Flower Specie Classification Using Machine Learning

CEP Report

By

NAME	RegistrationNumber
Farhan Ali Khan	CUI/FA21-BCE-001/LHR
Muhammad Faizan	CUI/FA21-BCE-041/LHR
Fouad Aziz	CUI/FA21-BCE-061/LHR
Osama Aftab	CUI/FA19-BCE-067/LHR

For the course

Machine Learning

Semester Spring 2025

Supervised by:

Dr. Ikramullah Khosa

Department of Computer Engineering

COMSATS University Islamabad – Lahore Campus

DECLARATION

We Student 1 (CUI/FA21-BCE-001/LHR), Student 2 (CUI/FA21-BCE-041/LHR), Student 3 (CUI/FA21-BCE-061/LHR) and Student 4 (CUI/FA19-BCE-067/LHR) hereby declare that we have produced the work presented in this report, during the scheduled period of study. We also declare that we have not taken any material from any source except referred to wherever due. If a violation of rules has occurred in this report, we shall be liable to punishable action.

Date: ____________________

____

Student 1
(CUI/FA21-
BCE-001/LHR)

____

Student 2
(CUI/FA21-
BCE-041/LHR)

____

Student 3
(CUI/FA21-
BCE-061/LHR)

____

Student 4
(CUI/FA19-
BCE-067/LHR)

ABSTRACT

This project presents a comparative study of different model architectures used for multiclass flower specie classification. We chose three architectures i.e., A simple deep neural network, LeNet-5 based CNN architecture, and a ResNet18 Architecture on which transfer learning is applied. We have performed preprocessing and augmentation on the dataset consisting of approx. total 14 thousand images divided into 14 different categories of species. Resultantly, we were able to showcase the accuracies of three mentioned models as a result of which the ResNet architecture is considered the best one in terms of the parameters count and the test and validation accuracy. The results mentioned highlight that the model architecture choice plays a significant role in improving the performance of the model. The transfer learning utilized by ResNet18, provides a significant advantage of using the pre-trained weights as compared to training the weights from scratch. Moreover, this project highlights the practical approach of developing a model for use in mobile in web applications in order to overcome the problems faced by layman.

Table of Contents

Table of Figure

1. Introduction

In recent years, the use of artificial intelligence specifically computer vision in image classification has been expanded significantly such as that in self-driving cars. One similar application is the identification of different flower species which has implications in fields like horticulture and botany. Classification of flower species using AI can help plants hobbyists, botanist, researchers and farmers to identify the type without having to look up in different guidebooks, which may be time consuming. Considering the ongoing trend of generative AI, it can be further expanded to generate the care guidelines for a particular type of flower specie.

This project aims to compare three classification models trained on the dataset consisting of total 13,716 images of 14 different types of flowers. The three models are LeNet based CNN, ResNet18 model utilizing transfer learning and a traditional Deep Neural Network used to emphasize the significance of the convolution based neural networks. Each model is evaluated on the training, validation and test accuracy as well as the number of parameters of each. PyTorch framework is utilized for model training and image preprocessing.

2. Literature Survey

Image Classification is one of the widely researched areas of computer vision in machine learning. Traditional classification algorithms like K-Nearest-Neighbors and Support Vector Machines works fine on data consisting of handcrafted features. But for higher dimensional data like images, traditional algorithms fail. That is why the Convolution Neural Network came into place. These networks efficiently learn the higher dimension features like the images.

LeNet-5, proposed by Yann LeCun in 1998, is one of the earliest CNN architectures which was initially used for handwritten digit classification. Despite its simple architecture, it has proved to be very useful for the task comprising of low-resolution images.

ResNet or Residual Network introduced in 2015 addressed the problem of vanishing gradients in traditional CNN architectures by making the use of skip connections. ResNet18, a variant consisting 18 layers has shown a promising benchmark on various image classification tasks and is used in transfer learning scenarios due to the availability of pre-trained weights which can be used.

Deep Neural Networks (DNN) consists of fully connected layers without convolution layers which theoretically capture the complexity of multidimensional data but often suffers from overfitting and poor generalization on image data. However, they are still used for the comparison with CNNs to highlight the importance of convolution layers in capturing image data features.

Above mentioned work form the foundation for this project, which aims to classify the flower species categories into 14 different ones.

3. Dataset

3.1. Data Visualization

Before training any machine learning model, It is important to visualization the data distribution to understand each category and analyze their color saturation, resolution and image size.

