

Python

```
# Defining optimizer and loss function
criterion = nn.CrossEntropyLoss()
optimizer_vgg = optim.Adam(vgg_model.parameters(), lr=0.001)
optimizer_resnet = optim.SGD(resnet_model.parameters(), lr=0.01, momentum=0.9)
```

Python

```
# Training process
def train(model, train_loader, optimizer, criterion, device):
  model.train()
  running_loss = 0.0
```

```
  correct_predictions = 0
  total_samples = 0

  for inputs, labels in train_loader:
    inputs, labels = inputs.to(device), labels.to(device)

    optimizer.zero_grad()

    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

    running_loss += loss.item()

    _, preds = torch.max(outputs, 1)
    correct_predictions += torch.sum(preds == labels.data)
    total_samples += labels.size(0)

  accuracy = correct_predictions.double() / total_samples

  return running_loss / len(train_loader), accuracy



# Validation process
def validate(model, val_loader, criterion, device):
  model.eval()
  all_preds = []
  all_labels = []

  with torch.no_grad():
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    for inputs, labels in val_loader:
      inputs, labels = inputs.to(device), labels.to(device)

      outputs = model(inputs)
      loss = criterion(outputs, labels)
      running_loss += loss.item()

      _, preds = torch.max(outputs, 1)
```

```
            correct_predictions += torch.sum(preds == labels.data)
            total_samples += labels.size(0)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    accuracy = accuracy_score(all_labels, all_preds)
    confusion_mat = confusion_matrix(all_labels, all_preds)

    return running_loss / len(val_loader), accuracy, confusion_mat
```

Python

```
# Training loop
num_epochs = 5
start_time = time.time()

train_losses_vgg = []
val_losses_vgg = []
train_acc_vgg = []
val_acc_vgg = []


train_losses_resnet = []
val_losses_resnet = []
train_acc_resnet = []
val_acc_resnet = []


for epoch in range(num_epochs):

    train_loss_vgg_score, train_accuracy_vgg_score = train(vgg_model, train_loader,
optimizer_vgg, criterion, device)
    val_loss_vgg_score, val_acc_vgg_score, _ = validate(vgg_model, val_loader,
criterion, device)

    train_loss_resnet_score, train_accuracy_resnet_score = train(resnet_model,
train_loader, optimizer_resnet, criterion, device)
    val_loss_resnet_score, val_acc_resnet_score, _ = validate(resnet_model,
val_loader, criterion, device)

    train_losses_vgg.append(train_loss_vgg_score)
```

```
  val_losses_vgg.append(val_loss_vgg_score)
  train_acc_vgg.append(train_accuracy_vgg_score.item())
  val_acc_vgg.append(val_acc_vgg_score.item())

  train_losses_resnet.append(train_loss_resnet_score)
  val_losses_resnet.append(val_loss_resnet_score)
  train_acc_resnet.append(train_accuracy_resnet_score.item())
  val_acc_resnet.append(val_acc_resnet_score)

print('------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------')
  print(f"Epoch {epoch+1}/{num_epochs} - VGG: Train Loss:
{train_loss_vgg_score:.4f}, Train Acc:
{train_accuracy_vgg_score:.4f}.................. Val Loss:
{val_loss_vgg_score:.4f}, Val Acc: {val_acc_vgg_score:.4f}")
  print(f"Epoch {epoch+1}/{num_epochs} - ResNet: Train Loss:
{train_loss_resnet_score:.4f}, Train Acc:
{train_accuracy_resnet_score:.4f}.............. Val Loss:
{val_loss_resnet_score:.4f}, Val Acc: {val_acc_resnet_score:.4f}")

print('------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------')

end_time = time.time()
```

output:

```
Python
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------
Epoch 1/5 - VGG: Train Loss: 2.6956, Train Acc: 0.1087.................. Val Loss:
2.4158, Val Acc: 0.1653
Epoch 1/5 - ResNet: Train Loss: 0.4114, Train Acc: 0.8817.............. Val Loss:
0.1997, Val Acc: 0.9527
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------------------
Epoch 2/5 - VGG: Train Loss: 2.1672, Train Acc: 0.2573.................. Val Loss:
1.9613, Val Acc: 0.3370
Epoch 2/5 - ResNet: Train Loss: 0.1016, Train Acc: 0.9717.............. Val Loss:
0.0338, Val Acc: 0.9893
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------------------
Epoch 3/5 - VGG: Train Loss: 1.7870, Train Acc: 0.3797.................. Val Loss:
1.5250, Val Acc: 0.4777
Epoch 3/5 - ResNet: Train Loss: 0.0503, Train Acc: 0.9860.............. Val Loss:
0.0587, Val Acc: 0.9823
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------------------
Epoch 4/5 - VGG: Train Loss: 1.4761, Train Acc: 0.4870.................. Val Loss:
1.6112, Val Acc: 0.4380
Epoch 4/5 - ResNet: Train Loss: 0.0166, Train Acc: 0.9957.............. Val Loss:
0.0635, Val Acc: 0.9823
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------------------
Epoch 5/5 - VGG: Train Loss: 1.4045, Train Acc: 0.5223.................. Val Loss:
1.3185, Val Acc: 0.5647
Epoch 5/5 - ResNet: Train Loss: 0.0101, Train Acc: 0.9973.............. Val Loss:
0.0163, Val Acc: 0.9950
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------------------
```

```python
# Calculate training time
training_time = end_time - start_time
print(f"Training Time: {training_time:.2f} seconds")
```

