Logo

Description automatically generated

CSE485 Deep Learning

**Project Documentation**

**Submitted to:**

Prof. Dr. Mahmoud Ibrahim Khalil

Eng. Mahmoud Soheil

**Submitted by:**

Youssef George Fouad 19P9824

Kerollos Wageeh Youssef 19P3468

Mahmoud Mohamed Omar 19P5803

CESS Senior-2

Table of Contents

[1. Problem Description 4](#_Toc154344630)

[2. GETTING STARTED 4](#_Toc154344631)

[2.1. Importing Needed Libraries 4](#_Toc154344632)

[2.2. Importing Leaf Classification Dataset 5](#_Toc154344633)

[3. DATA EXPLORATION 6](#_Toc154344634)

[4. DATA PREPROCESSING 7](#_Toc154344635)

[4.1. Standardization 7](#_Toc154344636)

[4.2. Correlation Analysis 8](#_Toc154344637)

[4.2.1. Margin features analysis 8](#_Toc154344638)

[4.2.2. Shape features analysis 10](#_Toc154344639)

[4.2.3. Texture features analysis 11](#_Toc154344640)

[4.3. Label Encoding 12](#_Toc154344641)

[4.4. Train-Test Split 13](#_Toc154344642)

[5. MULTI-LAYER PERCEPTRON 14](#_Toc154344643)

[5.1. Model Architecture 14](#_Toc154344644)

[5.2. Training 17](#_Toc154344645)

[5.3. Results 18](#_Toc154344646)

[5.4. Conclusion 20](#_Toc154344647)

[6. DEEP CONVOLUTIONAL NEURAL NETWORK 21](#_Toc154344648)

[6.1. Import Images and Preprocessing 21](#_Toc154344649)

[6.2. First CNN Model Trial 23](#_Toc154344650)

[6.3. Second Architecture and Trial 24](#_Toc154344651)

[6.4. Transfer Learning with a Pre-Trained ResNet-18 25](#_Toc154344652)

[6.4.1. Additional preprocessing steps 25](#_Toc154344653)

[6.4.2. Adapting the pre-trained model 25](#_Toc154344654)

[6.4.3. Model architecture using torchViz library 26](#_Toc154344655)

[6.4.4. Training 27](#_Toc154344656)

[6.4.5. Results 28](#_Toc154344657)

[6.5. Comparing CNN Trials 29](#_Toc154344658)

[7. A HYBRID APPROACH 30](#_Toc154344659)

[7.1. Model Architecture 30](#_Toc154344660)

[7.2. Training 31](#_Toc154344661)

[7.3. Results 32](#_Toc154344662)

[7.4. Conclusion 33](#_Toc154344663)

[8. FINAL WORDS 34](#_Toc154344664)

[9. SOURCE CODE and NOTEBOOK 35](#_Toc154344665)

# Problem Description

There are estimated to be nearly half a million species of plant in the world. Classification of species has been historically problematic and often results in duplicate identifications.

The objective of this project is to build a neural network that classifies 99 species of plants.

We will try 3 approaches:

1. **Multi-layer perceptron**, given some numerical extracted features.
2. **Deep Convolutional Neural Network**, using the given images.
3. **A Hybrid Approach**, utilizing both images and numerical features.

# GETTING STARTED

## Importing Needed Libraries

Before building our deep learning model, we start by importing needed python libraries that help us process the images, build, train, test our model accuracy, and visualize the results.

# deep learning libraries (pytorch)

import torch

import torch.nn as nn

import torch.optim as optim

import torch.nn.functional as F

import albumentations as A

from albumentations.pytorch import ToTensorV2

from torchvision import datasets, transforms, models

# importing dataset

import os

from PIL import Image

import cv2

import pandas as pd

# data processing

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

import numpy as np

# visualization

import matplotlib.pyplot as plt

from tqdm import tqdm

from torch.utils.tensorboard import SummaryWriter

from ann\_visualizer.visualize import ann\_viz

from torchviz import make\_dot

from torchsummaryX import summary

import time

## Importing Leaf Classification Dataset

The dataset consists of 990 **binary** images, labels. In addition, pre-extracted features provide a numerical representation of leaves different properties, which are margins, shapes, texture.

* There are 64 marginal features, 64 shape features, 64 texture features.

The **binary** images are in **JPG** format which are easy to read in shape.

* View 100 sample images of the dataset.

image\_filenames = sorted(os.listdir("Dataset/images/"))

fig = plt.figure(figsize=(10, 10))

gs = gridspec.GridSpec(10, 10, wspace=0.05, hspace=0.05)

for i, image\_filename in enumerate(image\_filenames[:100]):

image\_path = os.path.join(image\_folder, image\_filename)

img = Image.open(image\_path).convert('1') # '1' for black/white

ax = plt.subplot(gs[i])

plt.imshow(img)

A black and white image of different leaves

Description automatically generated

# DATA EXPLORATION

In this section we will explore the given numerical features of “train.csv” file.

* Import data in pandas DataFrame and check its shape and features type.

train\_csv = pd.read\_csv('Dataset/train.csv')

print(f"There are {len(train\_csv.species.unique())} species in the train set.")

print(f"There are {len(train\_csv.id.unique())} images in the train set.")

print(f"There are {train\_csv.shape[1]} columns in the train set.")

print(f"There are {train\_csv.isnull().sum()} missing values in the train set.")

* There are 990 images in the train dataset, which we will work on, divided among 99 classes/ labels/ species.

A black background with white text

Description automatically generated

* Each image is given an ID column, a label (species), 64 margin features, 64 shape features, and 64 texture features. **No missing values at all!**

print(train\_csv.dtypes.value\_counts())

print(f"{train\_csv.select\_dtypes(include=['int64']).columns[0]} is the only int column in the train set.")

print(f"{train\_csv.select\_dtypes(include=['object']).columns[0]} is the only object column in the train set.")

print(f"There are {train\_csv.filter(regex='margin').shape[1]} MARGIN features.")

print(f"There are {train\_csv.filter(regex='shape').shape[1]} SHAPE features.")

print(f"There are {train\_csv.filter(regex='texture').shape[1]} TEXTURE features.")

A screen shot of a computer

Description automatically generated

# DATA PREPROCESSING

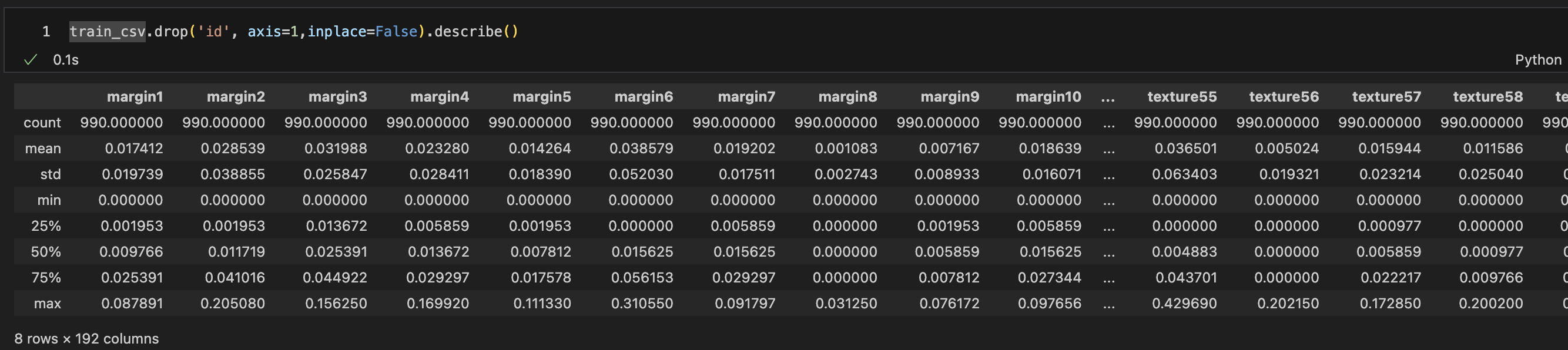
## Standardization

Standardizing numerical features ensures all of them are on a certain scale having a zero mean and standard deviation of one, which helps in better understanding of the relationships between these features in addition to helping us to reach better model training and performance results.

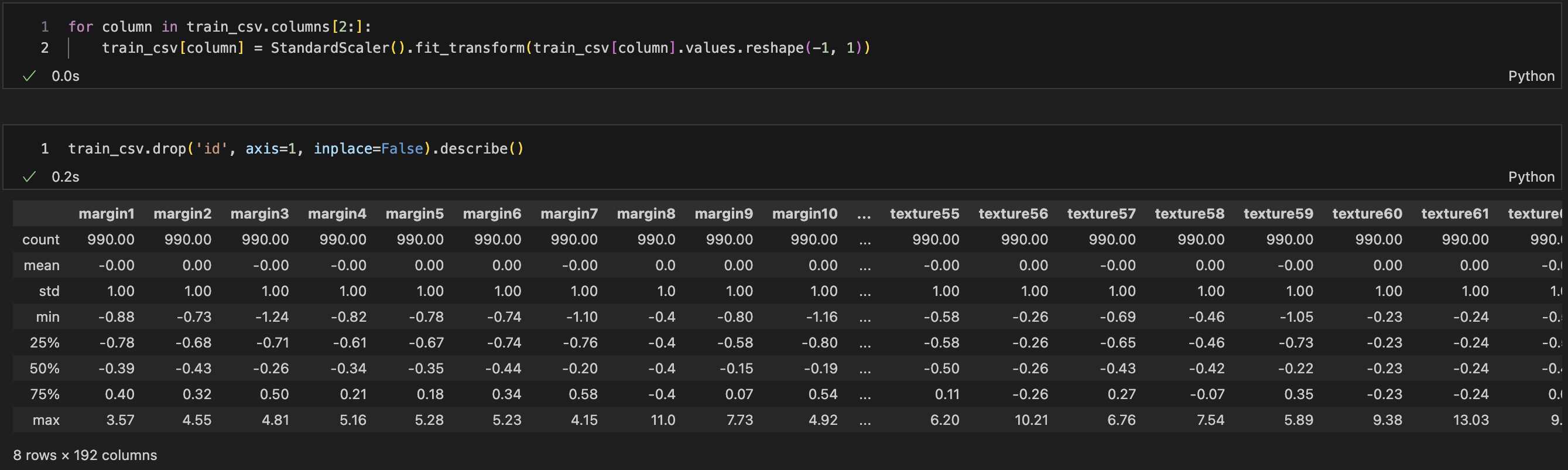
where:

As we can see in the table below, the mean and standard deviation for each feature varies; therefore, standardization helps us flatten the mean and standard deviation to 0 and 1.

**Before standardization:**



**After standardization:**



## Correlation Analysis

Since we have 192 numerical features, we would undergo Pearson correlation analysis between each category of features alone (margin, shape, texture).

We undergo correlation analysis between features of data to quantify relationships between these features and try to find patterns, identify trends, or exclude redundant features.

After we calculate correlation between each feature and the other features, we end up with a (n x n) matrix of numbers ranging from -1 to 1 that resembles how strong each feature correlates with another one.

We will consider only those who are > |0.8|, which are >0.8 or <-0.8, and we will filter them out in a separate chart as shown below in each of the 3 features categories.

### Margin features analysis

fig, ax = plt.subplots(1,2,figsize=(15,5))

margin\_corr = train\_csv.filter(regex='margin').corr()

sns.heatmap(margin\_corr,vmin=-1, vmax=1, center=0, cmap='viridis', mask=np.triu(np.ones\_like(margin\_corr,dtype=bool)), linewidths=0.5, ax=ax[0])

sns.heatmap(margin\_corr,vmin=-1, vmax=1, center=0, cmap='viridis', mask= np.triu(np.ones\_like(margin\_corr, dtype=bool))|(np.abs(margin\_corr) <= 0.8) , linewidths=0.5, ax=ax[1])

plt.title('Correlation between margin features')

plt.show()

print(f"There are {train\_csv.filter(regex='margin').shape[1]} MARGIN features in the train set.")

for i in range(len(margin\_corr)):

for j in range(i+1,len(margin\_corr)):

if i != j and abs(margin\_corr.iloc[i,j]) >= 0.8:

print(f"{margin\_corr.columns[i]} and {margin\_corr.columns[j]} have high correlation = {round(margin\_corr.iloc[i,j],2)}")

The results show how only few of the 64 marginal features are highly correlated, i.e. pass our threshold > |0.8|

A graph of a number of colored dots

Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated

### Shape features analysis

fig, ax = plt.subplots(1,2,figsize=(15,5))

shape\_corr = train\_csv.filter(regex='shape').corr()

sns.heatmap(shape\_corr,vmin=-1, vmax=1, center=0, cmap='viridis', mask=np.triu(np.ones\_like(shape\_corr,dtype=bool)), linewidths=0.5, ax=ax[0])

sns.heatmap(shape\_corr,vmin=-1, vmax=1, center=0, cmap='viridis', mask= np.triu(np.ones\_like(shape\_corr, dtype=bool))|(np.abs(shape\_corr) <= 0.8) , linewidths=0.5, ax=ax[1])

plt.title('Correlation between shape features')

plt.show()

print(f"There are {train\_csv.filter(regex='shape').shape[1]} SHAPE features in the train set.")

for i in range(len(shape\_corr)):

for j in range(i+1,len(shape\_corr)):

if i != j and abs(shape\_corr.iloc[i,j]) >= 0.8:

print(f"{shape\_corr.columns[i]} and {shape\_corr.columns[j]} have high correlation = {round(shape\_corr.iloc[i,j],2)}")

The results show how a lot of the 64 shape features are highly correlated with each other, i.e. pass our threshold > |0.8|

A screenshot of a graph

Description automatically generated

### Texture features analysis

fig, ax = plt.subplots(1,2,figsize=(15,5))

shape\_corr = train\_csv.filter(regex='texture').corr()

sns.heatmap(shape\_corr,vmin=-1, vmax=1, center=0, cmap='viridis', mask=np.triu(np.ones\_like(shape\_corr,dtype=bool)), linewidths=0.5, ax=ax[0])

sns.heatmap(shape\_corr,vmin=-1, vmax=1, center=0, cmap='viridis', mask= np.triu(np.ones\_like(shape\_corr, dtype=bool))|(np.abs(shape\_corr) <= 0.8) , linewidths=0.5, ax=ax[1])

plt.title('Correlation between texture features')

plt.show()

print(f"There are {train\_csv.filter(regex='texture').shape[1]} TEXTURE features in the train set.")

for i in range(len(shape\_corr)):

for j in range(i+1,len(shape\_corr)):

if i != j and abs(shape\_corr.iloc[i,j]) >= 0.8:

print(f"{shape\_corr.columns[i]} and {shape\_corr.columns[j]} have high correlation = {round(shape\_corr.iloc[i,j],2)}")

A comparison of a graph

Description automatically generated with medium confidence

A black background with white text

Description automatically generated

The results show how only 1 pair of the 64 texture features are highly correlated, i.e. pass our threshold > |0.8|

## Label Encoding

As we can see in the screenshot below, the label is not encoded but rather a string of the leaf class, which cannot be used in neural networks training process.

A screenshot of a computer

Description automatically generated

Therefore, encoding the species column using a Label Encoder will replace the text values with integer values between 0-98 replacing the 99 different classes of the leaves’ species.

train\_data = pd.DataFrame()

train\_data['label'] = LabelEncoder().fit\_transform(train\_csv['species'])

train\_data = pd.concat([train\_data,train\_csv.drop(['id','species'],axis=1)],axis=1)

print(f"There are {train\_csv.label.nunique()} unique labels in the train set.")

print(f"There are {train\_csv.species.nunique()} unique species in the train set.")

train\_data.head()

A black and white screen with numbers

Description automatically generated

## Train-Test Split

We use **stratified (stratify = y)** train-test split to ensure the distribution of classes is the same among the train and test samples as the original dataset.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, labels, test\_size=0.2, random\_state=43, stratify=y)

**Original Dataset:**

Total samples = 990,

equally divided between the 99 classes: **10 samples each**.

**Train Dataset:**

Total samples = 990\*0.8 = 792,

equally divided between the 99 classes: **8 samples each**.

**Train Dataset:**

Total samples = 990\*0.2 = 198,

equally divided between the 99 classes: **2 samples each**.

A screen shot of a computer program

Description automatically generated

# MULTI-LAYER PERCEPTRON

Our first classification approach would be utilizing the given numerical features to build a neural network (MLP) model.

## Model Architecture

A double hidden layer neural network with:

* 192 input layer nodes (192 input features).
* 256 nodes in the first hidden layer activated with ReLU activation function.
* 256 nodes in the second hidden layer activated with ReLU activation function.
* 99 output layer nodes (number of classes).

class MLP(nn.Module):

def \_\_init\_\_(self, input\_size=192, hidden\_size=256, num\_classes=99):

super(MLP, self).\_\_init\_\_()

self.fc1 = nn.Linear(input\_size, hidden\_size)

self.relu1 = nn.ReLU()

self.fc2 = nn.Linear(hidden\_size, hidden\_size)

self.relu2 = nn.ReLU()

self.fc3 = nn.Linear(hidden\_size, num\_classes)

def forward(self, x):

x = self.relu1(self.fc1(x))

x = self.relu2(self.fc2(x))

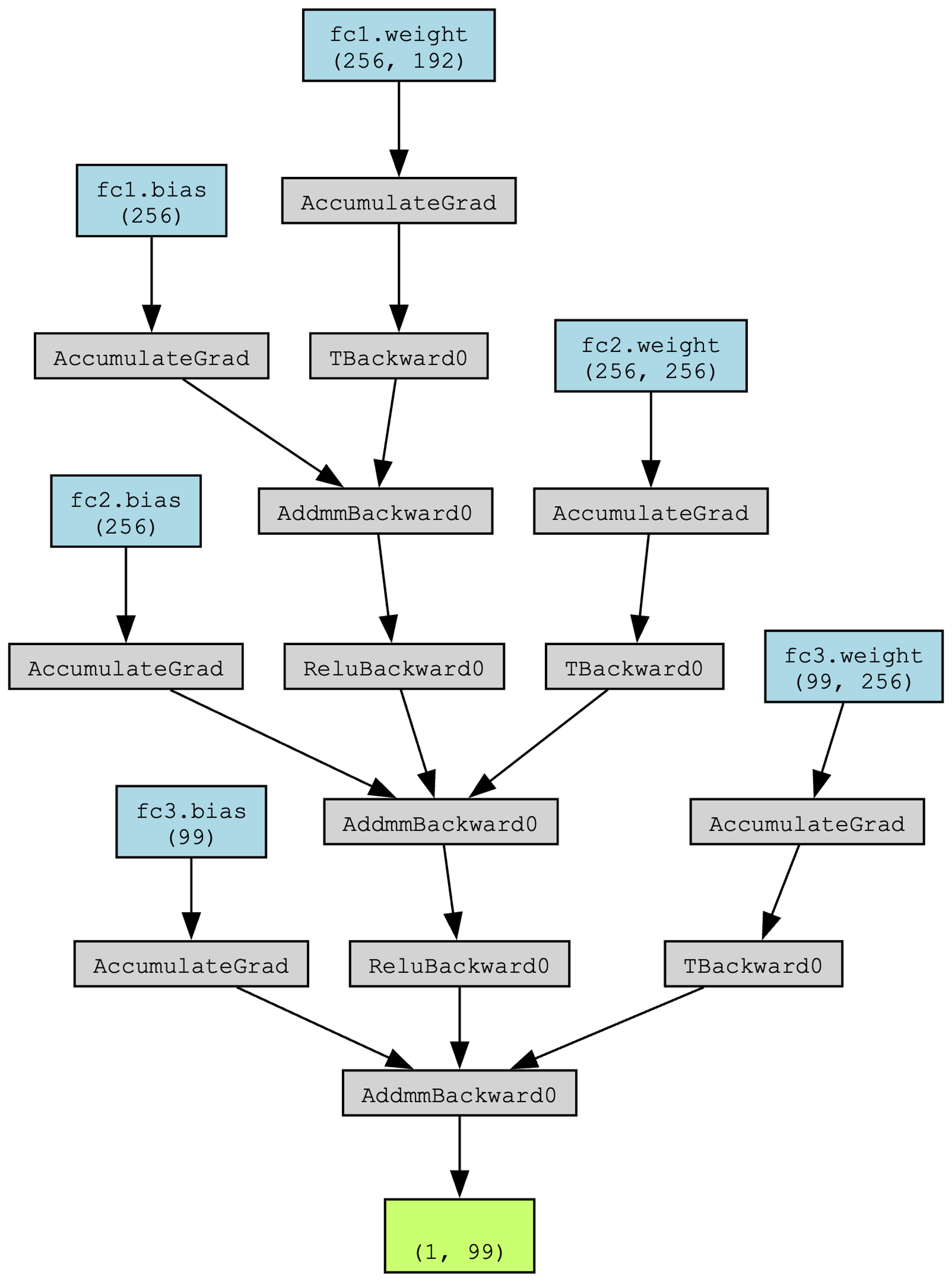
x = self.fc3(x)

return x

MLP architecture visualized using torchviz library:

model = MLP()

make\_dot(model(torch.randn(1, 192)), params=dict(model.named\_parameters()))



MLP architecture visualized using tensorboard library:

writer = SummaryWriter('runs/mlp')

writer.add\_graph(model, torch.randn(1, 192))

writer.close()

A screenshot of a computer

Description automatically generated

## Training

Training hyperparameters:

* Batch Size = tried 8, 16,3 2, 64
* Number of epochs = 1000
* Optimizer = Adam
* Loss Criteria = Cross Entropy
* Learning Rate = tried 0.001, 0.0001

batch\_size = 64

num\_epochs = 1000

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.0001)

Loading datasets in a data loader:

X\_train\_tensor = torch.tensor(X\_train.values, dtype=torch.float32)

y\_train\_tensor = torch.tensor(y\_train, dtype=torch.int64)

# Create a train DataLoader

train\_dataset = TensorDataset(X\_train\_tensor, y\_train\_tensor)

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

X\_test\_tensor = torch.tensor(X\_test.values, dtype=torch.float32)

y\_test\_tensor = torch.tensor(y\_test, dtype=torch.int64)

# Create a test DataLoader

test\_dataset = TensorDataset(X\_test\_tensor, y\_test\_tensor)

test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=True)

**Training loop:**

Throughout each epoch, the model trains on all batches and validates against the test set.

To monitor the model performance throughout the training process, we calculate both average training loss and validation loss.

for epoch in range(num\_epochs):

model.train()

running\_loss = 0.0

for x, labels in tqdm(train\_loader, desc=f"Epoch {epoch+1}/{num\_epochs}", unit="batch"):

optimizer.zero\_grad()

outputs = model(x)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

train\_loss = running\_loss / len(train\_dataset)

# Validation loop

model.eval()

correct\_predictions = 0

total\_samples = 0

with torch.no\_grad():

for x\_val, labels in test\_loader:

outputs = model(x\_val)

\_, predicted = torch.max(outputs, 1)

total\_samples += labels.size(0)

correct\_predictions += (predicted == labels).sum().item()

accuracy = correct\_predictions / total\_samples

val\_loss = 1-accuracy

val\_losses.append(val\_loss)

if val\_loss < best\_val\_loss:

best\_val\_loss = val\_loss

best\_epoch = epoch

best\_model = model

writer.add\_scalar('Training Loss', train\_loss, epoch)

writer.add\_scalar('Validation Loss', val\_loss, epoch)

writer.close()

## Results

print(f"Training accuracy = {round(accuracy\_score(y\_train\_tensor, torch.max(best\_model(X\_train\_tensor), 1)[1]),4)\*100}%")

print(f"Validation accuracy = {round(accuracy\_score(y\_test\_tensor, predictions),3)\*100}%")

print(f"Best Validation Epoch = {best\_epoch}")

predictions = best\_model(X\_test\_tensor)

predictions = torch.max(predictions, 1)[1]

sns.heatmap(confusion\_matrix(y\_test\_tensor, predictions), annot=False, fmt='g', cmap='summer')

plt.title('MLP Confusion Matrix')

plt.show()

A black background with white text

Description automatically generated

The results states that the model succeeded in memorizing the training data with a training accuracy 100% and succeeded in generalizing to the 99 classes with a 99% validation accuracy after 41 epochs only.

A graph with a line

Description automatically generated

A graph with a line

Description automatically generated

The confusion matrix shows how 196 test samples were classified correctly, and only 2 samples only were misclassified.

A green and yellow graph

Description automatically generated

## Conclusion

The first classification attempt is very successful as the model is very efficient, fast to train and test, while maintaining superior accuracy results. The given extracted features precisely describe and identify the leaves.

# DEEP CONVOLUTIONAL NEURAL NETWORK

## Import Images and Preprocessing

As we previously shown, labels need to be encoded from strings to integers to be used in our model, and we will also import the binary images this time.

trainSet = pd.read\_csv('Dataset/train.csv')

classes = trainSet[['id','species']].copy()

classes['id'] = classes['id'].astype(str)

classes['label'] = LabelEncoder().fit\_transform(classes['species'])

image\_folder = 'Dataset/images/'

imgs = []

labels = []

for i in sorted(os.listdir(image\_folder)):

id = i.split('.')[0]

if id in classes['id'].values:

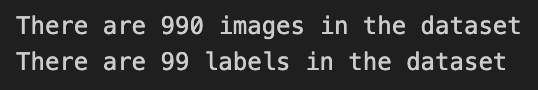
labels.append(classes[classes['id'] == id]['label'].values[0])

image = Image.open(os.path.join(image\_folder, i)).convert('1')

imgs.append(image)

print(f"There are {len(imgs)} images in the dataset")

print(f"There are {len(np.unique(labels))} labels in the dataset")



We will also undergo stratified train-test split.

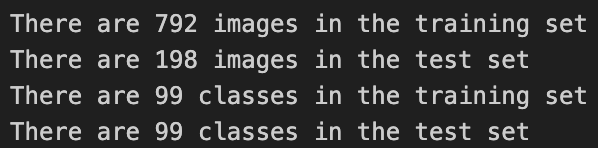
X\_train, X\_test, y\_train, y\_test = train\_test\_split(imgs, labels, test\_size=0.2, random\_state=42, stratify = labels)

print(f"There are {len(X\_train)} images in the training set")

print(f"There are {len(X\_test)} images in the test set")

print(f"There are {len(np.unique(y\_train))} classes in the training set")

print(f"There are {len(np.unique(y\_test))} classes in the test set")



We need to build a custom dataset class first to apply transformations on each image and match it to its correct label before introducing it to any model.

**This custom dataset applies the following transformations:**

* Resize each picture from its original size to 32\*32.
* Convert its format into a tensor.

class CustomDataSet(torch.utils.data.Dataset):

def \_\_init\_\_(self, images, labels, transform=None):

self.images = images

self.labels = labels

self.transform = transform

def \_\_len\_\_(self):

return len(self.images)

def \_\_getitem\_\_(self, index):

image = self.transform(image=np.array(self.images[index],dtype=np.float32))['image']

label = torch.tensor(self.labels[index], dtype=torch.long)

return image, label

transform = A.Compose([

A.Resize(32, 32),

ToTensorV2()

])

trainDataSet = CustomDataSet(images=X\_train, labels=y\_train, transform=transform)

testDataSet = CustomDataSet(images=X\_test, labels=y\_test, transform=transform)

## First CNN Model Trial

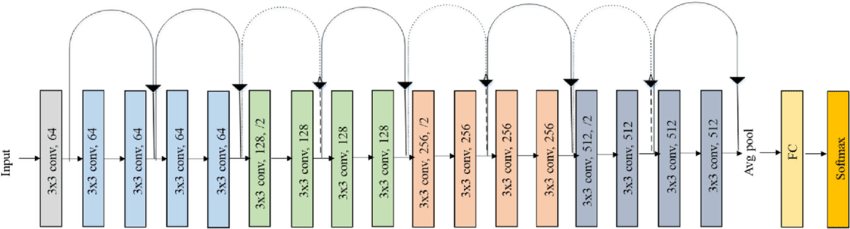
Our first attempt working on the images using a deep convolutional model.

## Second Architecture and Trial

## Transfer Learning with a Pre-Trained ResNet-18

**Transfer Learning** is used when the available training set size is not as large as it should be, so fine-tuning a pretrained model with the dataset in hand for improved performance.

**Resnet-18** is a well-known model that has been trained on a large dataset (ImageNet) and might help us with our classification task.