The flower specie dataset used in this project consists of 14 different classes (carnation, iris, bluebells, golden english, roses, fallen nephews, tulips, marigolds, dandelions, chrysanthemums, black-eyed daisies, water lilies, sunflowers, and daisies) with 13, 618 training images and 98 validation images.

Following is the sample plot of the images showing random types:

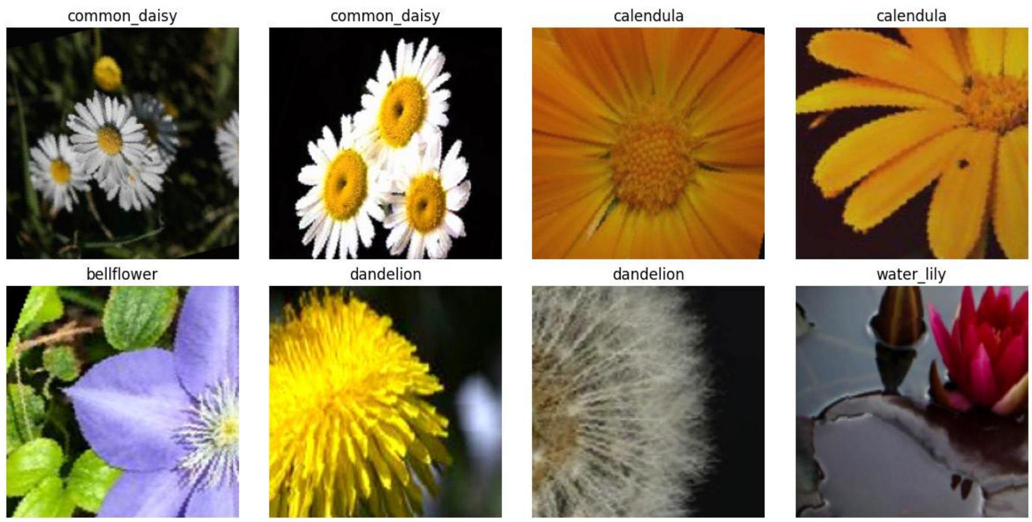
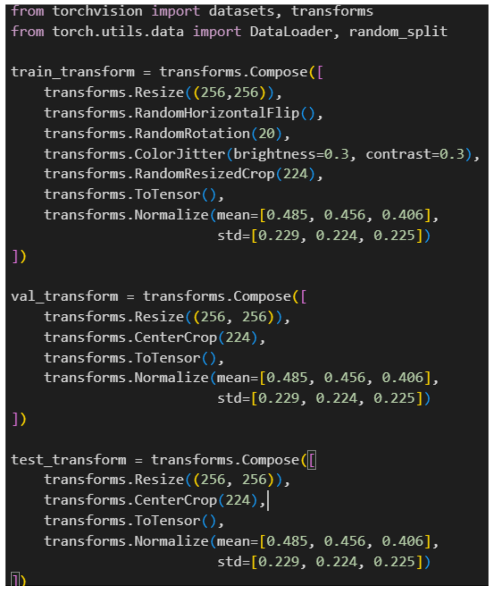


Figure 1: Random Sample Dataset Plot

The data has been split in such a way that the training data is divided into train and validation data, resulting in the train size of 10913 and validation size of 2729 images. Whereas the validation images are used as testing data of 98 images.

