Waste Classification Using Deep Learning Models

CS6005 - Deep Learning Techniques

Project Phase II Report

Team Details

Name	Roll Number	Batch
Karthik S	2020103531	R
Muthuraja Mounika	2020103543	R
Sai Methish Kumar	2020103560	R

NEED FOR THE SYSTEM

Effective waste classification is essential for addressing the challenges of a growing global population and escalating consumption rates. As waste production continues to rise, so too does the need for efficient and sustainable waste management solutions. Improper waste disposal practices can have devastating consequences for the environment and public health, while the underutilization of recyclable materials represents a significant loss of valuable resources.

Manual waste classification is a labor-intensive and costly process, highlighting the importance of automation in reducing human and monetary expenditures associated with waste management. Automation holds the potential to significantly enhance waste detection, sorting, and classification processes, ultimately making waste management more streamlined, sustainable, and economically viable.

Here are some specific ways in which automation can improve waste classification:

- **Accuracy:** Automated waste classification systems can achieve higher accuracy rates than manual sorting, resulting in fewer errors and improved efficiency.
- **Speed:** Automated systems can process waste much faster than humans, enabling faster and more efficient waste management.
- **Consistency:** Automated systems are less prone to human error, ensuring consistent and reliable waste classification results.
- **Cost-effectiveness:** Automated systems can reduce the need for manual labor, resulting in significant cost savings over time.
- **Safety:** Automated systems can reduce the risk of exposure to hazardous waste materials for workers, improving workplace safety.

In addition to these benefits, automation can also help to promote sustainability and circularity in waste management. By accurately identifying and separating recyclable materials, automated systems can facilitate increased recycling rates and reduce the amount of waste that is sent to landfills. This can help to conserve natural resources, reduce greenhouse gas emissions, and create new jobs in the recycling and reuse industries.

Overall, automation is a key enabler of effective waste classification and sustainable waste management. By embracing automation, we can create a more efficient, cost-effective, and environmentally friendly waste management system that protects our planet and safeguards public health.

OBJECTIVE OF THE MODEL:

This paper aims to develop an Integrated Waste Management System with a primary focus on creating a comprehensive and technologically advanced approach to waste recognition, sorting, and classification. Leveraging deep learning techniques, including Faster Region-Based Convolutional Neural Networks (Faster RCNN), ResNet50, and MobileNetV2, our goal is to automate waste classification and significantly enhance the accuracy of waste sorting processes. By doing so, we intend to promote environmental sustainability by facilitating recycling and proper waste disposal methods, thus contributing to the protection of our environment and overall sustainability goals. Additionally, the implementation of automated waste management solutions will play a vital role in reducing the economic burden associated with manual waste handling, making the process more cost-effective and efficient. Ultimately, we aspire to foster the adoption of advanced technologies, such as deep learning and computer vision techniques, in the field of waste management, with the long-term vision of improving overall efficiency and sustainability in waste management practices.

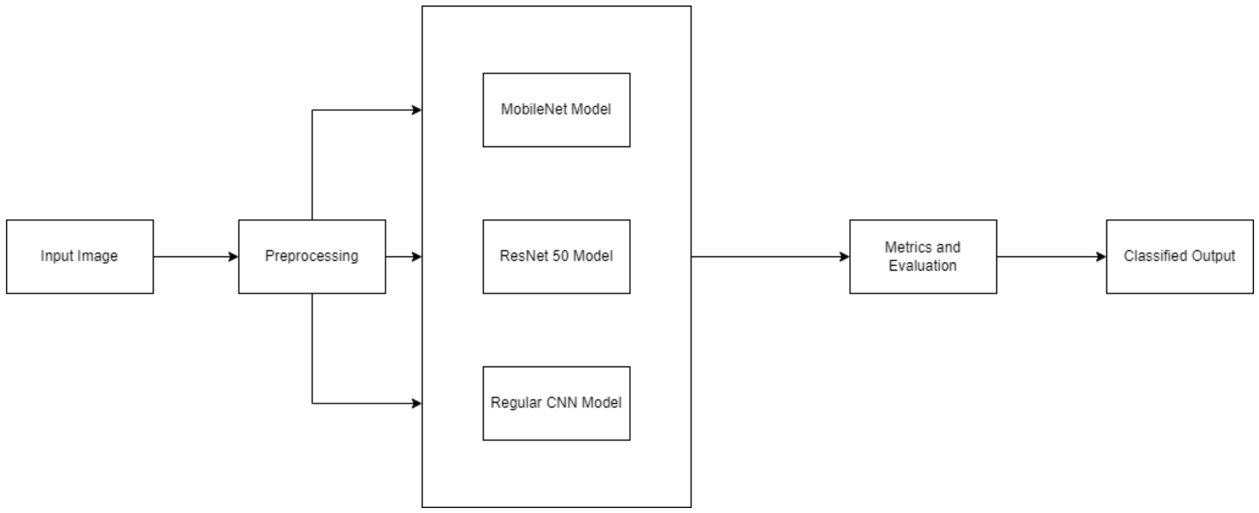
DATASET DESCRIPTION

Classification of recycling materials is vital for the environment, humanity, and civilization. The TrashNet dataset serves as a representative dataset for trash classification, and it has been widely used in various studies to evaluate proposed approaches. Enhancing the accuracy of this dataset is crucial for improving trash classification models.

Dataset Details:

- **Background:** The images in the dataset were placed on a white background and were captured under sunlight and/or room lighting conditions.
- **Image Preprocessing:** Each image in the dataset was preprocessed to a resolution of 512x384 pixels.
- **Dataset Size:** The original dataset comprises approximately 3.5 GB of data with more than 2500 images.
- **Classes:** The dataset consists of multiple classes representing different types of trash, such as cardboard, metal, glass, plastic, paper, and a dedicated class for "trash."

SYSTEM DESIGN DIAGRAM



The above system diagram describes the basic parts/modules of the proposed system. The major components are the pre-processing section and the model section.