output:

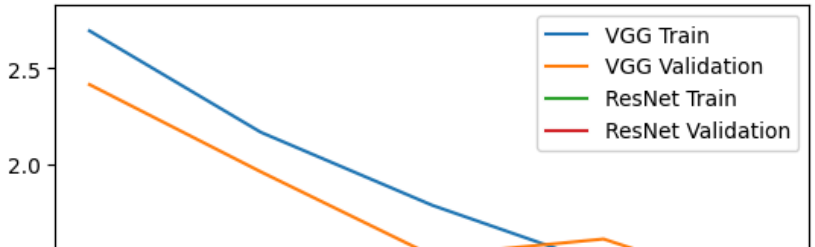
```
Training Time: 337.79 seconds
```

```python
# Print GPU memory footprint
print(f"Memory Usage: {torch.cuda.memory_allocated(device) / 1e9} GB")
```

ouput:

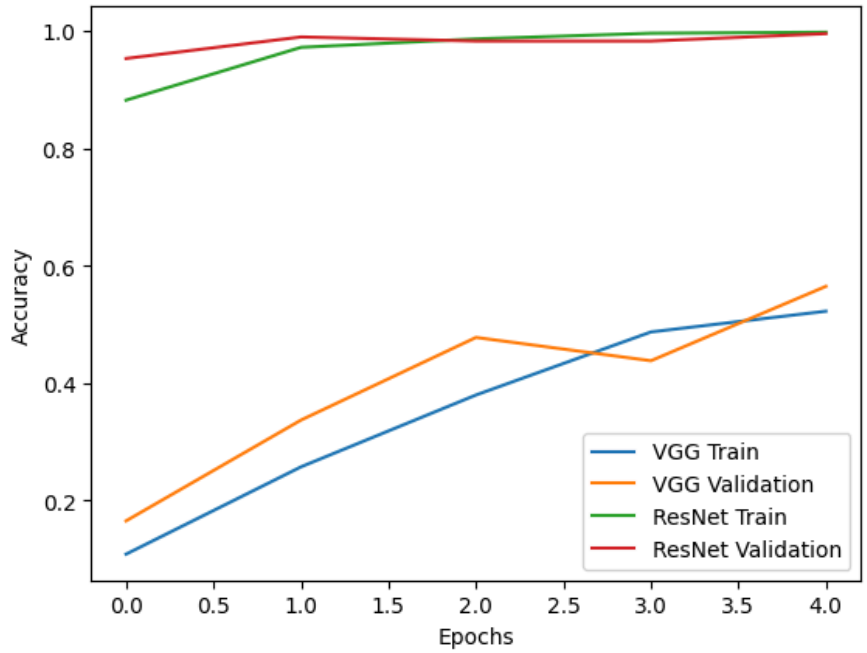
```
Memory Usage: 2.69265664 GB
```

```python
# Plot training and validation loss curves
import matplotlib.pyplot as plt

plt.plot(train_losses_vgg, label='VGG Train')
plt.plot(val_losses_vgg, label='VGG Validation')
plt.plot(train_losses_resnet, label='ResNet Train')
plt.plot(val_losses_resnet, label='ResNet Validation')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Python

```python
import matplotlib.pyplot as plt
# Plot training and validation accuracy curves
plt.plot(train_acc_vgg, label='VGG Train')
plt.plot(val_acc_vgg, label='VGG Validation')
plt.plot(train_acc_resnet, label='ResNet Train')
plt.plot(val_acc_resnet, label='ResNet Validation')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