3.2. Data Preprocessing

Data preprocessing is a crucial step in machine learning / deep learning workflows to squeeze the best performance out of the model. In regard to this project, the preprocessing steps are explained below.



```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, random_split

train_transform = transforms.Compose([
    transforms.Resize((256,256)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(20),
    transforms.ColorJitter(brightness=0.3, contrast=0.3),
    transforms.RandomResizedCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Figure 2: Image Transforms Using TorchVision

Each image is resized to 256x256 resolution and then center cropped to 224, making each image 224x224 with data augmentation (for training data only) like random flips and cropping.

Additionally, each image is normalized using mean and standard deviation of the ImageNet dataset values. This step ensures that the dataset is aligned with the input of pretrained models like ResNet18.

4. Methodology

4.1. LeNet

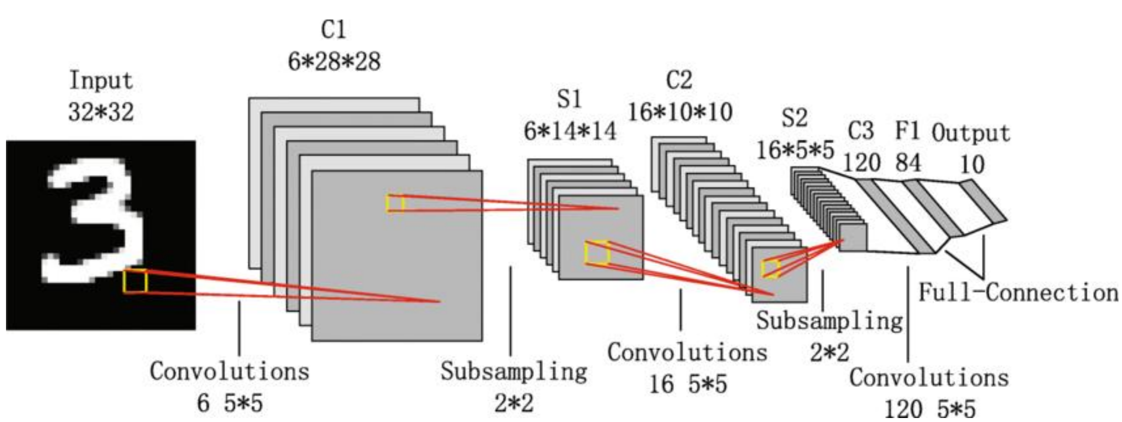


Figure 3: LeNet-5 Architecture

LeNet-5 Architecture [1] is one of the earliest architectures for image classification which was initially used for handwritten digit recognition.

In this project, a slightly modified structure is used which consists of the following layers:

- Fully connected layer consists of 14 output neurons.
- Raw logits output is replaced by softmax, as cross entropy loss in pytorch handles softmax output by default.

This model is trained from scratch having approx. 5 million parameters. The test and validation accuracy is very close which indicates better generalization as shown in figure.

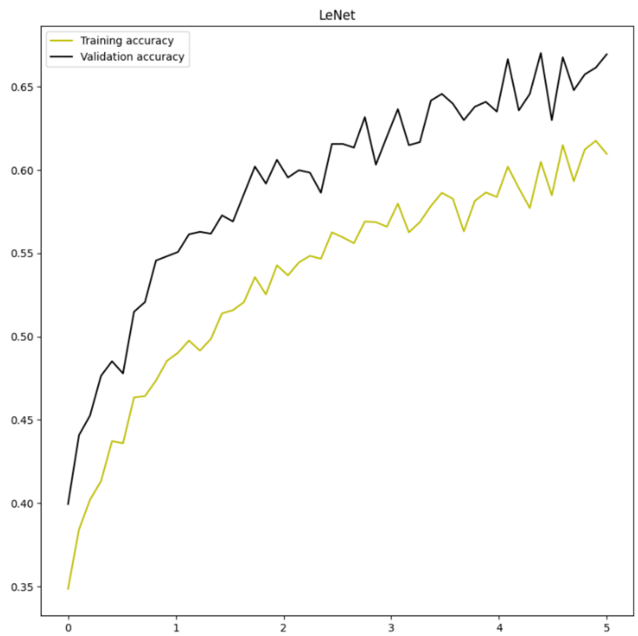
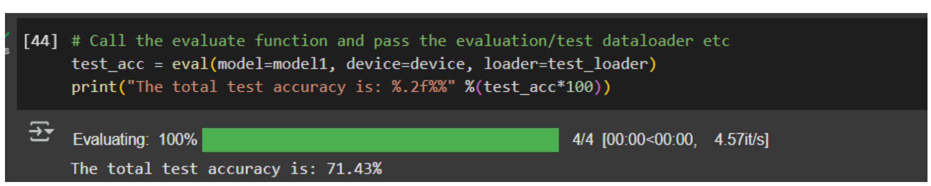


Figure 4: LeNet Training and Validation Accuracy



```
[44] # Call the evaluate function and pass the evaluation/test dataloader etc
     test_acc = eval(model=model1, device=device, loader=test_loader)
     print("The total test accuracy is: %.2f%%" %(test_acc*100))

     Evaluating: 100% 4/4 [00:00<00:00, 4.57it/s]
     The total test accuracy is: 71.43%
```

Figure 5: LeNet test accuracy

4.2. ResNet

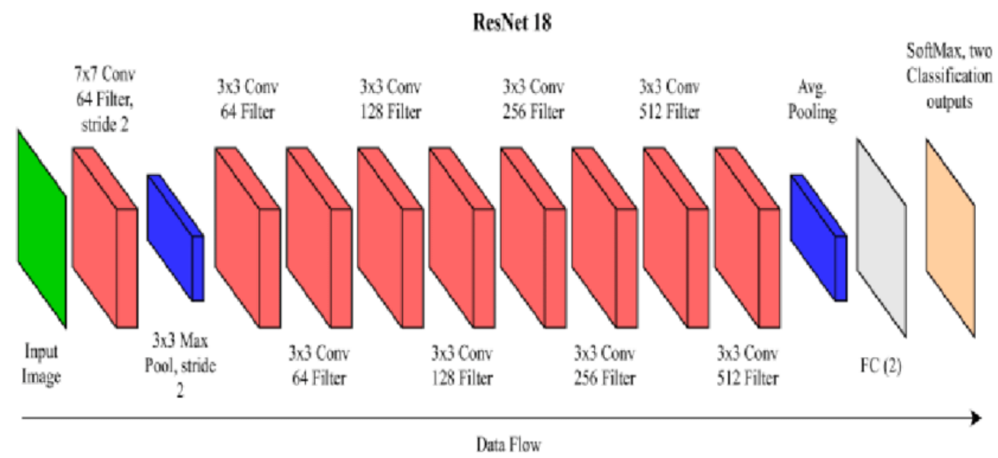


Figure 6: Resnet18 Architecture

ResNet [2] is a deep convolutional neural network introduced to solve the problem of vanishing gradients by using skip connections. These skip connections allow the model to back propagate the gradients without having to make them smaller that they vanish.

In this project, ResNet18 was used with pre-trained weights on ImageNet, applying transfer learning by freezing all the layers and replacing the fully connected layer with 14 output neurons. That means only the final layer has been trained.

The model results in total parameter count of 11 million approx. the train, validation and test accuracy is shown in the following figures.

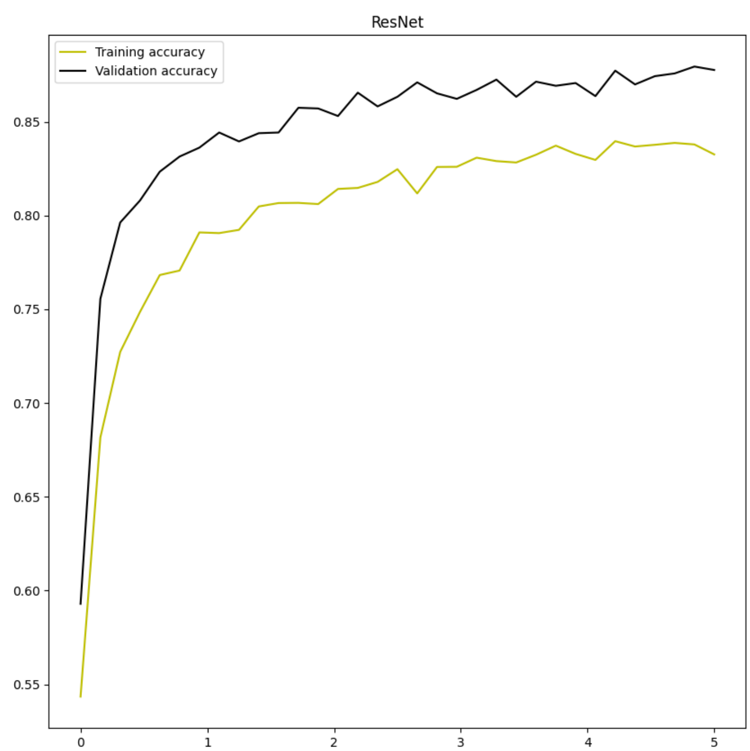


Figure 7: ResNet training and validation accuracy

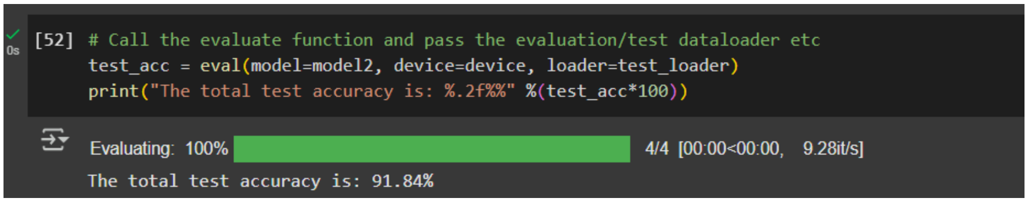


Figure 8: ResNet test accuracy

4.3. Deep Neural Network

A deep neural network [3] consists entirely of fully connected layer. Unlike, CNNs, Deep Neural Networks lack the ability to capture complex data patterns and features consisting of multiple dimensions like that of images. This fact makes them less reliable for computer vision task which is further elaborated below.

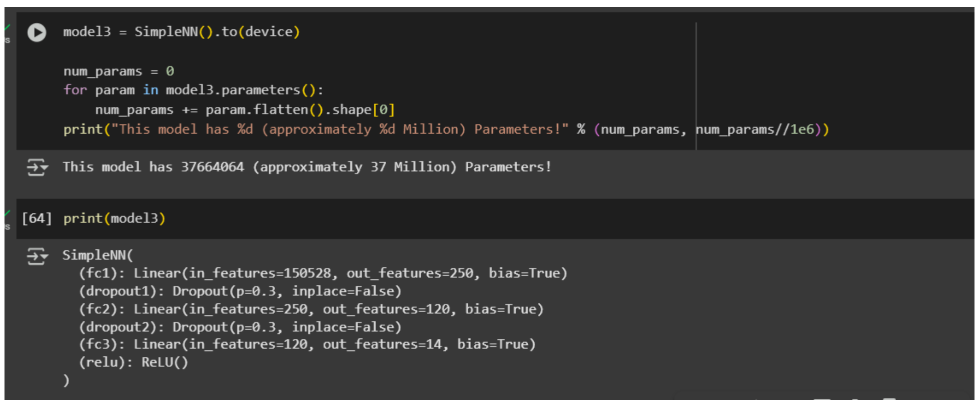


Figure 9: DNN architecture

In DNN architecture as shown in the figure, each image of size 3x224x224 is flatten to a vector of 150528 dimension. And that image is passed through further layers which are shown in the figure above. This flattening of the image vector increases the number of parameters of the neural network, which are 37 million in this case. Which are way too much as compared to the above two models.

Apart from the massive parameter count, this architecture also shows low accuracy scores for training, validation and testing as shown below:

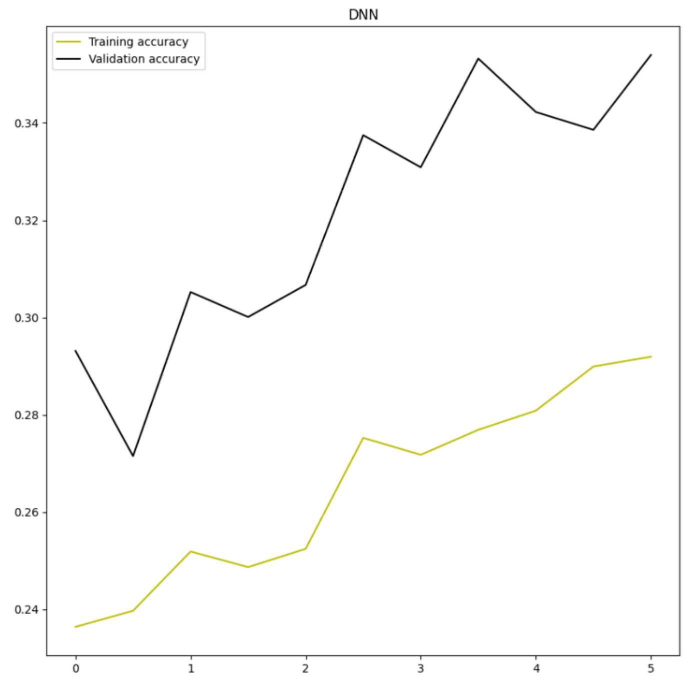


Figure 10: DNN train and validation accuracy

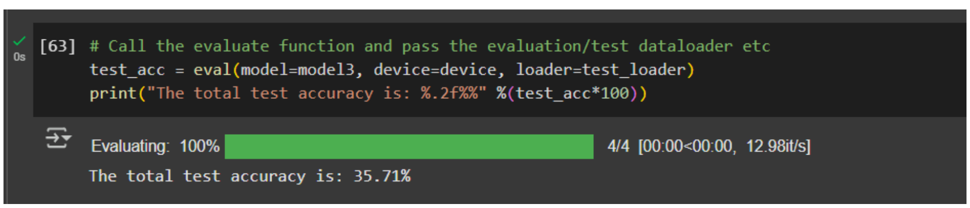


Figure 11: DNN test accuracy

5. Results

The following table shows the side-by-side comparison of the models that are trained.

Model	**Parameter**	**Train Accuracy**	**Val Accuracy**	**Test Accuracy**
LeNet-5	~5 million	61%	68%	71%
ResNet18	~11 million	81%	87%	91%
DNN	~37 million	29%	36%	35%

6. Conclusions

In this project, we implemented a flower classification using three model architectures namely CNN based LeNet-5, ResNet (using transfer learning) and a simple neural network.

These models were trained using a dataset containing approx. 13 thousand images of 14 different categories of flower species. The dataset initially consisted of train and validation folders, where validation folder consisted of only 98 images. The train folder is split into validation and the already existing validation folder is used as a training set.

Upon comparison, it was finalized that the ResNet architecture gave out the highest accuracy (validation and testing) while the having a reasonable parameter size of 11 million.

7. References

[1] K. He, J. Sun, S. Ren and X. Zhang, "Deep Residual Learning for Image Recognition," *Microsoft Research,* 2015.

[2] Y. LeCun, L. Bottou and Y. Bengio, "GradientBased Learning Applied to Document," *IEEE,* 1998.

[3] R. Uhrig, "Introduction to artificial neural networks," *IEEE,* 1995.

[4] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," 2015.

8. Appendix

Source Code

```
# -*- coding: utf-8 -*-
"""ML-CEP.ipynb

Automatically generated by Colab.

#Data Fetching
"""

!pip install opendatasets
import opendatasets as od

# od.download("https://www.kaggle.com/datasets/imsparsh/flowers-dataset")
od.download("https://www.kaggle.com/datasets/marquis03/flower-classification")

"""#Data Augmentation & Splitting"""

from torchvision import datasets, transforms
from torch.utils.data import DataLoader, random_split

train_transform = transforms.Compose([
    transforms.Resize((256,256)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(20),
    transforms.ColorJitter(brightness=0.3, contrast=0.3),
    transforms.RandomResizedCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

dataset = datasets.ImageFolder(root="/content/flower-classification/train")
```

```
len(dataset)

train_size = int(0.8*len(dataset))
val_size = len(dataset) - train_size

print(f"Train Size: {train_size}")
print(f"Validation Size: {val_size}")

train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

test_dataset = datasets.ImageFolder(root="/content/flower-classification/val")

print(f"Test Size: {len(test_dataset)}")

from torch.utils.data import Dataset

class Transform(Dataset):
  def __init__(self, subset, transform):
    self.subset = subset
    self.transform = transform

  def __len__(self):
    return len(self.subset)

  def __getitem__(self, idx):
    img, lbl = self.subset[idx]
    return self.transform(img), lbl

train_dataset = Transform(train_dataset, train_transform)
val_dataset = Transform(val_dataset, val_transform)
test_dataset = Transform(test_dataset, test_transform)

#val_dataset.dataset.transform = val_transform

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=2)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=2)

len(train_loader)

len(val_loader)

len(test_loader)

"""#Vizualization"""

import numpy as np
import matplotlib.pyplot as plt

def imageshow(image, title=None):
  img = image.numpy().transpose((1,2,0))
  mean = np.array([0.485, 0.456, 0.406])
```

```
  std  = np.array([0.229, 0.224, 0.225])
  img = std * img + mean  # unnormalize
  img = np.clip(img, 0, 1)
  plt.imshow(img)
  if title: plt.title(title)
  plt.axis('off')

image, lbls = next(iter(train_loader))
plt.figure(figsize=(12,6))
for i in range(8):
  plt.subplot(2,4,i+1)
  imageshow(image[i], title=f"{train_dataset.subset.dataset.classes[lbls[i]]}")
plt.tight_layout()
plt.show() # Added this line

class_names = train_dataset.subset.dataset.classes
print(len(class_names))

"""#Models"""

import torch
import torch.nn as nn

"""##LeNet"""

class myLeNet(nn.Module):
  def __init__(self, ip_channels):
    super().__init__()
    self.conv1 = nn.Conv2d(ip_channels, 6, kernel_size=5) #6 filters each of size 5x5 (6x5x5)
    self.conv2 = nn.Conv2d(6, 16, kernel_size=5) #16 filters each of size 5x5 (16x5x5)
    self.pooling = nn.MaxPool2d(kernel_size=2) #2x2 sampling (max pooling) with stride 2

    self.relu = nn.ReLU() #RELU activation

    # After conv1 (kernel 5, stride 1, padding 0): (224 - 5 + 1) = 220
    # After pooling1 (kernel 2, stride 2): 220 / 2 = 110
    # After conv2 (kernel 5, stride 1, padding 0): (110 - 5 + 1) = 106
    # After pooling2 (kernel 2, stride 2): 106 / 2 = 53
    # The number of output channels from conv2 is 16.
    # So the flattened size is 16 * 53 * 53
    self.linear1 = nn.Linear(16*53*53, 120) #input is the output of the last convolution layer
    self.linear2 = nn.Linear(120, 84)
    self.linear3 = nn.Linear(84, 14)

  def forward(self, x): #x: batch x 3 x 224 x 224
    conv1_out = self.conv1(x) #op: batch x 6 x 220 x 220
    conv1_out = self.relu(conv1_out) #op: batch x 6 x 220 x 220
    conv1_out = self.pooling(conv1_out) #op: batch x 6 x 110 x 110

    conv2_out = self.conv2(conv1_out) #op: batch x 16 x 106 x 106
    conv2_out = self.relu(conv2_out) #op: batch x 16 x 106 x 106
    conv2_out = self.pooling(conv2_out) #op: batch x 16 x 53 x 53
```

```
    flatten_out = conv2_out.view(conv2_out.size(0), -1) #op: batch X 16*53*53

    l1_out = self.linear1(flatten_out) #op: batch x 120
    l1_out = self.relu(l1_out)

    l2_out = self.linear2(l1_out) #op: batch X 84
    l2_out = self.relu(l2_out)

    l3_out = self.linear3(l2_out) #op: batch X 14

    return l3_out

data_iter = iter(train_loader)
imgs, lbls = next(data_iter)

import torchvision
plt.figure(figsize = (20,10))
out = torchvision.utils.make_grid(imgs, 8, normalize=True)
plt.imshow(out.numpy().transpose((1, 2, 0)))

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

model1 = myLeNet(ip_channels=imgs.shape[1]).to(device)
print(model1)

num_params = 0
for param in model1.parameters():
    num_params += param.flatten().shape[0]
print("This model has %d (approximately %d Million) Parameters!" % (num_params, num_params//1e6))

temp_out = model1(imgs.to(device))
print(temp_out.shape)

import torch.optim as optim

optimizer = optim.Adam(model1.parameters(), lr=1e-4)
loss_function = nn.CrossEntropyLoss()

from tqdm.notebook import trange, tqdm

def train(model, optimizer, loader, device, loss_fn, loss_logger):
  model.train()
  for i, (x, y) in enumerate(tqdm(loader, desc="Training")):
    fx = model(x.to(device))

    loss = loss_fn(fx, y.to(device))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    loss_logger.append(loss.item())
```

```
  return model, loss_logger

def eval(model, device, loader):
  epoch_acc = 0
  model.eval()
  with torch.no_grad():
    for i, (x,y) in enumerate(tqdm(loader, desc="Evaluating")):
      fx = model(x.to(device))
      epoch_acc += (fx.argmax(1) == y.to(device)).sum().item()

  return epoch_acc / len(loader.dataset)

training_loss_logger = []
validation_acc_logger = []
training_acc_logger = []

for epoch in trange(50, desc="Epochs"):
  model1, training_loss_logger = train(
                                model=model1,
                                optimizer=optimizer,
                                loader=train_loader,
                                device=device,
                                loss_fn=loss_function,
                                loss_logger=training_loss_logger
                              )

  train_acc = eval(model=model1, device=device, loader=train_loader)
  valid_acc = eval(model=model1, device=device, loader=val_loader)

  # Log the train and validation accuracies
  validation_acc_logger.append(valid_acc)
  training_acc_logger.append(train_acc)

plt.figure(figsize = (10,10))
train_x = np.linspace(0, 5, len(training_loss_logger))
plt.plot(train_x, training_loss_logger)
plt.title("LeNet Training Loss")

plt.figure(figsize = (10,10))
train_x = np.linspace(0, 5, len(training_acc_logger))
plt.plot(train_x, training_acc_logger, c = "y")
valid_x = np.linspace(0, 5, len(validation_acc_logger))
plt.plot(valid_x, validation_acc_logger, c = "k")

plt.title("LeNet")
plt.legend(["Training accuracy", "Validation accuracy"])

# Call the evaluate function and pass the evaluation/test dataloader etc
test_acc = eval(model=model1, device=device, loader=test_loader)
print("The total test accuracy is: %.2f%%" %(test_acc*100))