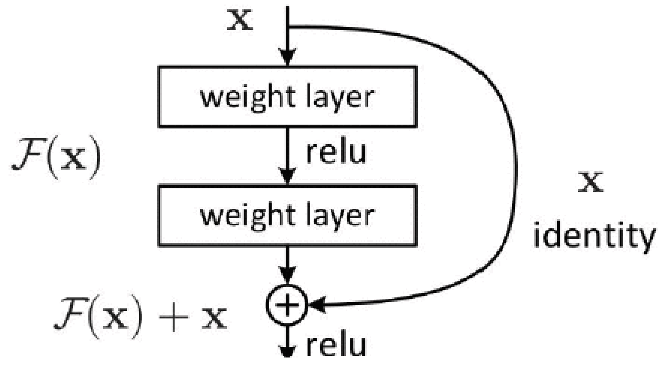
1. **Preprocessing:**
   In this module, the images of the dataset are transformed to resize it into 224 * 224 to make it suitable for the model.
2. **Models:**
   This module consists of the models, ResNet 50, MobileNet and a Regular CNN Model. The preprocessed image is fed to these models and the outputs are compared.
3. **Metrics and Evaluation:**
   The model is subjected to a metric and evaluation method called Accuracy.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

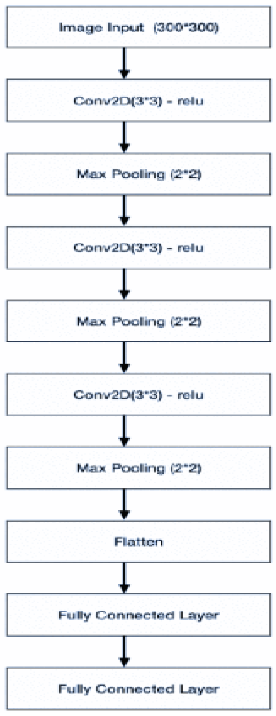
LAYER ARCHITECTURE OF THE DESIGNED MODEL

The following models are planned to be used inorder to compare, classify and recognize images from trash datasets.

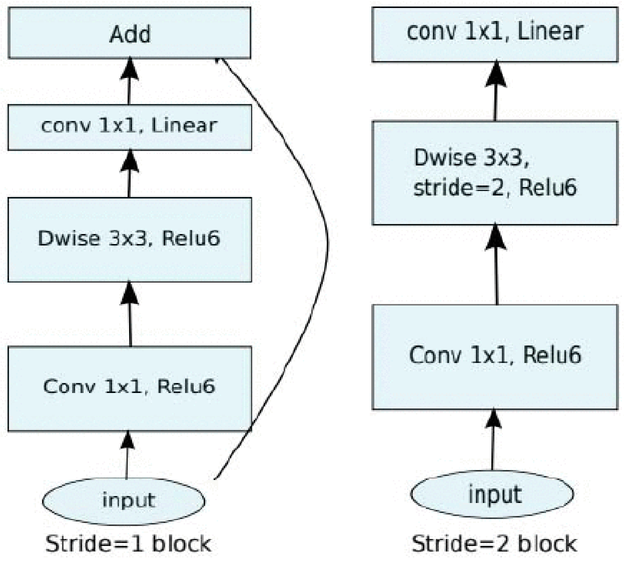
1. **ResNet 50 :** ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks. It consists of 23 million trainable parameters.


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

2. **Regular Convolutional Neural Network**



3. **MobileNet :** MobileNet is a type of convolutional neural network designed for mobile and embedded vision applications. They are based on a streamlined architecture that uses depth wise separable convolutions to build lightweight deep neural networks that can have low latency for mobile and embedded devices. It consists of 13 million parameters.



HOW MODEL WORKS WITH THE DATASET:

The chosen dataset consists of images of trash. Convolutional Neural Networks (CNNs) are the preferred choice for training image datasets due to their unique architecture designed for recognizing spatial hierarchies, achieving translation invariance, and efficiently extracting local features. They employ convolutional layers to capture patterns, share parameters to reduce overfitting, and enable transfer learning with pretrained models.

This dataset contains images of trash items with different classes, such as cardboard, metal, glass, plastic, paper, and a class for "trash".

TOOLS AND SOFTWARE REQUIREMENTS:

1. **PyTorch :** PyTorch is an open source machine learning (ML) framework based on the Python programming language and the Torch library.
2. **Keras :** Keras is an open-source library that provides a Python interface for artificial neural networks.
3. **Python :** Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It is predominantly used in data science.
4. **Google Colaboratory :** Google Colaboratory, often referred to as Colab, is a cloud-based platform provided by Google for working with and running Python code in a Jupyter Notebook environment.
5. **Visual Studio Code:** Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux and macOS.

Implementation Details:

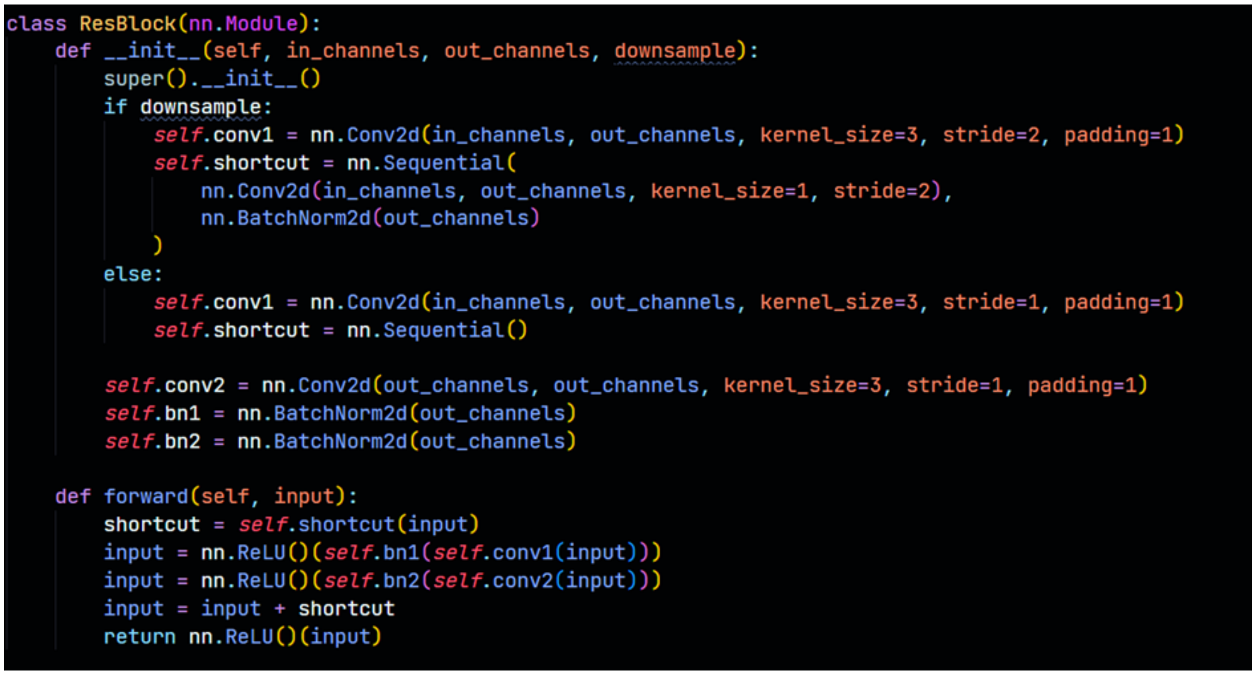
Data Preprocessing:

The images in the dataset have been preprocessed to a uniform resolution of 224x224 pixels. Preprocessing includes, random brightness contrast transformations, rotations and flips and resizing the image to 224 x 224 pixels each.

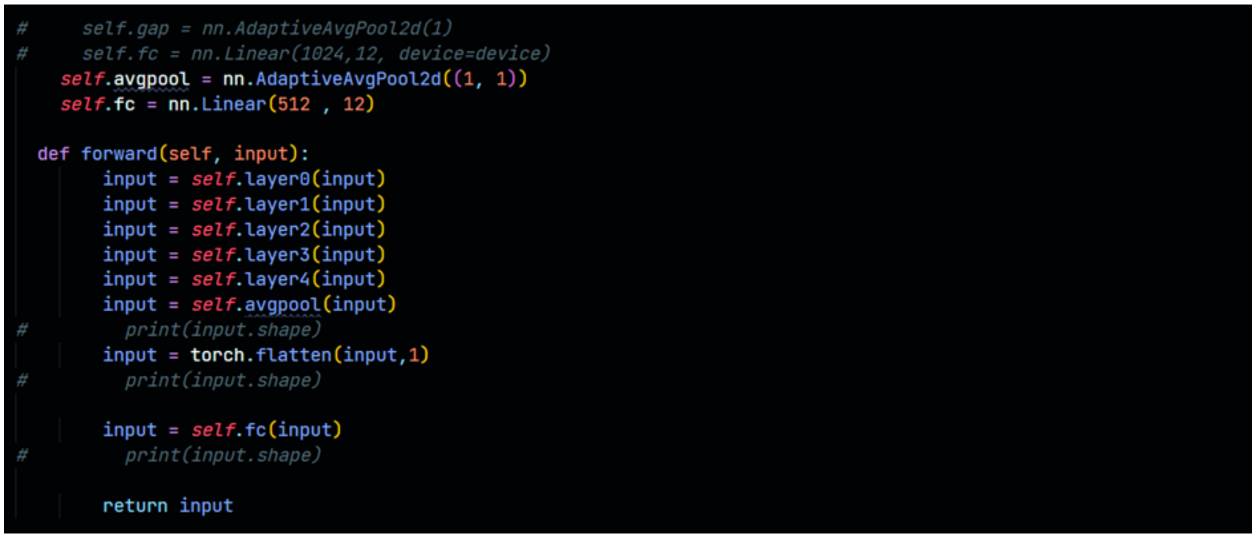
Model Selection and Training:

We have chosen three different models for the task:

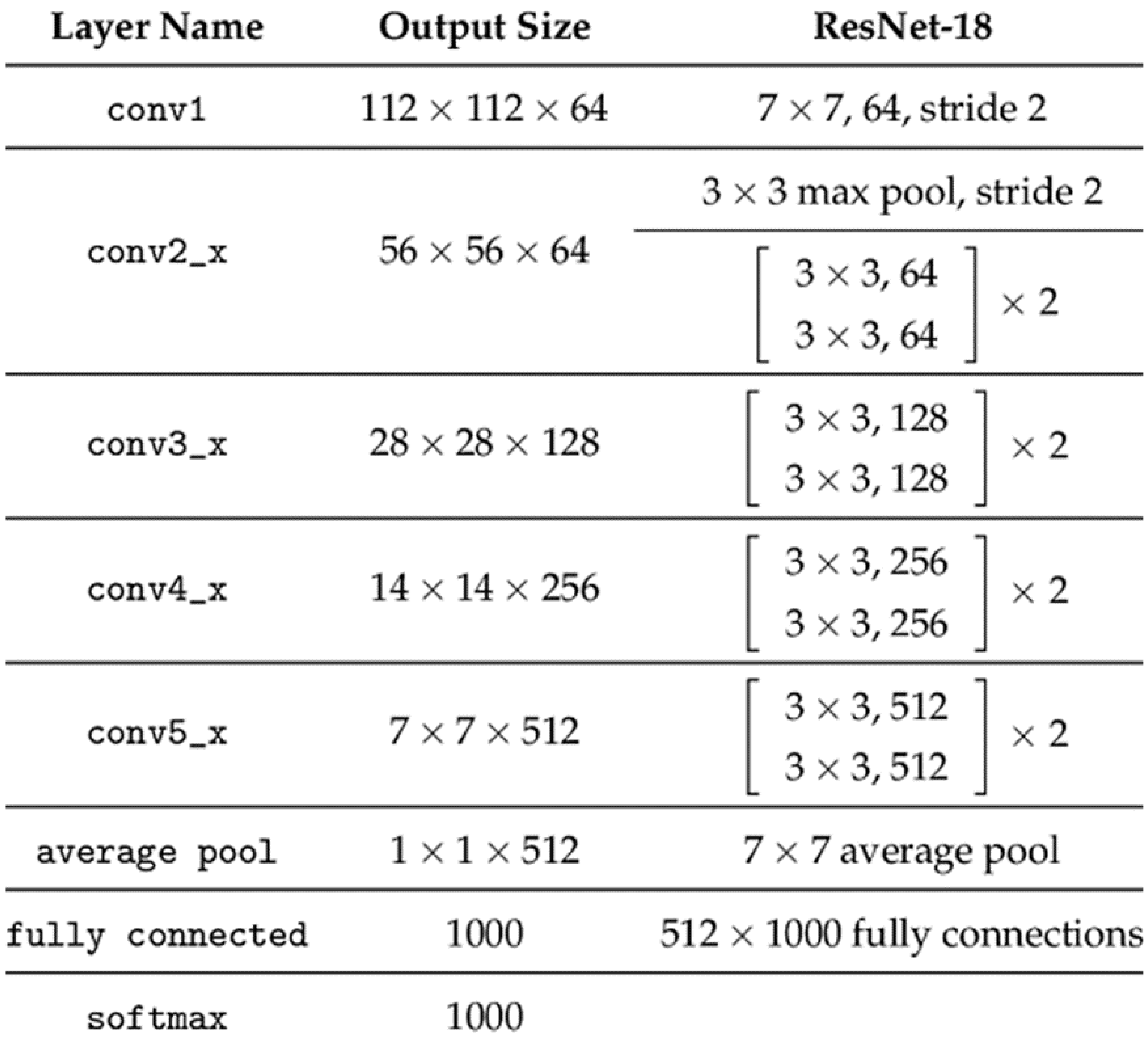
1) **ResNet-18:** A deep convolutional neural network with 18 layers that learns to extract features from images and make predictions based on those features. ResNet18 is an improvement on simple CNN models. Facing more complex problems, simply increasing the number of parameters does not solve the problem as it leads to the problem of exploding and vanishing gradients, leading to overfitting and a decline of performance beyond a certain point. To solve this, residual blocks are introduced in the architecture to learn the residual of the output.



```python
class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, downsample):
        super().__init__()
        if downsample:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=2, padding=1)
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=2),
                nn.BatchNorm2d(out_channels)
            )
        else:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1)
            self.shortcut = nn.Sequential()

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.bn2 = nn.BatchNorm2d(out_channels)

    def forward(self, input):
        shortcut = self.shortcut(input)
        input = nn.ReLU()(self.bn1(self.conv1(input)))
        input = nn.ReLU()(self.bn2(self.conv2(input)))
        input = input + shortcut
        return nn.ReLU()(input)
```

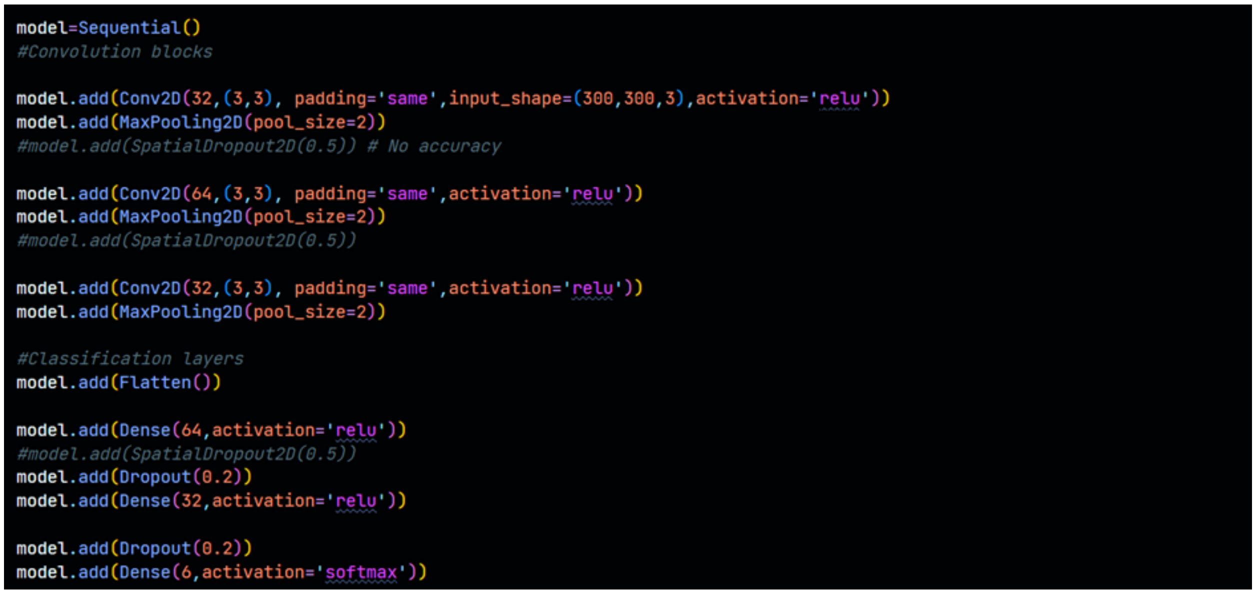


```python
#     self.gap = nn.AdaptiveAvgPool2d(1)
#     self.fc = nn.Linear(1024,12, device=device)
    self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
    self.fc = nn.Linear(512 , 12)

  def forward(self, input):
        input = self.layer0(input)
        input = self.layer1(input)
        input = self.layer2(input)
        input = self.layer3(input)
        input = self.layer4(input)
        input = self.avgpool(input)
#         print(input.shape)
        input = torch.flatten(input,1)
#         print(input.shape)

        input = self.fc(input)
#         print(input.shape)

        return input
```



Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

2) **Regular Convolutional Neural Network**: A custom-designed convolutional neural network tailored to your specific requirements. Feature extraction is facilitated by 2D convolution layers of 3X3 in size, reducing the parameter space and allowing for non-linear activation. In addition, the model is interleaved with max-pooling layers to reduce the parameter space and avoid overfitting while conserving critical features. The model structure can be seen in Figure 4. The fully connected layers learn the respective values for weights and biases using backpropagation, using gradient descent to minimize the loss function between the actual and predicted values.


```
model=Sequential()
#Convolution blocks

model.add(Conv2D(32,(3,3), padding='same',input_shape=(300,300,3),activation='relu'))
model.add(MaxPooling2D(pool_size=2))
#model.add(SpatialDropout2D(0.5)) # No accuracy

model.add(Conv2D(64,(3,3), padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))
#model.add(SpatialDropout2D(0.5))

model.add(Conv2D(32,(3,3), padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))

#Classification layers
model.add(Flatten())

model.add(Dense(64,activation='relu'))
#model.add(SpatialDropout2D(0.5))
model.add(Dropout(0.2))
model.add(Dense(32,activation='relu'))

model.add(Dropout(0.2))
model.add(Dense(6,activation='softmax'))
```

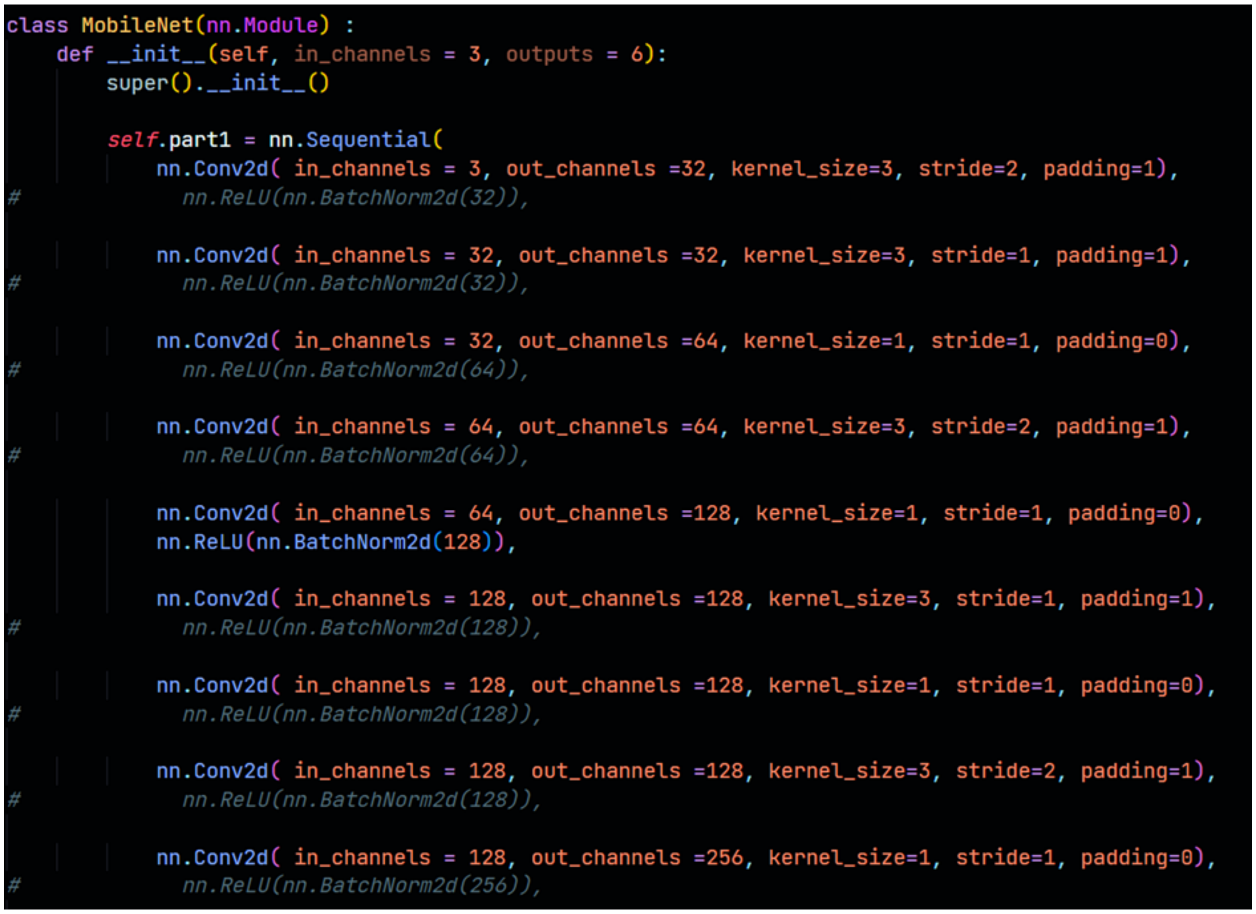
Layer Name	Output Shape	Param #
Conv2D	(None, 300, 300, 32)	896
MaxPooling2D	(None, 150, 150, 32)	0
Conv2D	(None, 150, 150, 64)	18,496
MaxPooling2D	(None, 75, 75, 64)	0
Conv2D	(None, 75, 75, 32)	18,464
MaxPooling2D	(None, 37, 37, 32)	0
Flatten	(None, 43808)	0
Dense	(None, 64)	2,803,776
Dropout	(None, 64)	0

The CNN was implemented according to the architecture specified above.

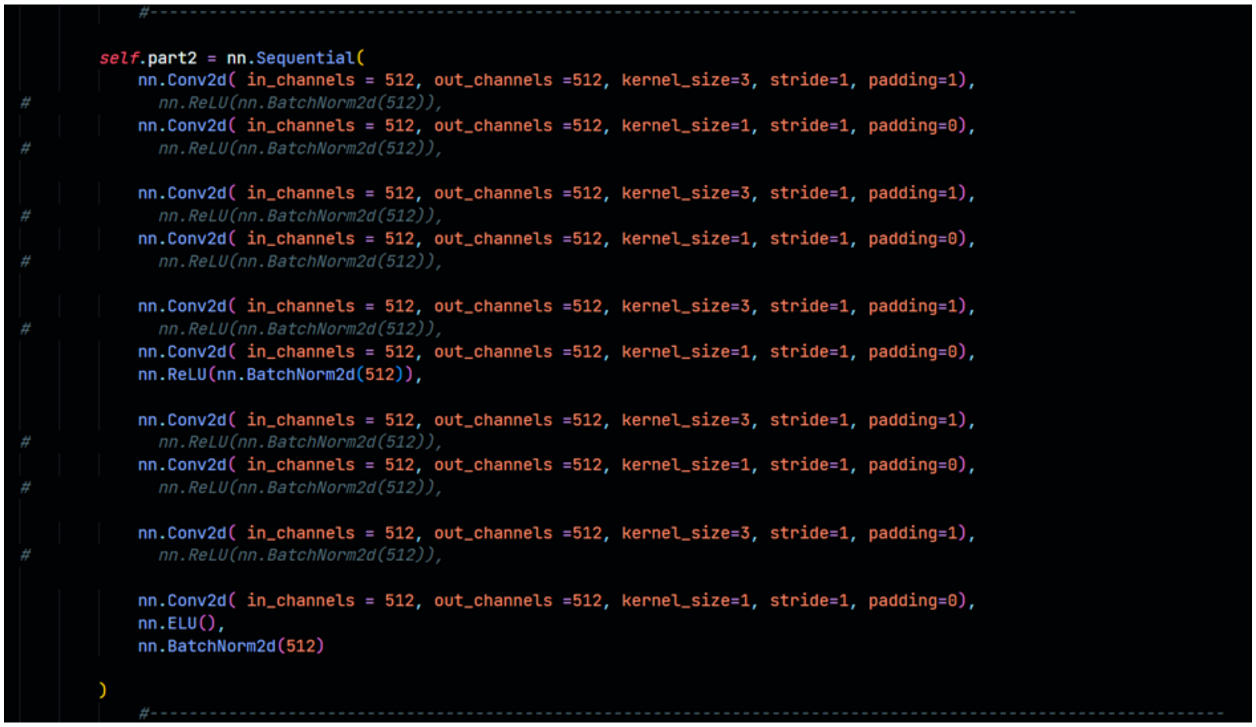
3) **MobileNet:** It is a convolution neural network architecture targeted at mobile devices. It is specially designed to work well with mobile devices while adhering to the computational constraints on such devices. It is founded on an inverse residual structure, connecting the bottleneck layers by residual connections. It uses lightweight depthwise convolutions as intermediate expansion layers to provide nonlinearity to the filter features.

Each selected model is trained on your preprocessed dataset. This training process involves the following steps:

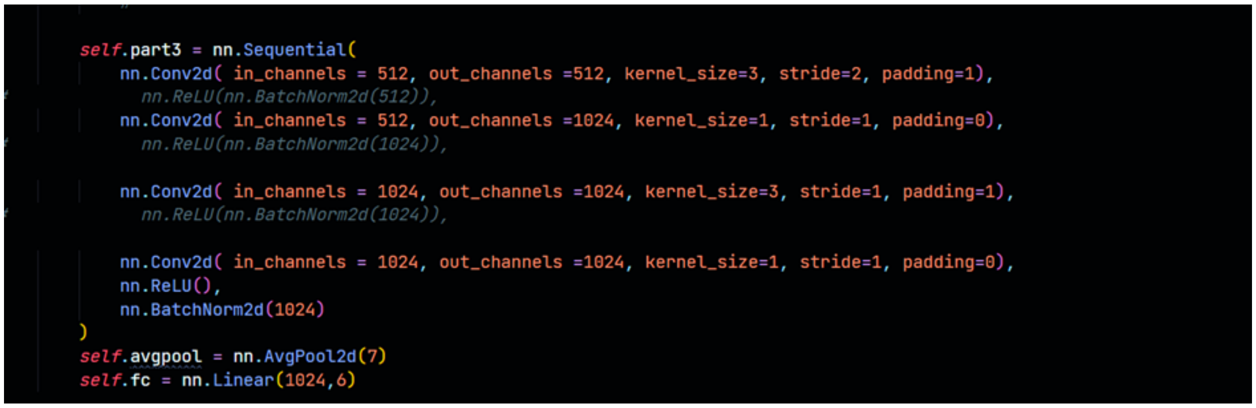
- **Initialization:** The model's weights and biases are initialized.
- **Forward Pass:** Each image is passed through the model to produce a prediction.
- **Loss Computation:** A loss function (typically categorical cross-entropy in classification tasks) measures the error between the model's prediction and the actual class labels.
- **Backpropagation:** The model's parameters are updated using gradient descent to minimize the loss.
- **Training Loop:** The above steps are repeated for a specified number of epochs, where the model learns to improve its predictions.



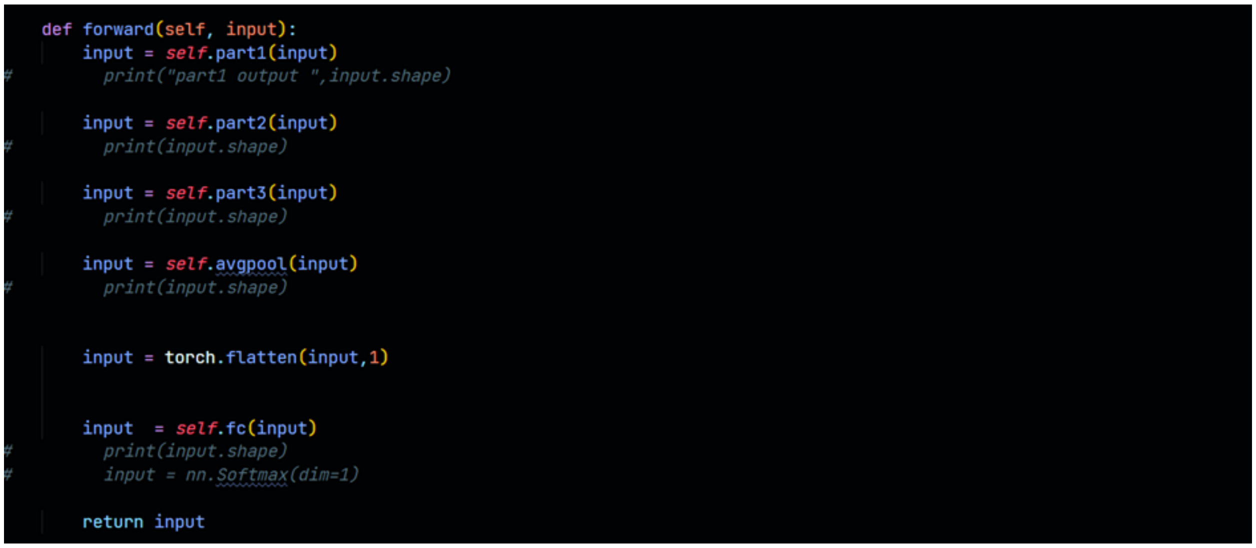
```
class MobileNet(nn.Module) :
    def __init__(self, in_channels = 3, outputs = 6):
        super().__init__()

        self.part1 = nn.Sequential(
            nn.Conv2d( in_channels = 3, out_channels =32, kernel_size=3, stride=2, padding=1),
#             nn.ReLU(nn.BatchNorm2d(32)),

            nn.Conv2d( in_channels = 32, out_channels =32, kernel_size=3, stride=1, padding=1),
#             nn.ReLU(nn.BatchNorm2d(32)),

            nn.Conv2d( in_channels = 32, out_channels =64, kernel_size=1, stride=1, padding=0),
#             nn.ReLU(nn.BatchNorm2d(64)),

            nn.Conv2d( in_channels = 64, out_channels =64, kernel_size=3, stride=2, padding=1),
#             nn.ReLU(nn.BatchNorm2d(64)),

            nn.Conv2d( in_channels = 64, out_channels =128, kernel_size=1, stride=1, padding=0),
            nn.ReLU(nn.BatchNorm2d(128)),

            nn.Conv2d( in_channels = 128, out_channels =128, kernel_size=3, stride=1, padding=1),
#             nn.ReLU(nn.BatchNorm2d(128)),

            nn.Conv2d( in_channels = 128, out_channels =128, kernel_size=1, stride=1, padding=0),
#             nn.ReLU(nn.BatchNorm2d(128)),

            nn.Conv2d( in_channels = 128, out_channels =128, kernel_size=3, stride=2, padding=1),
#             nn.ReLU(nn.BatchNorm2d(128)),

            nn.Conv2d( in_channels = 128, out_channels =256, kernel_size=1, stride=1, padding=0),
#             nn.ReLU(nn.BatchNorm2d(256)),
```



```
        #-----------------------------------------------------------------------------------------------

    self.part2 = nn.Sequential(
        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=3, stride=1, padding=1),
#         nn.ReLU(nn.BatchNorm2d(512)),
        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=1, stride=1, padding=0),
#         nn.ReLU(nn.BatchNorm2d(512)),

        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=3, stride=1, padding=1),
#         nn.ReLU(nn.BatchNorm2d(512)),
        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=1, stride=1, padding=0),
#         nn.ReLU(nn.BatchNorm2d(512)),

        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=3, stride=1, padding=1),
#         nn.ReLU(nn.BatchNorm2d(512)),
        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=1, stride=1, padding=0),
        nn.ReLU(nn.BatchNorm2d(512)),

        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=3, stride=1, padding=1),
#         nn.ReLU(nn.BatchNorm2d(512)),
        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=1, stride=1, padding=0),
#         nn.ReLU(nn.BatchNorm2d(512)),

        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=3, stride=1, padding=1),
#         nn.ReLU(nn.BatchNorm2d(512)),

        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=1, stride=1, padding=0),
        nn.ELU(),
        nn.BatchNorm2d(512)

    )
        #-----------------------------------------------------------------------------------------------------------
```

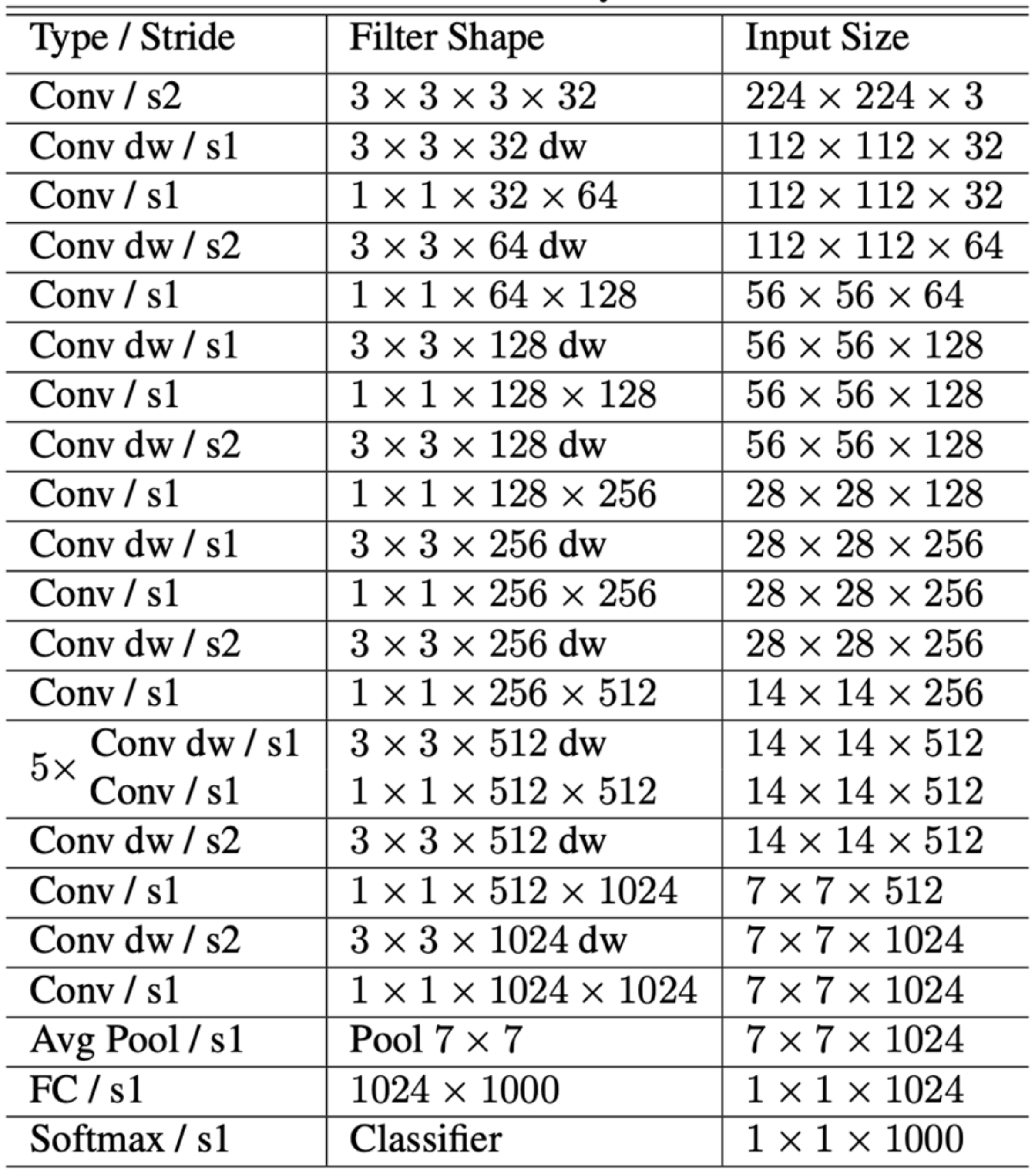


```
    self.part3 = nn.Sequential(
        nn.Conv2d( in_channels = 512, out_channels =512, kernel_size=3, stride=2, padding=1),
#         nn.ReLU(nn.BatchNorm2d(512)),
        nn.Conv2d( in_channels = 512, out_channels =1024, kernel_size=1, stride=1, padding=0),
#         nn.ReLU(nn.BatchNorm2d(1024)),

        nn.Conv2d( in_channels = 1024, out_channels =1024, kernel_size=3, stride=1, padding=1),
#         nn.ReLU(nn.BatchNorm2d(1024)),

        nn.Conv2d( in_channels = 1024, out_channels =1024, kernel_size=1, stride=1, padding=0),
        nn.ReLU(),
        nn.BatchNorm2d(1024)
    )
    self.avgpool = nn.AvgPool2d(7)
    self.fc = nn.Linear(1024,6)
```



```
def forward(self, input):
    input = self.part1(input)
#       print("part1 output ",input.shape)

    input = self.part2(input)
#       print(input.shape)

    input = self.part3(input)
#       print(input.shape)

    input = self.avgpool(input)
#       print(input.shape)


    input = torch.flatten(input,1)


    input  = self.fc(input)
#       print(input.shape)
#       input = nn.Softmax(dim=1)

    return input
```

Table 1. MobileNet Body Architecture



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
5× Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

The MobileNet network was implemented in the standard architecture as specified above.