### Additional preprocessing steps

* Resnet-18 model input must be a 224\*224 image.

transform = A.Compose([

A.Resize(224, 224),

ToTensorV2()

])

### Adapting the pre-trained model

1. Resnet-18 model input had 3 channels (RGB) but our dataset has only 1 channel (binary) so we adapted its first convolutional layer to accept a single input channel.

model = models.resnet18(pretrained=True)

model.conv1 = nn.Conv2d(1, 64, kernel\_size=7, stride=2, padding=3, bias=False)

1. Since Resnet-18 model was originally trained for ImageNet dataset, its last fully connected layer has 1000 nodes to classify 1000 classes. We will replace the last layer with 99 only to match the number of classes in the Leaf Classification dataset.

model.fc = nn.Linear(model.fc.in\_features, 99)

### Model architecture using torchViz library

A sequential layer consists of basic blocks 0, 1 and each basic block groups multiple 2D convolutional later, batch normalization, ReLU activation function as shown.

model = models.resnet18(pretrained=True)

model.conv1 = nn.Conv2d(1, 64, kernel\_size=7, stride=2, padding=3, bias=False)

model.fc = nn.Linear(model.fc.in\_features, 99)

writer = SummaryWriter('runs/resnet18Architecture'+time.strftime("%Y%m%d-%H%M%S"))

writer.add\_graph(model, torch.zeros([1,1,224,224]))

writer.close()

A screenshot of a diagram

Description automatically generatedA screen shot of a diagram

Description automatically generated

### Training

Training hyperparameters:

* Batch Size = 64
* Number of epochs = 100
* Optimizer = Adam
* Loss Criteria = Cross Entropy
* Learning Rate = 0.001

Same training and validation procedure as the previous models.

batch\_size = 64

trainDataLoader = torch.utils.data.DataLoader(trainDataSet, batch\_size=batch\_size)

testDataLoader = torch.utils.data.DataLoader(testDataSet, batch\_size=batch\_size)

min\_loss\_epoch = 0

min\_loss\_value = -1

best\_model\_weights\_paths = {}

best\_val\_loss = float('inf') # Initialize with a large value

best\_epoch = -1

best\_model = None

train\_losses = []

val\_losses = []

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

num\_epochs = 100

writer = SummaryWriter('runs/resnet18/'+time.strftime("%Y%m%d-%H%M%S"))

for epoch in range(num\_epochs):

model.train()

running\_loss = 0.0

for images, labels in tqdm(trainDataLoader, desc=f"Epoch {epoch+1}/{num\_epochs}", unit="batch"):

optimizer.zero\_grad()

outputs = model(images)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

train\_loss = running\_loss / len(trainDataSet)

train\_losses.append(train\_loss)

# Validation loop

model.eval()

correct\_predictions = 0

total\_samples = 0

with torch.no\_grad():

for images, labels in testDataLoader:

outputs = model(images)

\_, predicted = torch.max(outputs, 1)

total\_samples += labels.size(0)

correct\_predictions += (predicted == labels).sum().item()

accuracy = correct\_predictions / total\_samples

val\_loss = 1-accuracy

val\_losses.append(val\_loss)

if val\_loss < best\_val\_loss:

best\_val\_loss = val\_loss

best\_epoch = epoch

best\_model = model

writer.add\_scalar('Training Loss', train\_loss, epoch)

writer.add\_scalar('Validation Loss', val\_loss, epoch)

### Results

print(f"Training accuracy = {round(accuracy\_score(y\_train\_tensor, torch.max(best\_model(X\_train\_tensor), 1)[1]),4)\*100}%")

print(f"Validation accuracy = {round(accuracy\_score(y\_test\_tensor, predictions),3)\*100}%")

print(f"Best Validation Epoch = {best\_epoch}")

The results states that the model succeeded in memorizing the training data with a training accuracy 100% after 20 epochs and succeeded in generalizing to the 99 classes with an 89% validation accuracy after 25 epochs.

## Comparing CNN Trials

# A HYBRID APPROACH

In the previous two sections, we either used the numerical features alone or the spatial features (images) alone but now we are combining them both in one model.

The numerical features are concatenated to the last flat layer in our model that precedes the 99 nodes layer, so that the model utilizes both readily extracted numerical features and the features that the deep neural network propose.

## Model Architecture

The first part is a regular CNN model with batch normalization, ReLU activation, dropout layers. In the last fully connected layer, the nodes are concatenated with the given features.

## Training

Training hyperparameters:

* Batch Size = 64
* Number of epochs = 100
* Optimizer = Adam
* Loss Criteria = Cross Entropy
* Learning Rate = 0.001

## Results

## Conclusion

# FINAL WORDS

# SOURCE CODE and NOTEBOOK

You can access the project source code on GitHub. Press [Here.](https://github.com/youssefg7/CNNLeafClassification)