EE 470

PROJECT TITLE

Group 4

By

Okan Bulgur
Berke Berkay Tekçe

Faculty of Engineering
Department of Electrical and Electronics Engineering
Istanbul, 2024
Yeditepe University

ABSTRACT

Retail Product Recognition Challenge

In this project, a Convolution Neural Network (CNN) model was developed to recognize retail drinks through images. While developing this model, a 6969 dimensional data set consisting of 48 classes was utilized and 4 Convolution layers and 4 fully connected layers were used in the model.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

- **CNN**: Convolutional Neural Network
- **RGB**: Red, Green, Blue
- **IMG_SIZE**: Image Size
- **BATCH_SIZE**: Number of samples processed in one batch
- **LR**: Learning Rate
- **GPU**: Graphics Processing Unit
- **CPU**: Central Processing Unit
- **ML**: Machine Learning
- **DL**: Deep Learning
- **ReLU**: Rectified Linear Unit (Activation Function)
- **FC**: Fully Connected Layer
- **Acc**: Accuracy (Evaluation Metric)
- **ResNet**: Residual Neural Network (Architecture)

1. INTRODUCTION

In this project, as a group of students, we aimed to implement a deep learning algorithm for the given retail product identification problem. We approached the given problem with several methods and strategies. In the proposed work part, we mentioned these methods and strategies in detail.

2. LITERATURE REVIEW

- **Brownlee, J. (2019)** discussed preparing categorical data for deep learning in Python, providing insights into efficient data preprocessing techniques for model training.
- **Khan, M. A. (2023)** outlined steps to build a Softmax Classifier for images using PyTorch, emphasizing techniques for multi-class classification tasks.
- **PyTorch Team (2023)** provided detailed documentation on fine-tuning pre-trained models for image classification, highlighting the benefits of transfer learning in deep learning workflows.
- **Brownlee, J. (2020)** further elaborated on preparing categorical data, focusing on advanced techniques for deep learning projects in Python.
- **Jung, A. (2020)** introduced the ImgAug library for image augmentation, offering practical methods to improve model robustness and generalization through data augmentation.
- **Discipline of ICT, School of TED, University of Tasmania (2020)** explored deep learning challenges and techniques specific to retail product recognition, providing valuable insights into domain-specific issues.
- **Journal of Retailing (2017)** presented perspectives on the future of retailing, shedding light on the potential of AI and deep learning to transform the industry.

3. PROPOSED WORK

3.1. PROBLEM DEFINITION

This project will seek to develop a deep learning-based algorithm for the recognition of retail products. The work will involve the classification of images of retail products into distinct categories, with challenges ranging from diverse backgrounds and shadows to other variations within the dataset. The model would be evaluated on its accuracy against unseen test images, focusing on the generalization capability. Python will be used for implementation, and the focus is on achieving the highest classification accuracy to outperform other algorithms in a competitive settings.

3.2. PROPOSED METHODOLOGY

The proposed methodology for this project focuses on building a robust deep learning model to classify images into multiple categories effectively. The approach leverages convolutional neural networks (CNNs) with advanced data preprocessing, augmentation techniques, and customized training strategies to achieve high accuracy and generalization.

3.2.1. DATA PREPARATION

1. Dataset Splitting

The dataset is split into **train (70%)**, **validation (15%)**, and **test (15%)** sets using PyTorch's 'random_split' function to ensure proper evaluation of the model.

2. Image Processing and Augmentation

Firstly, input images are resized to a uniform size of **128x128x3 pixels** for consistency. Then to improve the model's generalization ability and reduce overfitting, the following data

augmentation techniques are applied: Random Horizontal Flip, Random Rotation, Random Resized Crop, Normalization.

3. Class Balancing

Class imbalance in the dataset is addressed by calculating **class weights** inversely proportional to the class frequencies. These weights are used in the CrossEntropyLoss function to ensure fair learning across all classes.

3.2.2. MODEL ARCHITECTURE

The 'NeuralNet' model is a Convolutional Neural Network (CNN) designed to provide accurate classification for retail product identification. The architecture consists of multiple convolutional, pooling and fully connected layers, as detailed below:

- **Convolutional and Pooling Layers**

The model contains four convolutional blocks, each followed by a 'ReLU' activation function and a 'MaxPooling' layer to progressively reduce the spatial dimensions while retaining essential features:

<u>Convolutional Block 1:</u>

Input Channels: 3 (RGB image), Output Channels: 32

Convolution: 3x3 kernel, stride 1, padding 1 → Output: 128x128x32

MaxPooling: 2x2 kernel, stride 2 → Output: 64x64x32

<u>Convolutional Block 2:</u>

Input Channels: 64, Output Channels: 64

Convolution: 3x3 kernel, stride 1, padding 1 → Output: 64x64x64

MaxPooling: 2x2 kernel, stride 2 → Output: 32x32x64

Convolutional Block 3:

Input Channels: 128, Output Channels: 128

Convolution: 3x3 kernel, stride 1, padding 1 → Output: 32x32x128

MaxPooling: 2x2 kernel, stride 2 → Output: 16x16x128

Convolutional Block 4:

Input Channels: 256, Output Channels: 256

Convolution: 3x3 kernel, stride 1, padding 1 → Output: 16x16x256

MaxPooling: 2x2 kernel, stride 2 → Output: 8x8x256

- **Fully Connected Layers**

After the convolutional and pooling blocks, the feature maps are flattened into a 1D vector and passed through a series of fully connected layers to perform classification:

Fully Connected Layer 1 (FC1):

Input Size: 8×8×256, Output Size: 512
Activation: ReLU

Fully Connected Layer 2 (FC2):

Input Size: 512, Output Size: 256
Activation: ReLU

Fully Connected Layer 3 (FC3):

Input Size: 256, Output Size: 128
Activation: ReLU

Output Layer (FC4):

Input Size: 128
Output Size: Number of classes (48)

- **Regularization**

To reduce overfitting and improve generalization, Dropout layers with a probability of 0.5 are added after each fully connected layer. This helps prevent the model from relying too heavily on specific neurons during training.

3.2.3. TRAINING STRATEGY

- **Loss Function:**

The weighted 'CrossEntropyLoss' function is used to handle class imbalance and ensure the model learns effectively across all classes.

- **Optimizer:**

The Adam optimizer is employed with a learning rate of 0.001 and a weight decay of 1e-4 to optimize the model parameters.

- **Training Process:**

The model is trained for 30 epochs using a batch size of 32.

  - At each epoch:

1. Forward propagation calculates predictions.
2. Backward propagation computes gradients using the loss function.
3. Model parameters are updated with the optimizer.
4. Training accuracy and loss are recorded for analysis.

- **Validation:**

After each epoch, the model is evaluated on the validation set to monitor the validation loss and accuracy.

3.2.4. EVALUATION AND VISUALIZATION

- **Model Evaluation:**

The model's performance is evaluated on the validation set using accuracy as the primary metric.

- **Confusion Matrix:**

A confusion matrix is generated to analyse the classification performance across all product classes. This helps in identifying misclassifications and class-specific weaknesses.

- **Training and Validation Loss Visualization:**

The training and validation losses are plotted across epochs to observe convergence behaviour and detect overfitting.

4. EXPERIMENTAL RESULTS

DATASET INFORMATION

- 6969 images
- 48 classes

CONFUSION MATRIX

- The confusion matrix shows the model's performance by comparing actual and predicted classes. Diagonal values indicate correct predictions, while off-diagonal values represent misclassifications. It helps identify which classes are well-predicted and which are often confused, providing insights into areas for improvement. Our confusion matrix is in Figure 4.1.

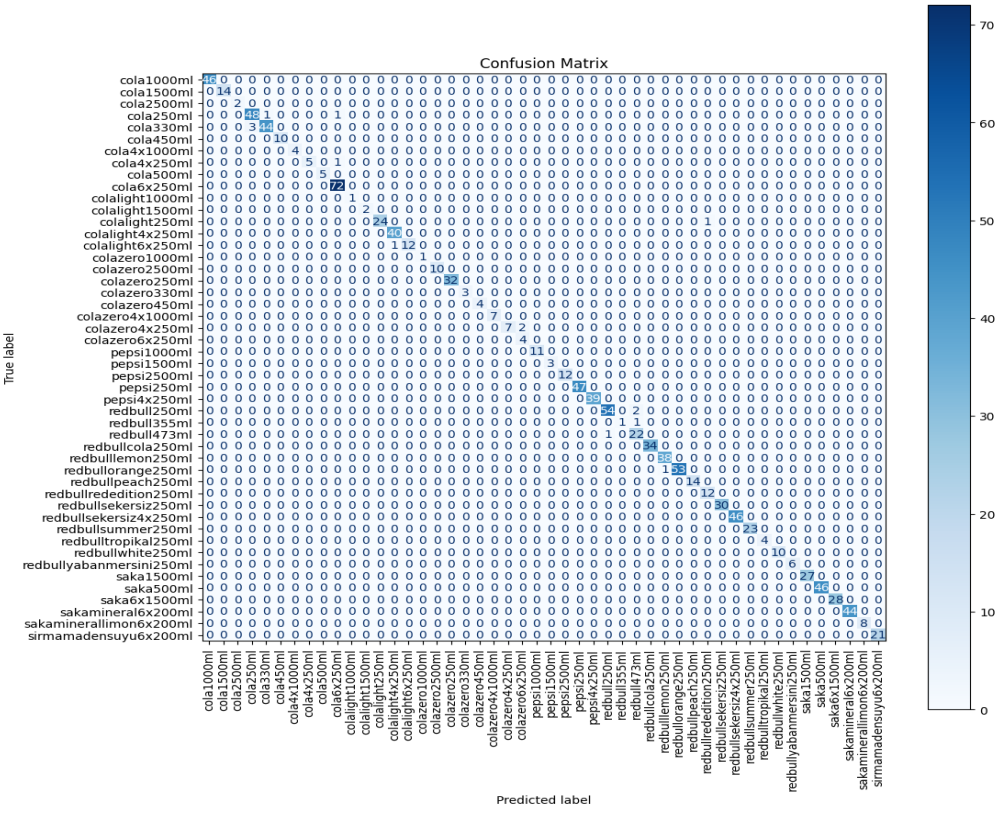

Figure 4.1 Confusion Matrix

CLASSIFICATION REPORT

- The classification report provides a summary of the model's performance for each class. It includes the following metrics:

    - Precision: The proportion of correctly predicted positive cases out of all predicted positive cases. Higher precision means fewer false positives.
    - Recall (Sensitivity): The proportion of correctly predicted positive cases out of all actual positive cases. Higher recall means fewer false negatives.
    - F1-Score: The harmonic mean of precision and recall, balancing both metrics.
    - Support: The number of actual samples for each class in the dataset.

- This report helps evaluate the model's performance for individual classes and overall, highlighting strengths and areas for improvement. Our classification report is in Figure 4.2.

	precision	recall	f1-score	support
cola1000ml	1.00	1.00	1.00	46
cola1500ml	1.00	1.00	1.00	14
cola2500ml	1.00	1.00	1.00	2
cola250ml	0.94	0.96	0.95	50
cola330ml	0.98	0.94	0.96	47
cola450ml	1.00	1.00	1.00	10
cola4x1000ml	1.00	1.00	1.00	4
cola4x250ml	1.00	0.83	0.91	6
cola500ml	1.00	1.00	1.00	5
cola6x250ml	0.97	1.00	0.99	72
colalight1000ml	1.00	1.00	1.00	1
colalight1500ml	1.00	1.00	1.00	2
colalight250ml	1.00	0.96	0.98	25
colalight4x250ml	0.98	1.00	0.99	40
colalight6x250ml	1.00	0.92	0.96	13
colazero1000ml	1.00	1.00	1.00	1
colazero2500ml	1.00	1.00	1.00	10
colazero250ml	1.00	1.00	1.00	32
colazero330ml	1.00	1.00	1.00	3
colazero450ml	1.00	1.00	1.00	4
colazero4x1000ml	1.00	1.00	1.00	7
colazero4x250ml	1.00	0.78	0.88	9
colazero6x250ml	0.67	1.00	0.80	4
pepsi1000ml	1.00	1.00	1.00	11
pepsi1500ml	1.00	1.00	1.00	3
pepsi2500ml	1.00	1.00	1.00	12
pepsi250ml	1.00	1.00	1.00	47
pepsi4x250ml	1.00	1.00	1.00	39
redbull250ml	0.98	0.96	0.97	56
redbull355ml	1.00	0.50	0.67	2
redbull473ml	0.88	0.96	0.92	23
redbullcola250ml	1.00	1.00	1.00	34
redbulllemon250ml	0.97	1.00	0.99	38
redbullorange250ml	1.00	0.98	0.99	54
redbullpeach250ml	1.00	1.00	1.00	14
redbullrededition250ml	0.92	1.00	0.96	12
redbullsekersiz250ml	1.00	1.00	1.00	30
redbullsekersiz4x250ml	1.00	1.00	1.00	46
redbullsummer250ml	1.00	1.00	1.00	23
redbulltropikal250ml	1.00	1.00	1.00	4
redbullwhite250ml	1.00	1.00	1.00	10
redbullyabanmersini250ml	1.00	1.00	1.00	6
saka1500ml	1.00	1.00	1.00	27
saka500ml	1.00	1.00	1.00	46
saka6x1500ml	1.00	1.00	1.00	28
sakamineral6x200ml	1.00	1.00	1.00	44
sakamineralimon6x200ml	1.00	1.00	1.00	8
sirmamadensuyu6x200ml	1.00	1.00	1.00	21
accuracy			0.99	1045
macro avg	0.99	0.97	0.98	1045
weighted avg	0.99	0.99	0.99	1045

Figure 4.2 Classification Report

TRAINING AND EVALUATION RESULTS

- After optimizing the model, the last measured accuracy values are shown in Table 4.3.
- As shown in Table 4.3, the accuracy value is 98.56% in the validation dataset with 15% of the data and 99.33% in the test dataset with the other 15% of the data.
- Finally, when we tried with 2 different epochs without changing the model in the transmitted Test dataset (483 samples), we obtained 86.13% at 30 epochs and 90.48% at 100 epochs.

Validation Accuracy (30 epoch)	% 98.56
Test Accuracy (30 epoch)	% 99.33

Last Test Accuracy (30 epoch)	% 86.13
Last Test Accuracy (100 epoch)	% 90.48

Table 4.3 Accuracy Rates

TRAIN - VALIDATION COMPARISON

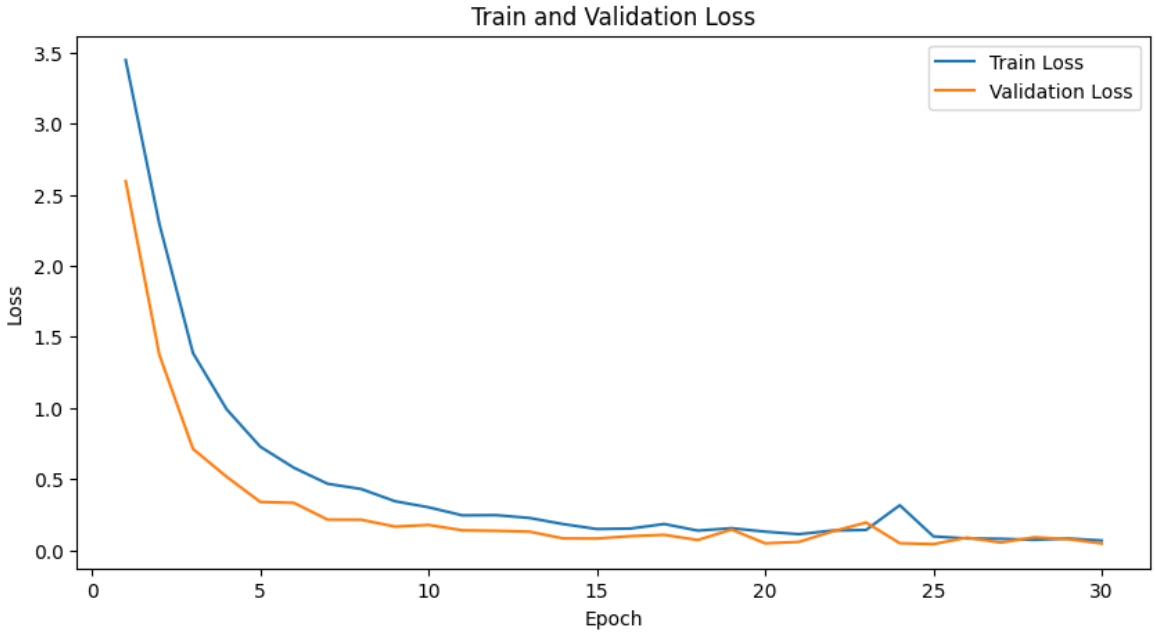

Figure 4.4 Train – Validation Loss Plot

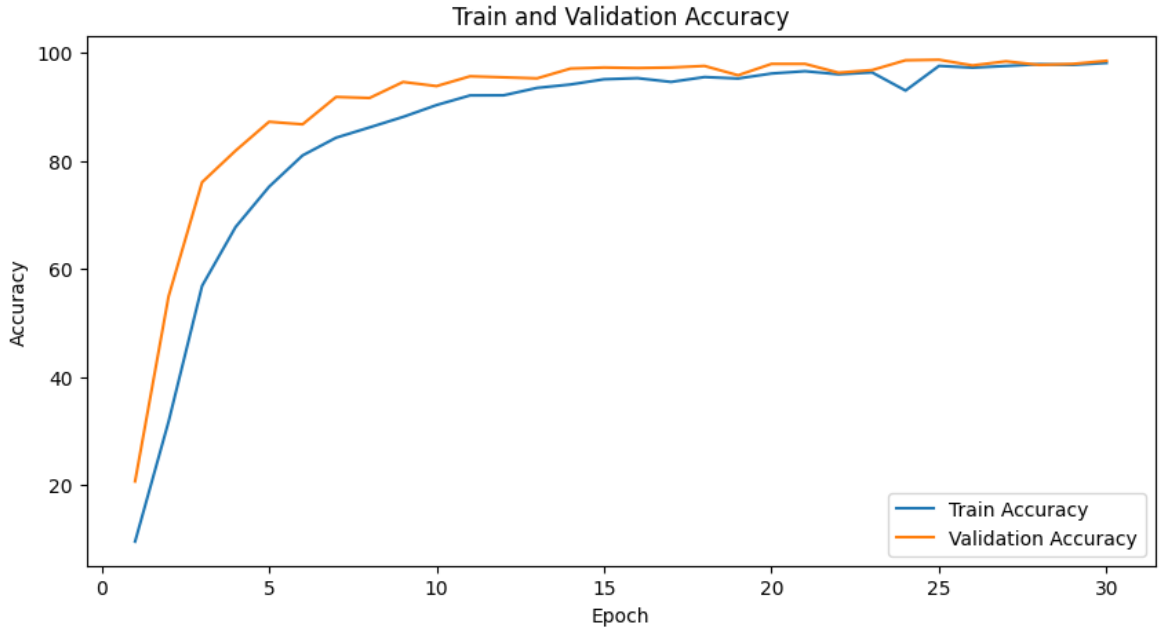

Figure 4.5 Train – Validation Accuracy Plot

5. CONCLUSION AND FEATURE WORK

Conclusion

This project successfully implemented a CNN for image classification, achieving competitive training and validation accuracy. Key contributions include effective data preprocessing, augmentation, and class weight balancing, which improved model stability and robustness. Regularization techniques like dropout and data augmentation reduced overfitting, ensuring better generalization to unseen data.

Future Work

1. Hyperparameter Optimization: Systematic tuning of parameters like learning rate and batch size can further improve performance.
2. Transfer Learning: Fine-tuning of pre-trained models, such as ResNet or EfficientNet, may boost accuracy and reduce training time.
3. Real-Time Deployment: Adapting the model to real-world applications, like inventory systems, would bring out its practical value.
4. Model Optimization: The model can be made lightweight for deployment on resource-constrained devices by using various techniques such as pruning and quantization.

6. REFERENCES

- Brownlee, J. (2019, November 22). Machine Learning Mastery. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/how-to-prepare-categoricaldata-for-deep-learning-in-python/
- Khan, M. A. (2023, January 7). Machine Learning Mastery. Retrieved from Building a Softmax Classifier for Images in PyTorch: https://machinelearningmastery.com/building-asoftmax-classifier-for-images-in-pytorch/
- PyTorch Team. (2023). PyTorch Documentation - Image Classification Guide. Retrieved from PyTorch: https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html
- Brownlee, J. (2020, October 19). *How to Prepare Categorical Data for Deep Learning in Python*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/
- Jung, A. (2020). *ImgAug: Image Augmentation Library*.Retrieved from ImgAug: https://imgaug.readthedocs.io/en/latest/index.html
- Discipline of ICT, School of TED, University of Tasmania, Launceston, Tasmania, Australia(2020: https://www.researchgate.net/publication/346856840_Deep_Learning_for_Retail_Product_Recognition_Challenges_and_Techniques
- Journal of Retailing (March 2017, Pages 1-6) :The Future of Retailing - ScienceDirect

APPENDIX A

```
import numpy as np
import seaborn as sns
from PIL import Image
import os
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
from torch.utils.data import random_split, DataLoader
from sklearn.metrics import confusion_matrix, classification_report
from torch.utils.data import ConcatDataset
```

Import Data Dictionary to Colab

```
[ ] from google.colab import drive

    drive.mount('/content/drive')
    zip_file_path = '/content/drive/MyDrive/Train.zip'
    extracted_folder_path = '/content'

    !unzip -q "{zip_file_path}" -d "{extracted_folder_path}"
```

```
[ ] zip_file_path = '/content/drive/MyDrive/Test.zip'
    extracted_folder_path = '/content'

    !unzip -q "{zip_file_path}" -d "{extracted_folder_path}"
```

```
[2] zip_file_path = '/content/Train.zip'
    extracted_folder_path = '/content'

    !unzip -q "{zip_file_path}" -d "{extracted_folder_path}"

    zip_file_path = '/content/Test.zip'
    extracted_folder_path = '/content'

    !unzip -q "{zip_file_path}" -d "{extracted_folder_path}"
```

Setted Variables

```
BATCH_SIZE = 32
NUM_WORKERS = 2

EPOCHS = 30
LEARNING_RATE = 0.001
WEIGHT_DECAY = 1e-4

IMG_SIZE = 128

TRAIN_PERCENTAGE = 0.7
VALIDATION_PERCENTAGE = 0.15
TEST_PERCENTAGE = 0.15
```

Generate Custom Dataset

```
class Dataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.data_dir = data_dir
        self.transform = transform
        self.classes = sorted(
            [folder for folder in os.listdir(data_dir) if not folder.startswith(".")]
        )
        self.data = []

        for label, class_name in enumerate(self.classes):
            class_path = os.path.join(data_dir, class_name)
            if os.path.isdir(class_path):
                for file_name in os.listdir(class_path):
                    file_path = os.path.join(class_path, file_name)
                    if file_name.lower().endswith(('.png', '.jpg', '.jpeg')) and not file_name.startswith("."):
                        self.data.append((file_path, label))

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

    def __getitem__(self, idx):
        image_path, label = self.data[idx]
        image = Image.open(image_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image, label
```

Setted Transform to Resize Image

```
[197] transform_1 = transforms.Compose([
          transforms.Resize((IMG_SIZE, IMG_SIZE)),
          transforms.ToTensor(),
          transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ])
```

```
[198] transform_2 = transforms.Compose([
          transforms.Resize((IMG_SIZE, IMG_SIZE)),
          transforms.RandomHorizontalFlip(p=0.5),
          transforms.RandomRotation(degrees=15),
          transforms.RandomResizedCrop(size=IMG_SIZE, scale=(0.9, 1.0)),
          transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.05),
          transforms.ToTensor(),
          transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ])
```

Generate Dataset with Resize

```
9] dataset_path = f"{extracted_folder_path}/Train"
   dataset_1 = Dataset(data_dir=dataset_path, transform=transform_1)
   dataset_2 = Dataset(data_dir=dataset_path, transform=transform_2)
   dataset = ConcatDataset([dataset_1, dataset_2])

   NUM_CLASSES = len(dataset_1.classes)

   print("Classes:")
   print(dataset_1.classes)
   print("Classes Size: ", NUM_CLASSES)
   print("Total Size: ", len(dataset))
```

```
Classes:
['cola1000ml', 'cola1500ml', 'cola2500ml', 'cola250ml', 'cola330ml
Classes Size:  48
Total Size:  13938
```

Split Dataset

```
0] from sklearn.model_selection import train_test_split

   dataset_size = len(dataset)
   train_size = int(TRAIN_PERCENTAGE * dataset_size)
   val_size = int(VALIDATION_PERCENTAGE * dataset_size)
   test_size = dataset_size - train_size - val_size

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

   print(f"Total Size: {len(dataset)}\nTrain Size: {len(train_dataset)}\nValidation Size: {len(val_dataset)}\nTest Size: {len(test_dataset)}")
```

Generate Loaders

```
] train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=NUM_WORKERS)
```

```
] val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=NUM_WORKERS)
```

APPENDIX B

```python
class Model(nn.Module):
    def __init__(self, num_classes):
        super().__init__()

        # Convolutional Layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)  # Output: 128x128x32
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)  # Output: 64x64x32

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)  # Output: 64x64x64
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)  # Output: 32x32x64

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)  # Output: 32x32x128
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)  # Output: 16x16x128

        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)  # Output: 16x16x256
        self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2)  # Output: 8x8x256

        # Fully Connected Layers
        self.fc1 = nn.Linear(8 * 8 * 256, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, num_classes)

        # Regularization
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):

        # Convolutional Layers
        x = self.pool1(nn.ReLU()(self.conv1(x)))  # Output: 64x64x32
        x = self.pool2(nn.ReLU()(self.conv2(x)))  # Output: 32x32x64
        x = self.pool3(nn.ReLU()(self.conv3(x)))  # Output: 16x16x128
        x = self.pool4(nn.ReLU()(self.conv4(x)))  # Output: 8x8x256

        # Flatten
        x = torch.flatten(x, 1) # Output: 8*8*256 = 16384

        # Fully Connected Layers
        x = nn.ReLU()(self.fc1(x))
        x = self.dropout(x)
        x = nn.ReLU()(self.fc2(x))
        x = self.dropout(x)
        x = nn.ReLU()(self.fc3(x))
        x = self.fc4(x)

        return x
```

Calculate Weight for Whole Class (For classes with few samples)

```
from collections import Counter

all_labels = [label for _, label in dataset]

class_counts = Counter(all_labels)

num_classes = len(dataset_1.classes)
class_weights = torch.zeros(num_classes, dtype=torch.float)

for cls, count in class_counts.items():
    label_name = dataset_1.classes[cls]
    print(cls, ") ", label_name, " : ", count, " | ", 1.0/count)
    class_weights[cls] = 1.0 / count

print("Class weights:", class_weights)
```

Generate Model, Loss Function and Optimizer

```
model = Model(NUM_CLASSES)
class_weights = class_weights.to(device)
loss_fn = nn.CrossEntropyLoss(weight=class_weights)
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY)
```

Move Model to GPU

```
[205] print(f"Using device: {device}")
      model.to(device)
```

```
Using device: cuda
Model(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=16384, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=128, bias=True)
  (fc4): Linear(in_features=128, out_features=48, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
```

Optimize Model Function

```
] def optimize_model(mdl, loader):
      running_loss = 0.0
      correct_train = 0
      total_train = 0

      mdl.train()

      for i, data in enumerate(loader):
          inputs, labels = data
          inputs, labels = inputs.to(device), labels.to(device)

          optimizer.zero_grad()
          outputs = mdl(inputs)
          loss = loss_fn(outputs, labels)
          loss.backward()
          optimizer.step()

          running_loss += loss.item()

          # Calculate training accuracy
          _, predicted = torch.max(outputs, 1)
          total_train += labels.size(0)
          correct_train += (predicted == labels).sum().item()

      return running_loss, correct_train, total_train
```

Evaluate Mode Function

```
7] def evaluate_model(mdl, loader):
       running_loss = 0.0
       correct_val = 0
       total_val = 0

       mdl.eval()

       with torch.no_grad():
         for data in loader:
           images, labels = data
           images, labels = images.to(device), labels.to(device)

           outputs = mdl(images)
           loss = loss_fn(outputs, labels)
           running_loss += loss.item()

           _, predicted = torch.max(outputs, 1)
           total_val += labels.size(0)
           correct_val += (predicted == labels).sum().item()

       return running_loss, correct_val, total_val
```

Get Optimization and Evaluation Results

```
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []
```

```
for epoch in range(EPOCHS):
    running_loss, correct_train, total_train = optimize_model(model, train_loader)

    train_loss = running_loss / len(train_loader)
    train_losses.append(train_loss)
    train_accuracy = correct_train / total_train
    train_accuracies.append(train_accuracy * 100)

    runing_loss, correct_val, total_val = evaluate_model(model, val_loader)

    val_loss = runing_loss / len(val_loader)
    val_losses.append(val_loss)
    val_accuracy = correct_val / total_val
    val_accuracies.append(val_accuracy * 100)

    print(f"Epoch [{epoch+1}/{EPOCHS}], "
        f"Train Loss: {train_loss:.2f}, Val Loss: {val_loss:.2f}, "
        f"Train Acc: {train_accuracy:.2f}, Val Acc: {val_accuracy:.2f}")
```

Setup Test Dataset

```
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=NUM_WORKERS)
```

```
torch.save({
    "test_dataset": test_dataset
}, "test_dataset.pth")
```

Calculate Accurancy for Test Dataset

```
running_loss, correct_test, total_test = evaluate_model(model, test_loader)

test_loss = running_loss / len(test_loader)
test_accuracy = (100 * correct_test) / total_test

print(f"Test Loss: {test_loss:.2f}, Test Accuracy: {test_accuracy:.2f}%")
```

Last Test Part

```
test_dataset_path = f"{extracted_folder_path}/Test"
test_dataset_last = Dataset(data_dir=test_dataset_path, transform=transform_1)

NUM_CLASSES = len(test_dataset_last.classes)

print("Classes:")
print(test_dataset_last.classes)
print("Classes Size: ", NUM_CLASSES)
print("Total Size: ", len(test_dataset_last))
```

```
test_loader_last = DataLoader(test_dataset_last, batch_size=BATCH_SIZE, shuffle=False, num_workers=NUM_WORKERS)
```

luate Last Test

```
running_loss, correct_test_last, total_test_last = evaluate_model(model, test_loader_last)

test_loss_last = running_loss / len(test_loader_last)
test_accuracy_last = (100 * correct_test_last) / total_test_last

print(f"Last Test Loss: {test_loss_last:.2f}, Last Test Accuracy: {test_accuracy_last:.2f}%")
```