"""##ResNet (Transfer Learning)"""
```

```
import torch
import torch.nn as nn
import torchvision.models as models

# Load a pretrained model
model2 = models.resnet18(pretrained=True)

# Freeze all layers
for param in model2.parameters():
    param.requires_grad = False

# Replace the classifier (last fully connected layer)
num_classes = 14  # 14 flower categories
in_features = model2.fc.in_features
model2.fc = nn.Linear(in_features, num_classes)

num_params = 0
for param in model2.parameters():
    num_params += param.flatten().shape[0]
print("This model has %d (approximately %d Million) Parameters!" % (num_params, num_params//1e6))

print(model2)

resnet_training_loss_logger = []
resnet_validation_acc_logger = []
resnet_training_acc_logger = []

optimizer = optim.Adam(filter(lambda p: p.requires_grad, model2.parameters()), lr=1e-4)
loss_function = nn.CrossEntropyLoss()

model2.to(device)
for epoch in trange(50, desc="Epochs"):
  model2, resnet_training_loss_logger = train(
                                model=model2,
                                optimizer=optimizer,
                                loader=train_loader,
                                device=device,
                                loss_fn=loss_function,
                                loss_logger=resnet_training_loss_logger
                              )

  train_acc = eval(model=model2, device=device, loader=train_loader)
  valid_acc = eval(model=model2, device=device, loader=val_loader)

  # Log the train and validation accuracies
  resnet_validation_acc_logger.append(valid_acc)
  resnet_training_acc_logger.append(train_acc)

plt.figure(figsize = (10,10))
train_x = np.linspace(0, 5, len(resnet_training_loss_logger))
plt.plot(train_x, resnet_training_loss_logger)
```

```
plt.title("ResNet Training Loss")

plt.figure(figsize = (10,10))
train_x = np.linspace(0, 5, len(resnet_training_acc_logger))
plt.plot(train_x, resnet_training_acc_logger, c = "y")
valid_x = np.linspace(0, 5, len(resnet_validation_acc_logger))
plt.plot(valid_x, resnet_validation_acc_logger, c = "k")

plt.title("ResNet")
plt.legend(["Training accuracy", "Validation accuracy"])

# Call the evaluate function and pass the evaluation/test dataloader etc
test_acc = eval(model=model2, device=device, loader=test_loader)
print("The total test accuracy is: %.2f%%" %(test_acc*100))

"""##Deep Neural Network (DNN)"""

import torch
import torch.nn as nn

class SimpleNN(nn.Module):
    def __init__(self, input_size=3*224*224, num_classes=14):
        super(SimpleNN, self).__init__()

        self.fc1 = nn.Linear(input_size, 250)
        self.dropout1 = nn.Dropout(0.3)
        self.fc2 = nn.Linear(250, 120)
        self.dropout2 = nn.Dropout(0.3)
        self.fc3 = nn.Linear(120, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten the image
        x = self.relu(self.fc1(x))
        x = self.dropout1(x)
        x = self.relu(self.fc2(x))
        x = self.dropout2(x)
        x = self.fc3(x)
        return x

model3 = SimpleNN().to(device)

num_params = 0
for param in model3.parameters():
    num_params += param.flatten().shape[0]
print("This model has %d (approximately %d Million) Parameters!" % (num_params, num_params//1e6))

print(model3)

optimizer = optim.Adam(model3.parameters(), lr=1e-4)

DNN_training_loss_logger = []
```

```
DNN_validation_acc_logger = []
DNN_training_acc_logger = []

for epoch in trange(50, desc="Epochs"):
  model3, DNN_training_loss_logger = train(
                                model=model3,
                                optimizer=optimizer,
                                loader=train_loader,
                                device=device,
                                loss_fn=loss_function,
                                loss_logger=DNN_training_loss_logger
                               )

  train_acc = eval(model=model3, device=device, loader=train_loader)
  valid_acc = eval(model=model3, device=device, loader=val_loader)

  # Log the train and validation accuracies
  DNN_validation_acc_logger.append(valid_acc)
  DNN_training_acc_logger.append(train_acc)

plt.figure(figsize = (10,10))
train_x = np.linspace(0, 5, len(DNN_training_loss_logger))
plt.plot(train_x, DNN_training_loss_logger)
plt.title("DNN Training Loss")

plt.figure(figsize = (10,10))
train_x = np.linspace(0, 5, len(DNN_training_acc_logger))
plt.plot(train_x, DNN_training_acc_logger, c = "y")
valid_x = np.linspace(0, 5, len(DNN_validation_acc_logger))
plt.plot(valid_x, DNN_validation_acc_logger, c = "k")

plt.title("DNN")
plt.legend(["Training accuracy", "Validation accuracy"])

# Call the evaluate function and pass the evaluation/test dataloader etc
test_acc = eval(model=model3, device=device, loader=test_loader)
print("The total test accuracy is: %.2f%%" %(test_acc*100))
```