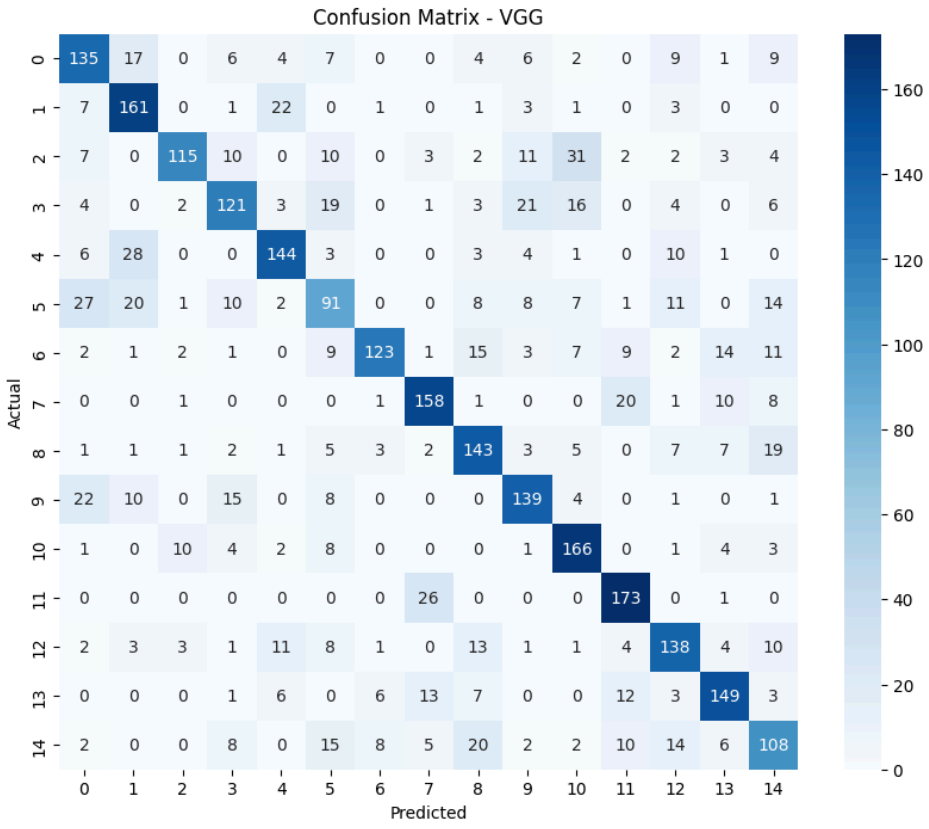
Python

```python
# Testing process
def test(model, test_loader, criterion, device):
  model.eval()
  all_preds = []
  all_labels = []

  with torch.no_grad():
```

```
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    for inputs, labels in test_loader:
      inputs, labels = inputs.to(device), labels.to(device)

      outputs = model(inputs)
      loss = criterion(outputs, labels)
      running_loss += loss.item()

      _, preds = torch.max(outputs, 1)

      correct_predictions += torch.sum(preds == labels.data)
      total_samples += labels.size(0)

      all_preds.extend(preds.cpu().numpy())
      all_labels.extend(labels.cpu().numpy())

  accuracy = accuracy_score(all_labels, all_preds)
  confusion_mat = confusion_matrix(all_labels, all_preds)

  return running_loss / len(test_loader), accuracy, confusion_mat


# Testing the models
test_dataset =
torchvision.datasets.ImageFolder(root='/kaggle/input/vegetable-image-dataset/Ve
getable Images/test', transform=transform)
test_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=4)

test_loss_vgg, test_acc_vgg, confusion_mat_vgg = test(vgg_model, test_loader,
criterion, device)
test_loss_resnet, test_acc_resnet, confusion_mat_resnet = test(resnet_model,
test_loader, criterion, device)


# Plot confusion matrix for VGG
print("VGG Test Results:")
print(f"Test Loss: {test_loss_vgg:.4f}, Test Accuracy: {test_acc_vgg:.4f}")
print("Confusion Matrix:")
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mat_vgg, annot=True, fmt="d", cmap="Blues",
xticklabels=range(num_classes), yticklabels=range(num_classes))
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
plt.title("Confusion Matrix - VGG")
plt.show()

# Plot confusion matrix for Resnet
print("\nResNet Test Results:")
print(f"Test Loss: {test_loss_resnet:.4f}, Test Accuracy:
{test_acc_resnet:.4f}")
print("Confusion Matrix:")
print(confusion_mat_resnet)

# Plot confusion matrix for ResNet
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mat_resnet, annot=True, fmt="d", cmap="Blues",
xticklabels=range(num_classes), yticklabels=range(num_classes))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - ResNet")
plt.show()
```

VGG Test Results:
Test Loss: 0.9910, Test Accuracy: 0.6880
Confusion Matrix:



ResNet Test Results:
Test Loss: 0.0174, Test Accuracy: 0.9933
Confusion Matrix: