DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GARBAGE CLASSIFIER

IN THE GUIDANCE OF
NEELIMA SANTOSHI.K, M. TECH
Assistant Professor
GITAM School Of Technology
Visakhapatnam

By
N.Akhil Kumar 121910309040
E.Kailash Kumar 121910309041
Harsh Kumar Gupta 121910309063
D.Shiva 121910309052

INTRODUCTION:

Garbage classification refers to the process of dividing waste into different categories based on their composition and properties.

It is an important aspect of waste management as it helps in the efficient disposal and treatment of waste, reducing its impact on the environment.

The classification of waste typically includes categories such as organic, recyclable, and hazardous waste.

With the increase in waste generation and the need for efficient waste management, there has been growing interest in using machine learning techniques for garbage classification.

ABSTRACT

- This project aims to develop a model for the classification of garbage into different categories, such as organic, recyclable, and hazardous waste, using multiple machine learning techniques including Convolutional Neural Network (CNN), Residual Network .

- The models are trained and evaluated on a dataset of images of garbage, and the results show that the ResNet model outperforms the other models in terms of accuracy.

- The implementation of these models and their comparison can provide insights into the most effective method for garbage classification, which has important implications for waste management and reduction of environmental impact.

MORE INFORMATION

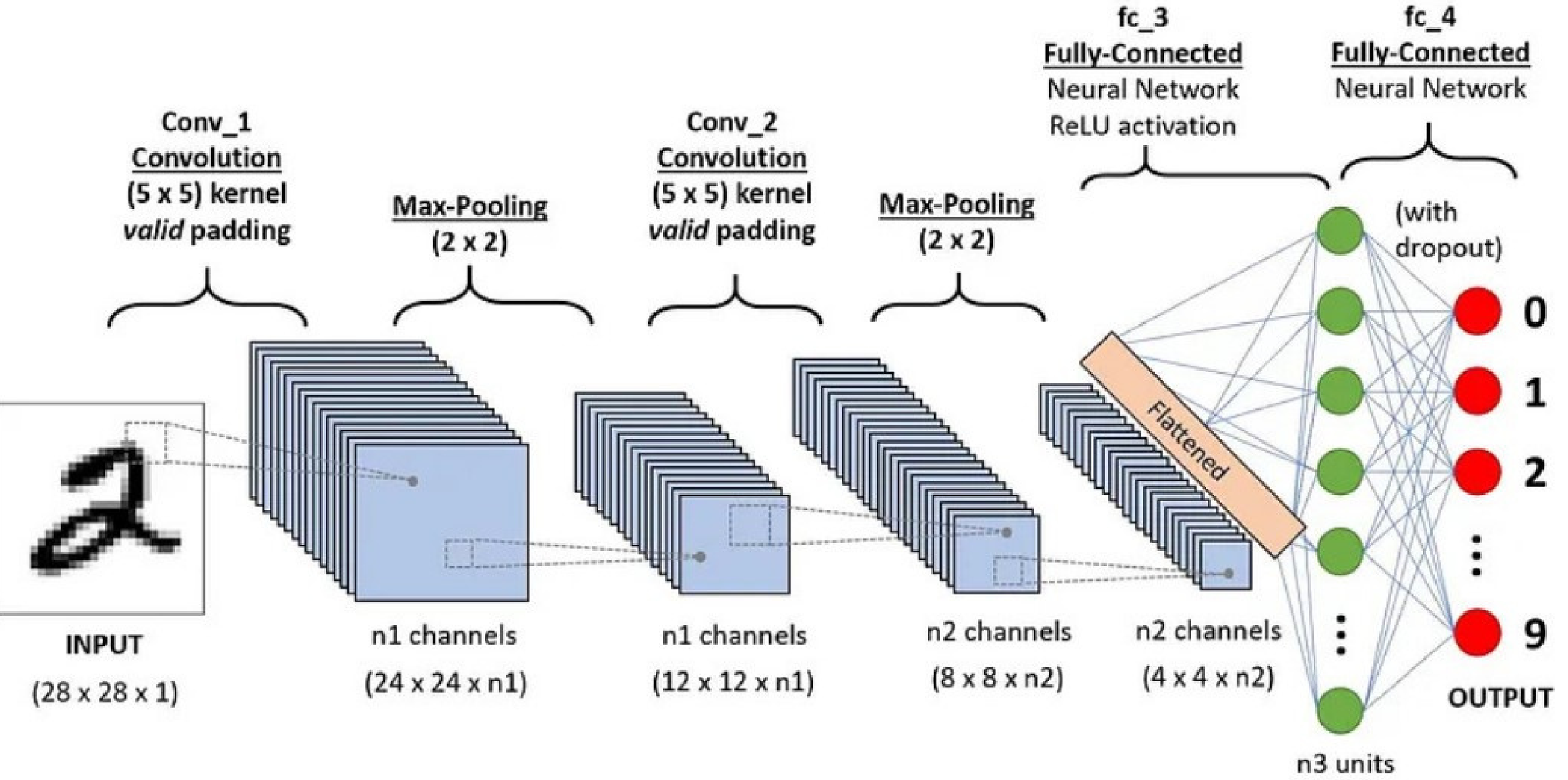
Problem Statement

1. **Data imbalance: If the dataset used to train the classifier is imbalanced , the classifier may be biased towards the dominant class and not perform well on the underrepresented class.**
2. **Overfitting: The classifier may be too complex and overfit the training data, meaning it does not generalize well to new, unseen data.**
3. **Outdated data: The classifier may be trained on outdated data and not reflect current trends or changes in the data distribution.**
4. **Poor feature selection: The classifier may be trained on irrelevant or noisy features, leading to poor performance.**
5. **Inadequate training data: If there is not enough training data available, the classifier may not have sufficient information to learn an accurate model.**
6. **Insufficient pre-processing: Data pre-processing, such as normalization and feature scaling, is critical for the performance of a classifier. If pre-processing is not done correctly, the classifier may perform poorly.**

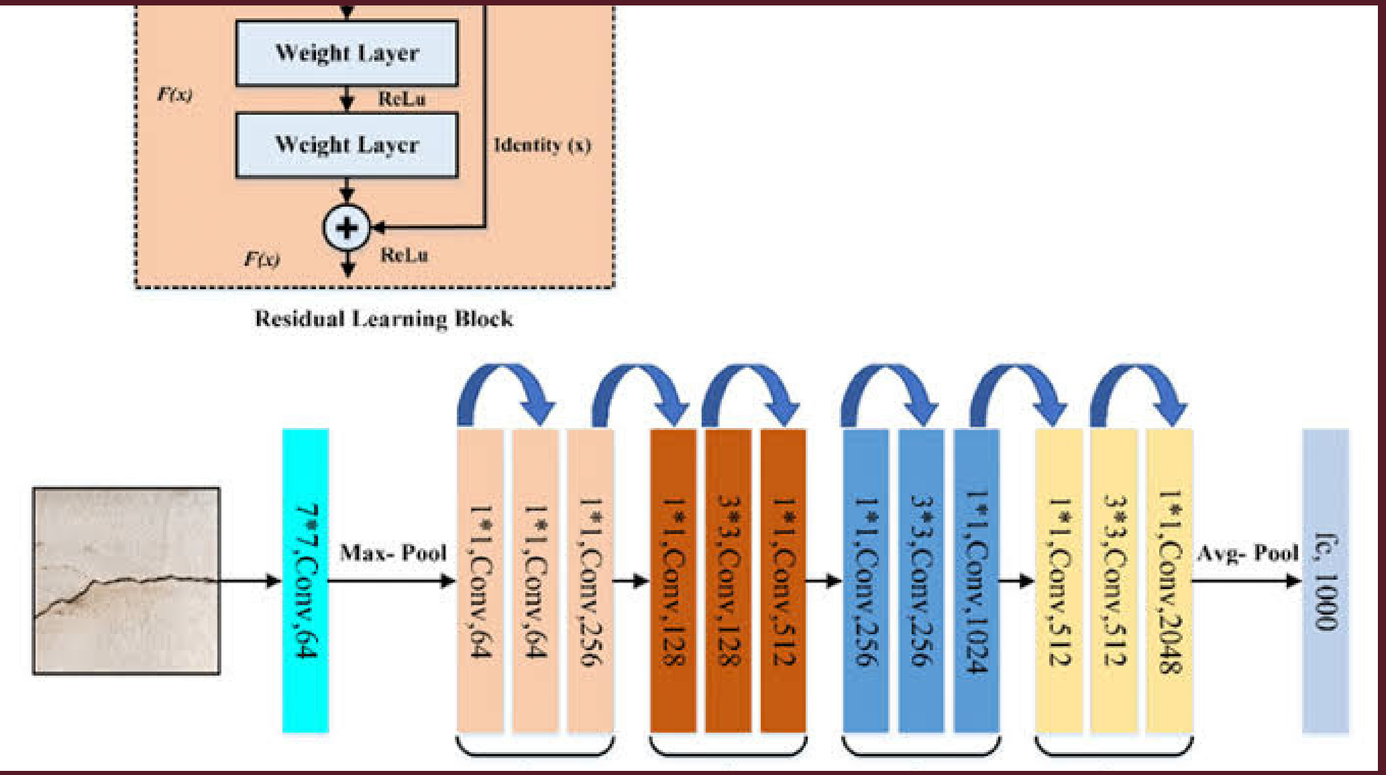
e.

CONVOLUTIONAL NEURAL NETWORK

- Type of deep learning neural network that is primarily used for image recognition and processing. It's called "convolutional" because it uses a mathematical operation called convolution to scan the input image, looking for specific patterns and features.
- These patterns are then transformed into a higher level representation and processed through multiple layers of the network, with each layer detecting more complex features. Finally, the output of the last layer is used for classification or prediction.
- In comparison to traditional feedforward neural networks, CNNs have several key features that make them well suited for image processing, such as local connectivity, shared weights, and pooling layers.
- These features allow CNNs to efficiently process and identify the important features in an image while reducing the number of parameters that need to be learned.
- This makes CNNs a powerful tool in computer vision and has led to many breakthroughs in image classification and object detection tasks.


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with
dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

ResNet-50



1.ResNet-50 is a deep convolutional neural network architecture that was introduced in 2015 by Microsoft researchers. It consists of 50 layers and is used for various computer vision tasks such as image classification, object detection, and semantic segmentation.

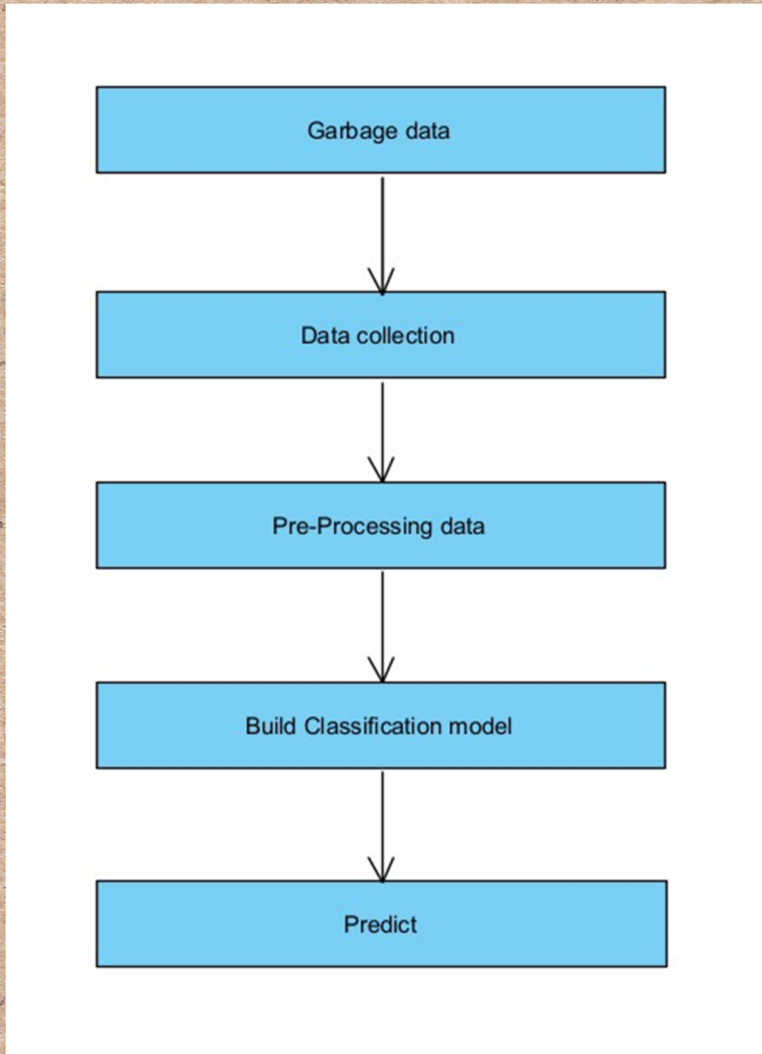
2 .The key innovation of ResNet-50 is the use of residual connections, which allow information to flow directly from one layer to another without being affected by the activation functions in between. This helps to address the vanishing gradient problem

UML DIAGRAM

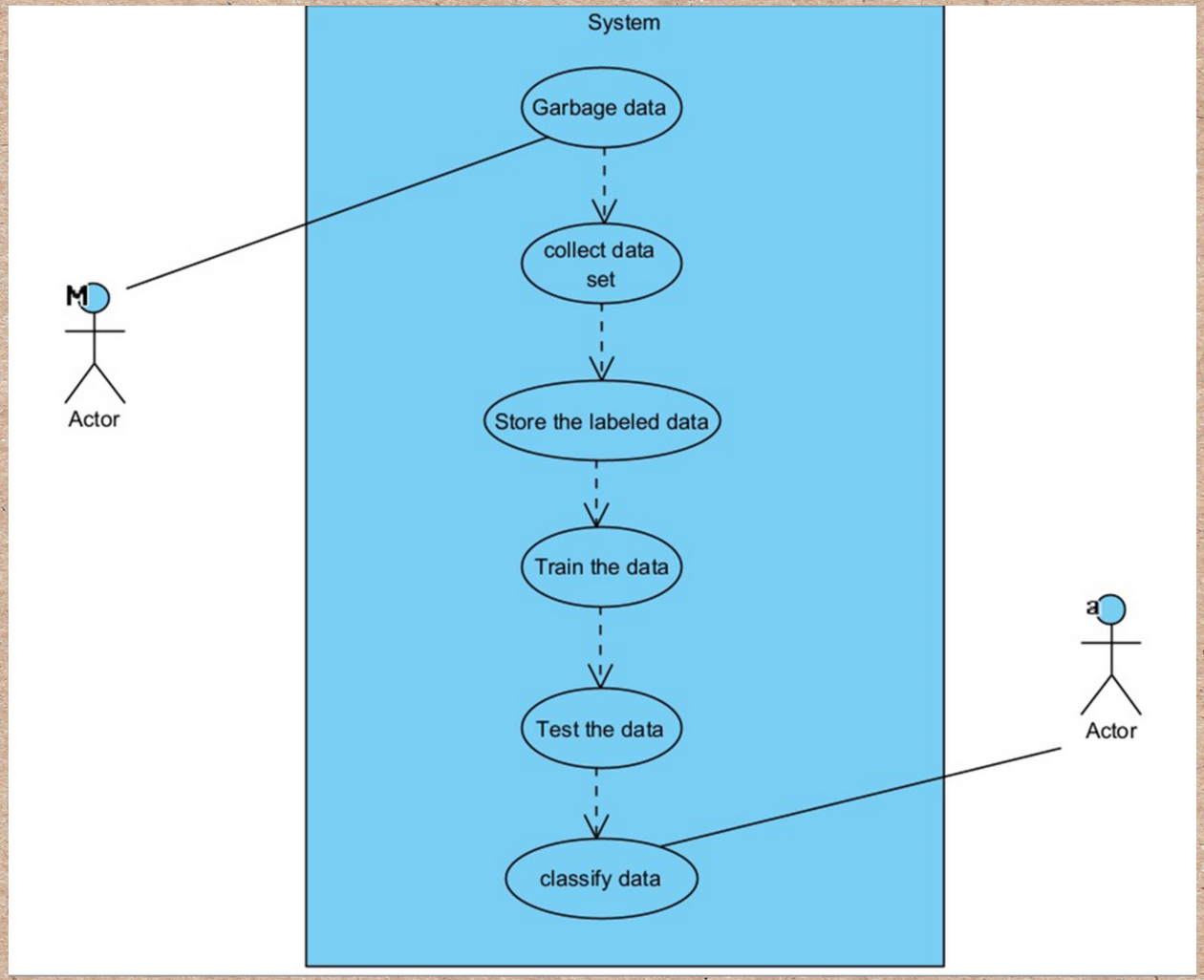
- UML (Unified Modeling Language) is a standardized visual language for modeling and documenting software systems and their components, including their structure, behavior, and interactions. It provides a set of symbols, diagrams, and notations that can be used to represent software components, such as classes, objects, use cases, sequences, and state machines.

- UML is used by software developers, architects, and analysts to design, communicate, and visualize software systems before they are built. UML is used across a variety of domains, from small systems to large and complex enterprise systems.

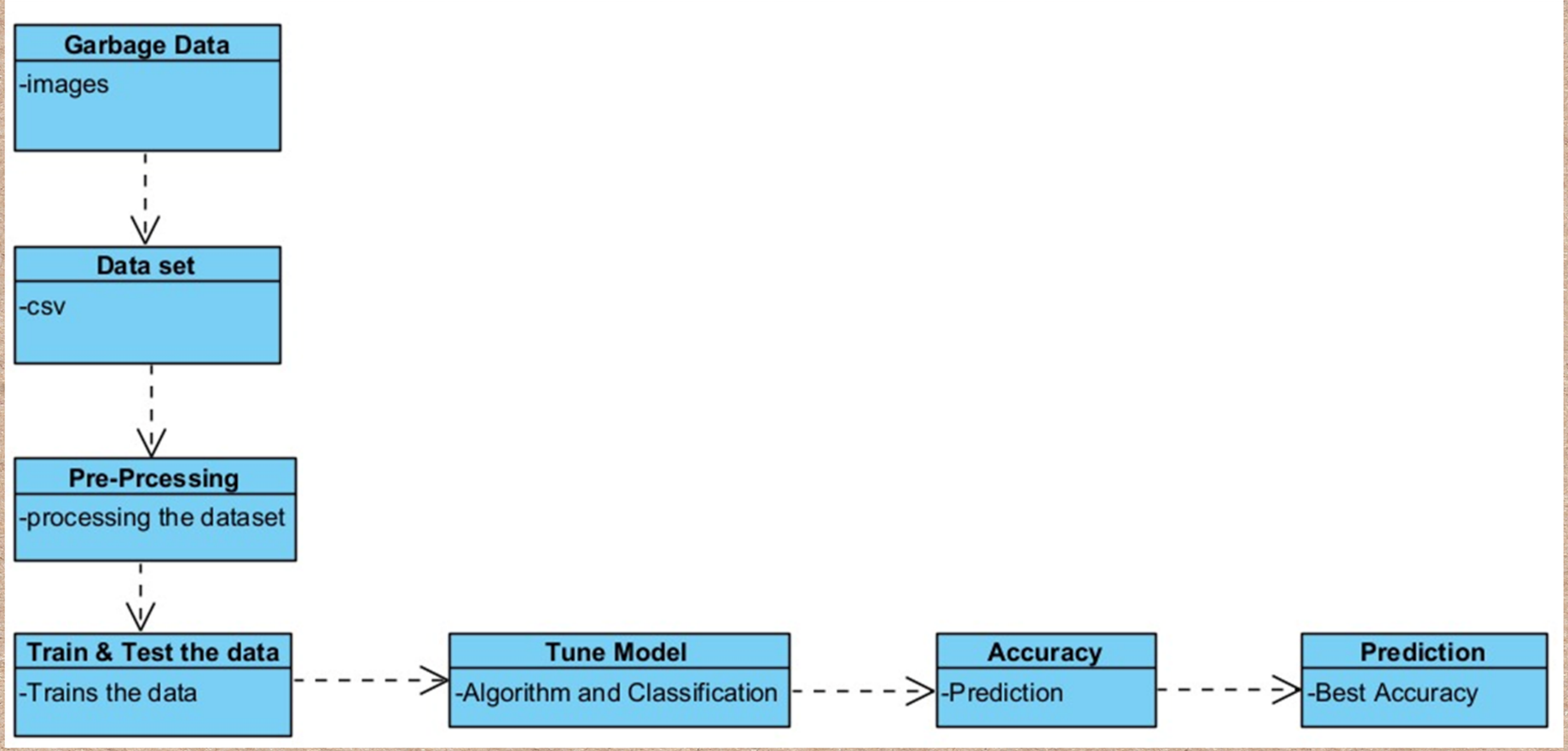
DATA FLOW DIAGRAM



USE CASE DIAGRAM



CLASS DIAGRAM



Dataset

```
import os
import torch
import torchvision
from torch.utils.data import random_split
import torchvision. (module) nn els
import torch.nn as nn
import torch.nn.functional as F
```

22] ✓ 0.7s

Let us see the classes present in the dataset:

```
data_dir  = "C:/Users/Kailash/Downloads/archive/Garbageclassification/Garbageclassification"

classes = os.listdir(data_dir)
print(classes)
```

23] ✓ 0.0s

```
['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash']
```

Resize an image

applying transformations to dataset and import it for use.

+ Code + Markdown

```
#Responsible for converting images (resizing and adding it to folder)
from torchvision.datasets import ImageFolder
import torchvision.transforms as transforms

transformations = transforms.Compose([transforms.Resize((256, 256)), transforms.ToTensor()])

dataset = ImageFolder(data_dir, transform = transformations)
```

✓ 0.1s

Loading and Splitting Data

```
#loading and splitting the data
random_seed = 42
torch.manual_seed(random_seed)
```

```
train_ds, val_ds, test_ds = random_split(dataset, [1593, 176, 758])
len(train_ds), len(val_ds), len(test_ds)
```

```
#Training and Validation part
train_dl = DataLoader(train_ds, batch_size, shuffle = True, num_workers = 4, pin_memory = True)
val_dl = DataLoader(val_ds, batch_size*2, num_workers = 4, pin_memory = True)
```

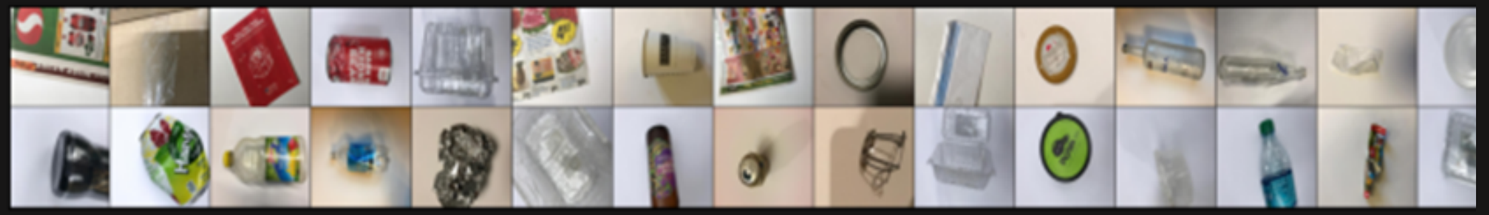
Now, we'll create training and validation dataloaders using DataLoader


```
#Training and Validation part
train_dl = DataLoader(train_ds, batch_size, shuffle = True, num_workers = 4, pin_memory = True)
val_dl = DataLoader(val_ds, batch_size*2, num_workers = 4, pin_memory = True)
```

Used to visualize data

```
from torchvision.utils import make_grid

def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(12, 6))
        ax.set_xticks([])
        ax.set_yticks([])
        ax.imshow(make_grid(images, nrow = 12).permute(1, 2, 0))
        break
```

```
show_batch(train_dl)
```



Model Base

Let's create the model base.

```python
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))

class ImageClassificationBase(nn.Module):
    def training_step(self, batch):
        images, labels = batch
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch {}: train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
            epoch+1, result['train_loss'], result['val_loss'], result['val_acc']))
```

✓ 0.1s

ResNet50

```
class ResNet(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        # Use a pretrained model
        self.network = models.resnet50(pretrained=True)
        # Replace last layer
        num_ftrs = self.network.fc.in_features
        self.network.fc = nn.Linear(num_ftrs, len(dataset.classes))

    def forward(self, xb):
        return torch.sigmoid(self.network(xb))

model = ResNet()
```

✓ 1.1s

Porting to GPU : GPUs tend to perform faster calculations than CPU

```
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):

    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():

    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):

        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        """Number of batches"""
        return len(self.dl)
```

```
device = get_default_device()
device
```

```
train_dl = DeviceDataLoader(train_dl, device)
val_dl = DeviceDataLoader(val_dl, device)
to_device(model, device)
```

```python
class ResNet(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        # Use a pretrained model
        self.network = models.resnet50(pretrained=True)
        # Replace last layer
        num_ftrs = self.network.fc.in_features
        self.network.fc = nn.Linear(num_ftrs, len(dataset.classes))

    def forward(self, xb):
        return torch.sigmoid(self.network(xb))

model = ResNet()
```

Function for fitting the model:

```python
@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):

        model.train()
        train_losses = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

```
model = to_device(ResNet(), device)
```

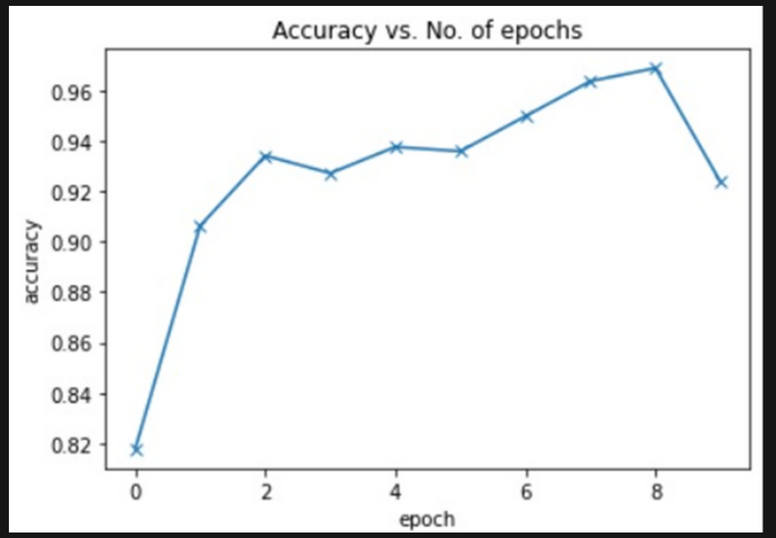
```
evaluate(model, val_dl)
```

Training the model process:

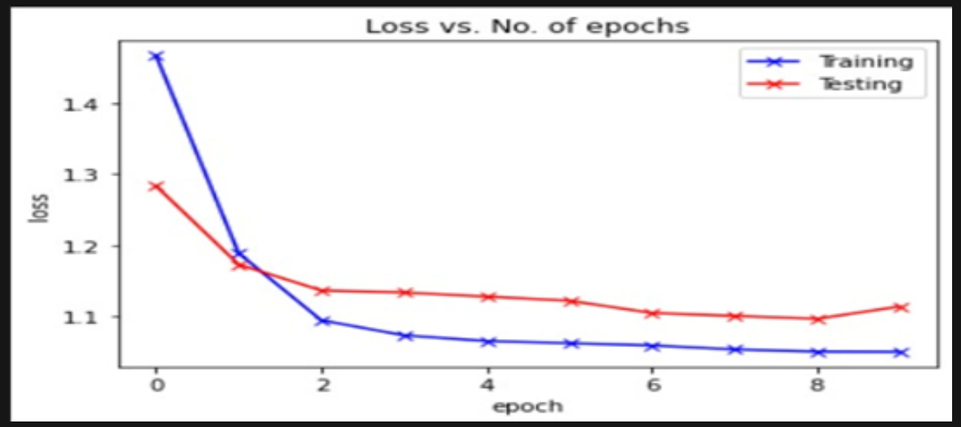
```
num_epochs = 10
opt_func = torch.optim.Adam
lr = 5.5e-5

history = fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

```
Epoch 1: train_loss: 1.4668, val_loss: 1.2835, val_acc: 0.8177
Epoch 2: train_loss: 1.1888, val_loss: 1.1730, val_acc: 0.9062
Epoch 3: train_loss: 1.0946, val_loss: 1.1364, val_acc: 0.9340
Epoch 4: train_loss: 1.0733, val_loss: 1.1336, val_acc: 0.9271
Epoch 5: train_loss: 1.0653, val_loss: 1.1278, val_acc: 0.9375
Epoch 6: train_loss: 1.0623, val_loss: 1.1221, val_acc: 0.9358
Epoch 7: train_loss: 1.0591, val_loss: 1.1048, val_acc: 0.9497
Epoch 8: train_loss: 1.0535, val_loss: 1.1007, val_acc: 0.9635
Epoch 9: train_loss: 1.0505, val_loss: 1.0966, val_acc: 0.9688
Epoch 10: train_loss: 1.0501, val_loss: 1.1139, val_acc: 0.9236
```

```
def (variable) accuracies: list
    accuracies = [x['val_acc'] for x in history]
    plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.title('Accuracy vs. No. of epochs');

plot_accuracies(history)
```




```
def plot_losses(history):
    train_losses = [x.get('train_loss') for x in history]
    val_losses = [x['val_loss'] for x in history]
    plt.plot(train_losses, '-bx')
    plt.plot(val_losses, '-rx')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['Training', 'Testing'])
    plt.title('Loss vs. No. of epochs');

plot_losses(history)
```

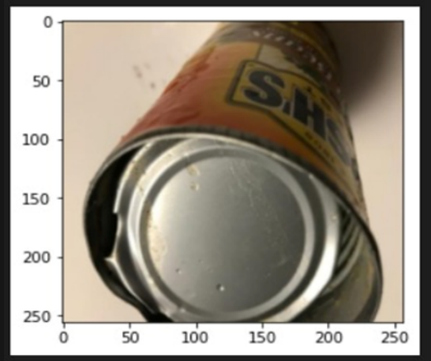


Visualizing Predictions:

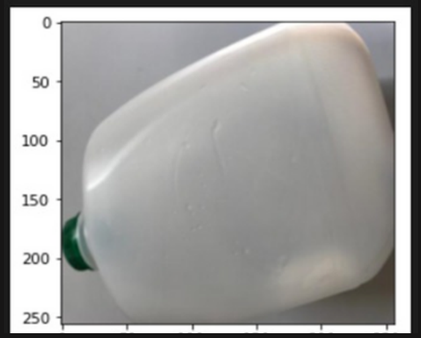
```
def predict_image(img, model):

    xb = to_device(img.unsqueeze(0), device)

    yb = model(xb)

    prob, preds  = torch.max(yb, dim=1)

    return dataset.classes[preds[0].item()]
```

```
img, label = test_ds[17]
plt.imshow(img.permute(1, 2, 0))
print('Label:', dataset.classes[label], ', Predicted:', predict_image(img, model))
```

```
Label: metal , Predicted: metal
```




```
img, label = test_ds[45]
plt.imshow(img.permute(1, 2, 0))
print('Label:', dataset.classes[label], ', Predicted:', predict_image(img, model))
```



Any questions?

yes!