shufflenet2new_batch_size_32

April 17, 2023

[ ]:
```
#import packages
import os
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
import torchvision.models as models
import glob
import shutil
import random
from tqdm import tqdm
import matplotlib.pyplot as plt
from torch.utils.data import Dataset
import datasets
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
leaf_datasets = ImageFolder(
    '/content/drive/MyDrive/AI Project/Dataset1/PlantVillage_15 classes/',
    transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
↪225]),
    ])
)
```

[ ]:
```
mkdir PVdatasetsplit
```

[ ]:
```
#Splitting the data

# Defining the path to dataset
```

```
dataset_path = '/content/drive/MyDrive/AI Project/Dataset1/PlantVillage_15␣
↪classes/'

train_ratio = 0.75
valid_ratio = 0.10
test_ratio = 0.15

# The path to the output directory
output_path = '/content/drive/MyDrive/AI Project/Dataset1/working/
↪PVdatasetsplit/'

# Create the output directory if it doesn't exist
if not os.path.exists(output_path):
    os.makedirs(output_path)

# Defining the names of the subdirectories for each set
train_dir = 'train'
valid_dir = 'valid'
test_dir = 'test'

# Creating the subdirectories for each set
os.makedirs(os.path.join(output_path, train_dir))
os.makedirs(os.path.join(output_path, valid_dir))
os.makedirs(os.path.join(output_path, test_dir))

# Loop over each class in the dataset
classes = os.listdir(dataset_path)
for cls in classes:
    # Create the subdirectories for each class in each set
    os.makedirs(os.path.join(output_path, train_dir, cls))
    os.makedirs(os.path.join(output_path, valid_dir, cls))
    os.makedirs(os.path.join(output_path, test_dir, cls))

    # Get the list of images for this class
    images = os.listdir(os.path.join(dataset_path, cls))
    num_images = len(images)

    # Shuffle the images
    random.shuffle(images)

    # Split the images into sets
    num_train = int(train_ratio * num_images)
    num_valid = int(valid_ratio * num_images)
    num_test = int(test_ratio * num_images)

    train_images = images[:num_train]
    valid_images = images[num_train:num_train+num_valid]
```

```python
    test_images = images[num_train+num_valid:]

    # Copy the images to the corresponding subdirectories for each set
    for img in train_images:
        src_path = os.path.join(dataset_path, cls, img)
        print(img)
        dst_path = os.path.join(output_path, train_dir, cls, img)
        shutil.copyfile(src_path, dst_path)

    for img in valid_images:
        src_path = os.path.join(dataset_path, cls, img)
        dst_path = os.path.join(output_path, valid_dir, cls, img)
        shutil.copyfile(src_path, dst_path)

    for img in test_images:
        src_path = os.path.join(dataset_path, cls, img)
        dst_path = os.path.join(output_path, test_dir, cls, img)
        shutil.copyfile(src_path, dst_path)
```

[ ]:

```python
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize the images to 224x224
    transforms.RandomHorizontalFlip(),  # Randomly flip the images horizontally
    transforms.RandomRotation(10),  # Randomly rotate the images by up to 10␣
 ↪degrees
    transforms.ToTensor(),  # Convert the images to PyTorch tensors
    transforms.Normalize(  # Normalize the images
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

batch_size = 32
# Load the dataset
train_dataset = ImageFolder('/content/drive/MyDrive/AI Project/Dataset1/working/
 ↪PVdatasetsplit/train', transform=transform)
test_dataset = ImageFolder('/content/drive/MyDrive/AI Project/Dataset1/working/
 ↪PVdatasetsplit/test', transform=transform)
val_dataset = ImageFolder('/content/drive/MyDrive/AI Project/Dataset1/working/
 ↪PVdatasetsplit/valid', transform=transform)

# Create data loaders
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size,␣
 ↪shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size,␣
 ↪shuffle=False)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size, shuffle=False)
```

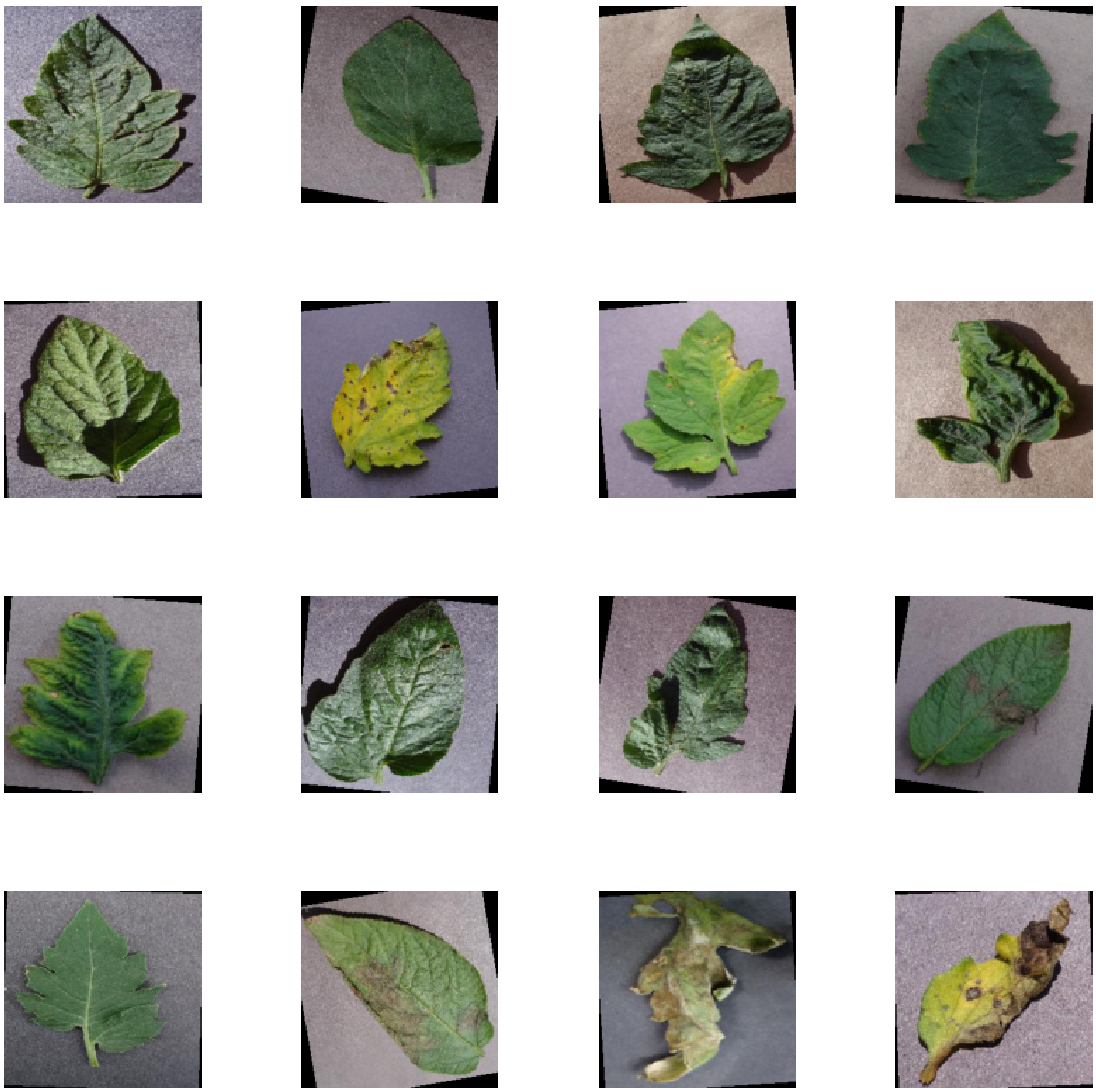
[ ]:
```
print(len(train_loader))
print(len(test_loader))
print(len(val_loader))
```

```
483
97
65
```

[ ]:
```
# Define the model architecture
model = models.shufflenet_v2_x2_0(weights=None)
model.classifier = nn.Sequential(
    nn.Linear(1280, 1024),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(1024, 512),
    nn.ReLU(),
    nn.Dropout(0.5),

    nn.Linear(512, 256),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(256, 15)
)
```

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import torch

# Get a batch of images from the train loader
images, labels = next(iter(train_loader))

# Convert the PyTorch tensor to a NumPy array
images = images.numpy()

# Convert the NumPy array to a PyTorch tensor
images = torch.from_numpy(images)

# Denormalize the images
mean = torch.tensor([0.485, 0.456, 0.406])
std = torch.tensor([0.229, 0.224, 0.225])
images = images.permute(0, 2, 3, 1)  # Change the order of the dimensions
images = std * images + mean

# Create a figure with 4x4 subplots
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(10, 10))
fig.subplots_adjust(hspace=0.5, wspace=0.5) # Adjust subplot parameters
for i, ax in enumerate(axes.flat):
```

```
    # Display the image
    ax.imshow(images[i])
    ax.axis('off')
plt.show()
```



```
[ ]: # Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Define the device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Training on device {device}.")
```

```python
# Move model to the device
model = model.to(device)

# Define the number of epochs to train for
num_epochs = 10

# Create lists to store train and validation loss and accuracy
train_loss_list = []
train_acc_list = []
val_loss_list = []
val_acc_list = []

# Train the model
for epoch in range(1, num_epochs+1):
    # Set the model to training mode
    model.train()
    train_loss = 0
    total_train_images = 0
    total_train_correct = 0

    # Loop over the training dataset in batches
    for images, labels in tqdm(train_loader, desc=f'Epoch {epoch}/
 ↪{num_epochs}'):
        # Move data to the device
        images, labels = images.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

        # Calculate training loss and accuracy
        train_loss += loss.item() * labels.size(0)
        _, predicted = torch.max(outputs, 1)
        total_train_correct += (predicted == labels).sum().item()
        total_train_images += labels.size(0)

    # Calculate training accuracy
    train_acc = total_train_correct / total_train_images

    # Evaluate the model on the validation set
```

```
    model.eval()
    with torch.no_grad():
        total_val_loss = 0
        total_val_correct = 0
        total_val_images = 0
        for images, labels in val_loader:
            # Move data to the device
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)
            total_val_loss += loss.item() * labels.size(0)
            _, predicted = torch.max(outputs, 1)
            total_val_correct += (predicted == labels).sum().item()
            total_val_images += labels.size(0)
        val_acc = total_val_correct / total_val_images
        print('Epoch [{}/{}], Training Loss: {:.4f}, Training Accuracy: {:.
↪2f}%, Validation Loss: {:.4f}, Validation Accuracy: {:.2f}%'
              .format(epoch, num_epochs, train_loss/len(train_loader.dataset),␣
↪train_acc*100, total_val_loss/len(val_loader.dataset), val_acc*100))

    # Append training and validation metrics to the corresponding lists
    train_loss_list.append(train_loss/len(train_loader.dataset))
    train_acc_list.append(train_acc*100)
    val_loss_list.append(total_val_loss/len(val_loader.dataset))
    val_acc_list.append(val_acc*100)
```

```
Training on device cuda.

Epoch 1/10: 100%|      | 483/483 [01:55<00:00,  4.19it/s]

Epoch [1/10], Training Loss: 1.1378, Training Accuracy: 62.99%, Validation Loss:
0.8271, Validation Accuracy: 71.85%

Epoch 2/10: 100%|      | 483/483 [01:54<00:00,  4.23it/s]

Epoch [2/10], Training Loss: 0.5342, Training Accuracy: 81.61%, Validation Loss:
0.3624, Validation Accuracy: 87.29%

Epoch 3/10: 100%|      | 483/483 [01:54<00:00,  4.22it/s]

Epoch [3/10], Training Loss: 0.3581, Training Accuracy: 87.89%, Validation Loss:
0.2856, Validation Accuracy: 90.50%

Epoch 4/10: 100%|      | 483/483 [01:56<00:00,  4.15it/s]

Epoch [4/10], Training Loss: 0.2765, Training Accuracy: 90.40%, Validation Loss:
0.2661, Validation Accuracy: 91.18%

Epoch 5/10: 100%|      | 483/483 [01:54<00:00,  4.23it/s]
```

```
Epoch [5/10], Training Loss: 0.2279, Training Accuracy: 92.30%, Validation Loss:
0.2332, Validation Accuracy: 92.60%

Epoch 6/10: 100%|          | 483/483 [01:54<00:00,  4.23it/s]

Epoch [6/10], Training Loss: 0.1959, Training Accuracy: 93.18%, Validation Loss:
0.1284, Validation Accuracy: 95.66%

Epoch 7/10: 100%|          | 483/483 [01:54<00:00,  4.22it/s]

Epoch [7/10], Training Loss: 0.1810, Training Accuracy: 93.77%, Validation Loss:
0.3672, Validation Accuracy: 89.04%

Epoch 8/10: 100%|          | 483/483 [01:54<00:00,  4.22it/s]

Epoch [8/10], Training Loss: 0.1502, Training Accuracy: 94.80%, Validation Loss:
0.1088, Validation Accuracy: 96.20%

Epoch 9/10: 100%|          | 483/483 [01:53<00:00,  4.26it/s]

Epoch [9/10], Training Loss: 0.1276, Training Accuracy: 95.78%, Validation Loss:
0.1481, Validation Accuracy: 94.64%

Epoch 10/10: 100%|          | 483/483 [01:53<00:00,  4.25it/s]

Epoch [10/10], Training Loss: 0.1276, Training Accuracy: 95.69%, Validation
Loss: 0.1193, Validation Accuracy: 95.57%
```

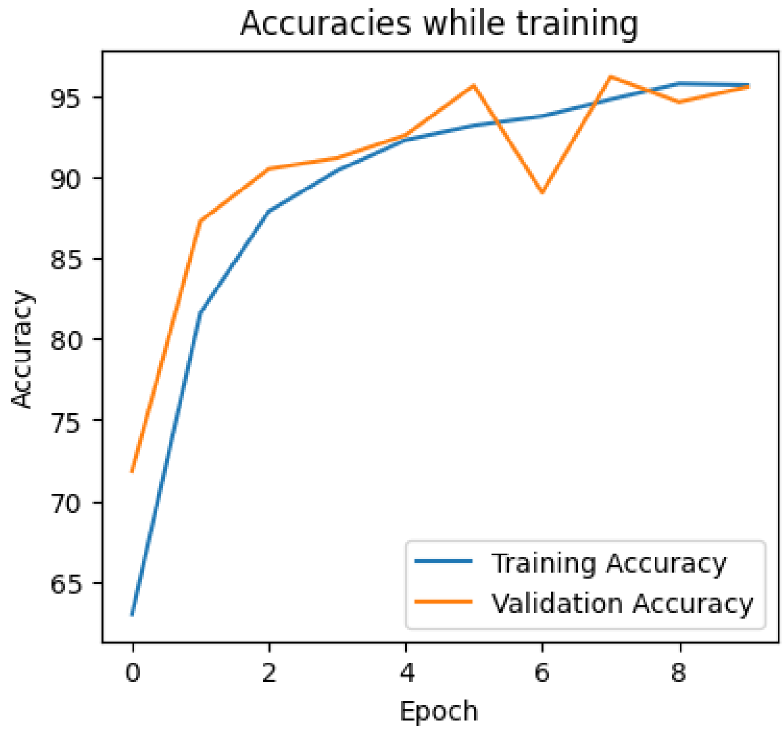
[ ]:

```python
# Train the model
num_epochs=10
for epoch in range(16, num_epochs+1):
    # Set the model to training mode
    model.train()
    train_loss = 0
    total_train_images = 0
    total_train_correct = 0

    # Loop over the training dataset in batches
    for images, labels in tqdm(train_loader, desc=f'Epoch {epoch}/
 ↪{num_epochs}'):
        # Move data to the device
        images, labels = images.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        loss.backward()
```

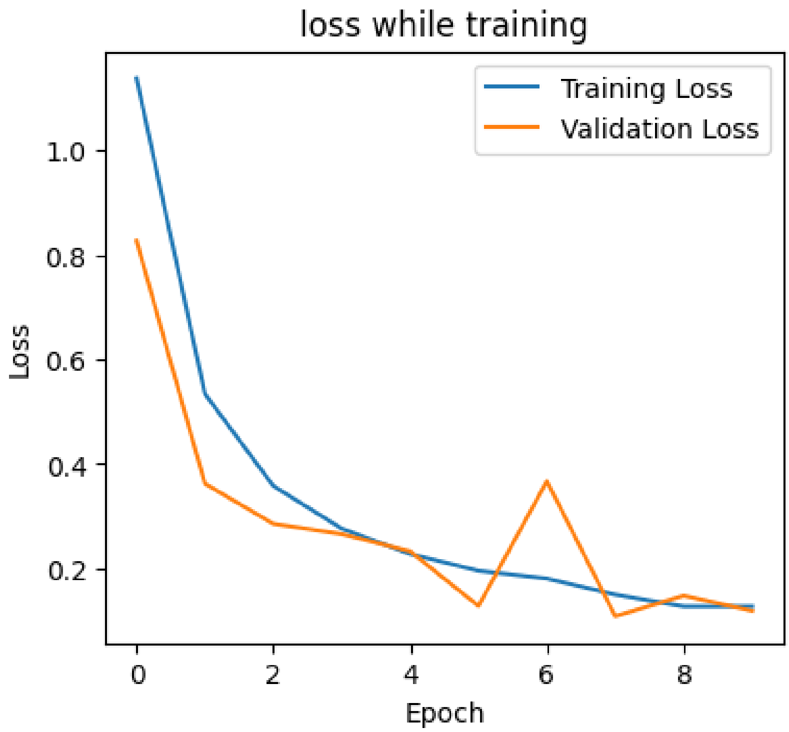
```
        optimizer.step()

        # Calculate training loss and accuracy
        train_loss += loss.item() * labels.size(0)
        _, predicted = torch.max(outputs, 1)
        total_train_correct += (predicted == labels).sum().item()
        total_train_images += labels.size(0)

    # Calculate training accuracy
    train_acc = total_train_correct / total_train_images

    # Evaluate the model on the validation set
    model.eval()
    with torch.no_grad():
        total_val_loss = 0
        total_val_correct = 0
        total_val_images = 0
        for images, labels in val_loader:
            # Move data to the device
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)
            total_val_loss += loss.item() * labels.size(0)
            _, predicted = torch.max(outputs, 1)
            total_val_correct += (predicted == labels).sum().item()
            total_val_images += labels.size(0)
        val_acc = total_val_correct / total_val_images
        print('Epoch [{}/{}], Training Loss: {:.4f}, Training Accuracy: {:.
↪2f}%, Validation Loss: {:.4f}, Validation Accuracy: {:.2f}%'
              .format(epoch, num_epochs, train_loss/len(train_loader.dataset),␣
↪train_acc*100, total_val_loss/len(val_loader.dataset), val_acc*100))

    # Append training and validation metrics to the corresponding lists
    train_loss_list.append(train_loss/len(train_loader.dataset))
    train_acc_list.append(train_acc*100)
    val_loss_list.append(total_val_loss/len(val_loader.dataset))
    val_acc_list.append(val_acc*100)
```

[ ]:
```
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 2)
plt.plot(train_acc_list, label='Training Accuracy')
plt.plot(val_acc_list, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracies while training')
```

```
plt.show()
```



[ ]:
```
# Plot the training and validation loss and accuracy
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(train_loss_list, label='Training Loss')
plt.plot(val_loss_list, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('loss while training')
plt.show()
```



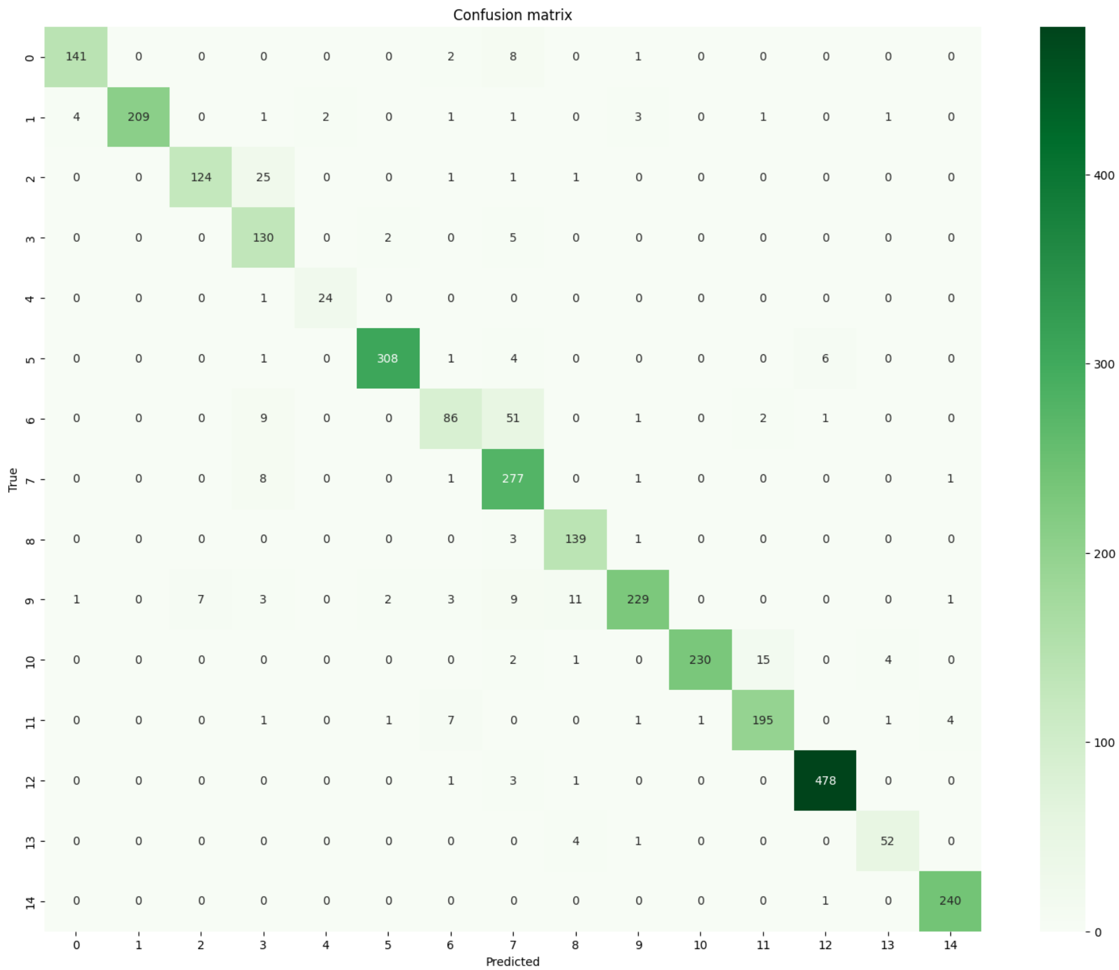
[ ]:
```
# Evaluate the model on the test set
model.eval()
with torch.no_grad():
    total_test_loss = 0
    total_correct = 0
    total_images = 0
    # Use tqdm to add a progress bar
    for images, labels in tqdm(test_loader):
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        total_test_loss += loss.item() * labels.size(0)
        _, predicted = torch.max(outputs, 1)
        total_correct += (predicted == labels).sum().item()
        total_images += labels.size(0)
    test_loss = total_test_loss / total_images
    accuracy = total_correct / total_images
    print('Test Loss: {:.4f}, Test Accuracy: {:.2f}%'.format(test_loss,
 ↪accuracy*100))
```

```
100%|          | 97/97 [00:18<00:00,  5.20it/s]

Test Loss: 0.2384, Test Accuracy: 92.55%
```

[ ]:
```python
from sklearn.metrics import confusion_matrix, precision_score, recall_score,␣
↪f1_score

# Set the model to evaluation mode
model.eval()

# Initialize lists to store true labels and predicted labels
true_labels = []
pred_labels = []

# Loop over the validation dataset in batches
for images, labels in test_loader:
    images, labels = images.to(device), labels.to(device)
    # Predict the labels
    outputs = model(images)
    _, predicted = torch.max(outputs, 1)

    # Append the true and predicted labels to the corresponding lists
    true_labels.extend(labels.tolist())
    pred_labels.extend(predicted.tolist())

# Compute the confusion matrix
conf_matrix = confusion_matrix(true_labels, pred_labels)

# Compute precision, recall, and F-score
precision = precision_score(true_labels, pred_labels, average='macro')
recall = recall_score(true_labels, pred_labels, average='macro')
f_score = f1_score(true_labels, pred_labels, average='macro')

print('Precision: {:.4f}, Recall: {:.4f}, F-score: {:.4f}'.format(precision,␣
↪recall, f_score))
```

```
Precision: 0.9171, Recall: 0.9103, F-score: 0.9094
```

[ ]:
```python
# Plot the confusion matrix
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(18, 14))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens')
plt.title('Confusion matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



[ ]:
```
torch.save({
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict()
}, 'dataset2')
```

[ ]:
```
torch.save(model.state_dict(), 'best_model.pt')
print("model saved")
```

[ ]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
```

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[6]:
```
!cp 'drive/My Drive/Colab Notebooks/shufflenet2new_batch_size_32.ipynb' ./
```

[ ]:
```
!apt-get update
```

[11]:
```
!apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic
```

```
done.
```

[7]:
```
!jupyter nbconvert --to PDF "shufflenet2new_batch_size_32.ipynb"
```

```
[NbConvertApp] Converting notebook shufflenet2new_batch_size_32.ipynb to PDF
[NbConvertApp] ERROR | Notebook JSON is invalid: Additional properties are not
allowed ('metadata' was unexpected)

Failed validating 'additionalProperties' in stream:

On instance['cells'][9]['outputs'][0]:
{'metadata': {'tags': None},
 'name': 'stdout',
 'output_type': 'stream',
 'text': 'Training on device cuda.\n'}
[NbConvertApp] Support files will be in shufflenet2new_batch_size_32_files/
[NbConvertApp] Making directory ./shufflenet2new_batch_size_32_files
[NbConvertApp] Making directory ./shufflenet2new_batch_size_32_files
[NbConvertApp] Making directory ./shufflenet2new_batch_size_32_files
[NbConvertApp] Making directory ./shufflenet2new_batch_size_32_files
[NbConvertApp] Writing 70618 bytes to notebook.tex
[NbConvertApp] Building PDF
Traceback (most recent call last):
  File "/usr/local/bin/jupyter-nbconvert", line 8, in <module>
    sys.exit(main())
  File "/usr/local/lib/python3.9/dist-packages/jupyter_core/application.py",
line 277, in launch_instance
    return super().launch_instance(argv=argv, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/traitlets/config/application.py",
line 992, in launch_instance
    app.start()
  File "/usr/local/lib/python3.9/dist-packages/nbconvert/nbconvertapp.py", line
423, in start
    self.convert_notebooks()
  File "/usr/local/lib/python3.9/dist-packages/nbconvert/nbconvertapp.py", line
597, in convert_notebooks
    self.convert_single_notebook(notebook_filename)
  File "/usr/local/lib/python3.9/dist-packages/nbconvert/nbconvertapp.py", line
560, in convert_single_notebook
    output, resources = self.export_single_notebook(
  File "/usr/local/lib/python3.9/dist-packages/nbconvert/nbconvertapp.py", line
488, in export_single_notebook
    output, resources = self.exporter.from_filename(
  File "/usr/local/lib/python3.9/dist-packages/nbconvert/exporters/exporter.py",
```

```
line 189, in from_filename
    return self.from_file(f, resources=resources, **kw)
  File "/usr/local/lib/python3.9/dist-packages/nbconvert/exporters/exporter.py",
line 206, in from_file
    return self.from_notebook_node(
  File "/usr/local/lib/python3.9/dist-packages/nbconvert/exporters/pdf.py", line
194, in from_notebook_node
    self.run_latex(tex_file)
  File "/usr/local/lib/python3.9/dist-packages/nbconvert/exporters/pdf.py", line
164, in run_latex
    return self.run_command(
  File "/usr/local/lib/python3.9/dist-packages/nbconvert/exporters/pdf.py", line
111, in run_command
    raise OSError(
OSError: xelatex not found on PATH, if you have not installed xelatex you may
need to do so. Find further instructions at
https://nbconvert.readthedocs.io/en/latest/install.html#installing-tex.
```