GithubLink:
https://github.com/karthiksenthil2803/CS6005-DL-Project-Garbage-Classification

Model Evaluation and comparison:

In all cases the dataset was split into train and validation datasets. All the three models were trained up to 50 epochs and their performance at the end of 50 epochs are taken into consideration.

The loss functions and optimizers used are :

Model	Loss Functions	Optimizers
ResNet18	Cross Entropy	Stochastic Gradient Descent
MobileNetV1	Cross Entropy	Adam
CNN	Categorical Cross Entropy	Adam

The cross-Entropy Loss:

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

The Adam Optimizer:

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

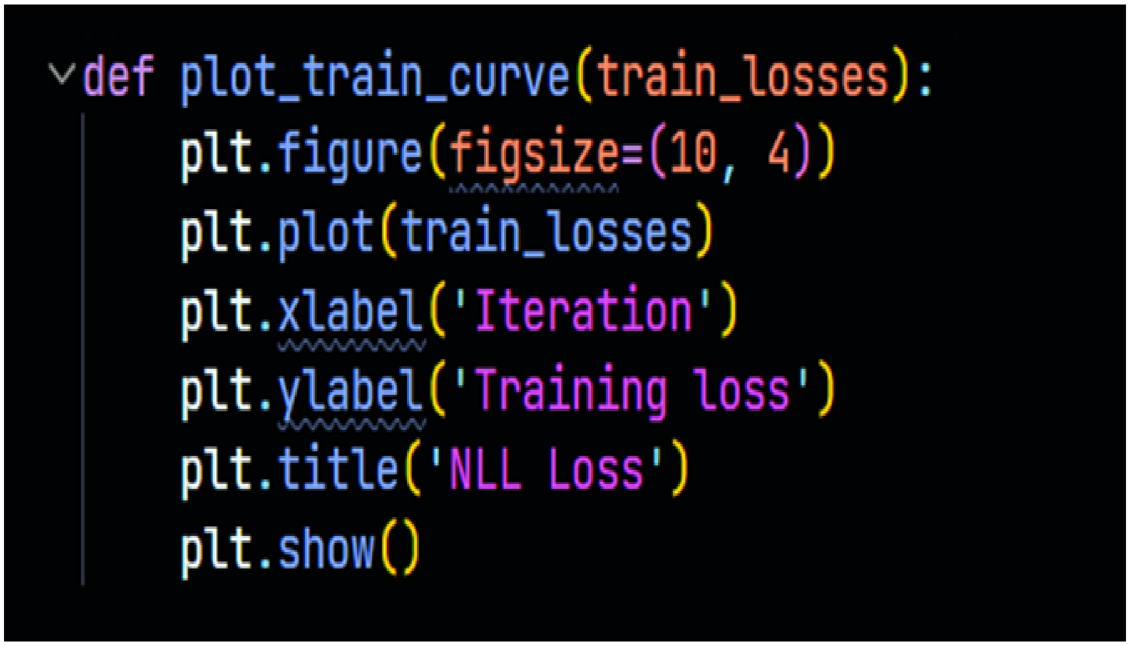
The Stochastic Gradient Descent:

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

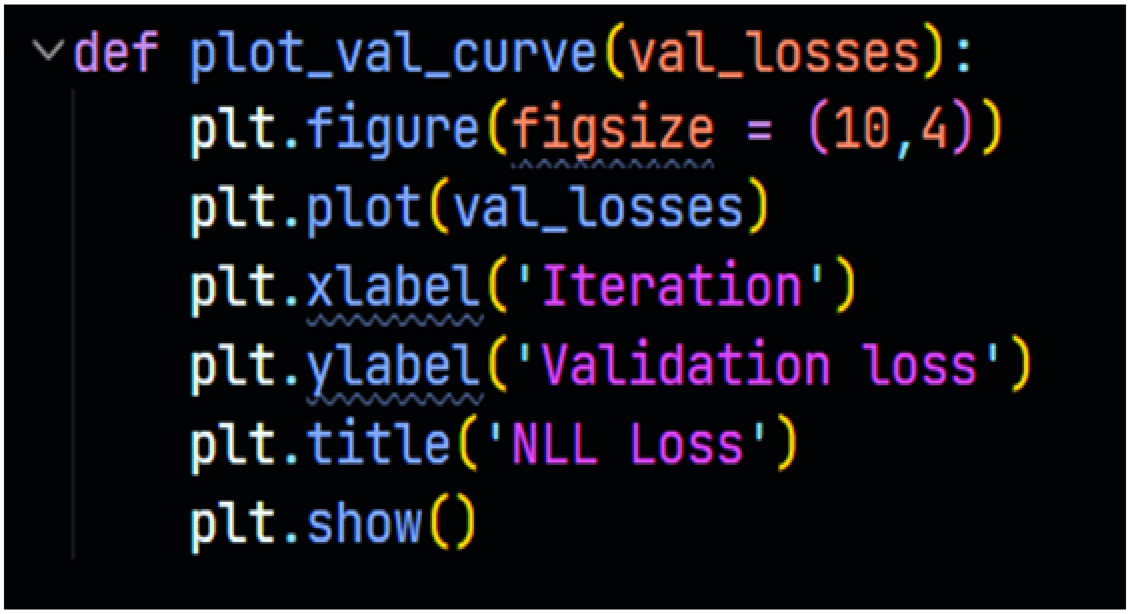
```
Require: Learning rate $\epsilon_k$.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while
```

Training and Validation accuracy:

At the end of training, the loss and accuracy obtained during training and validation are plotted as a graph.



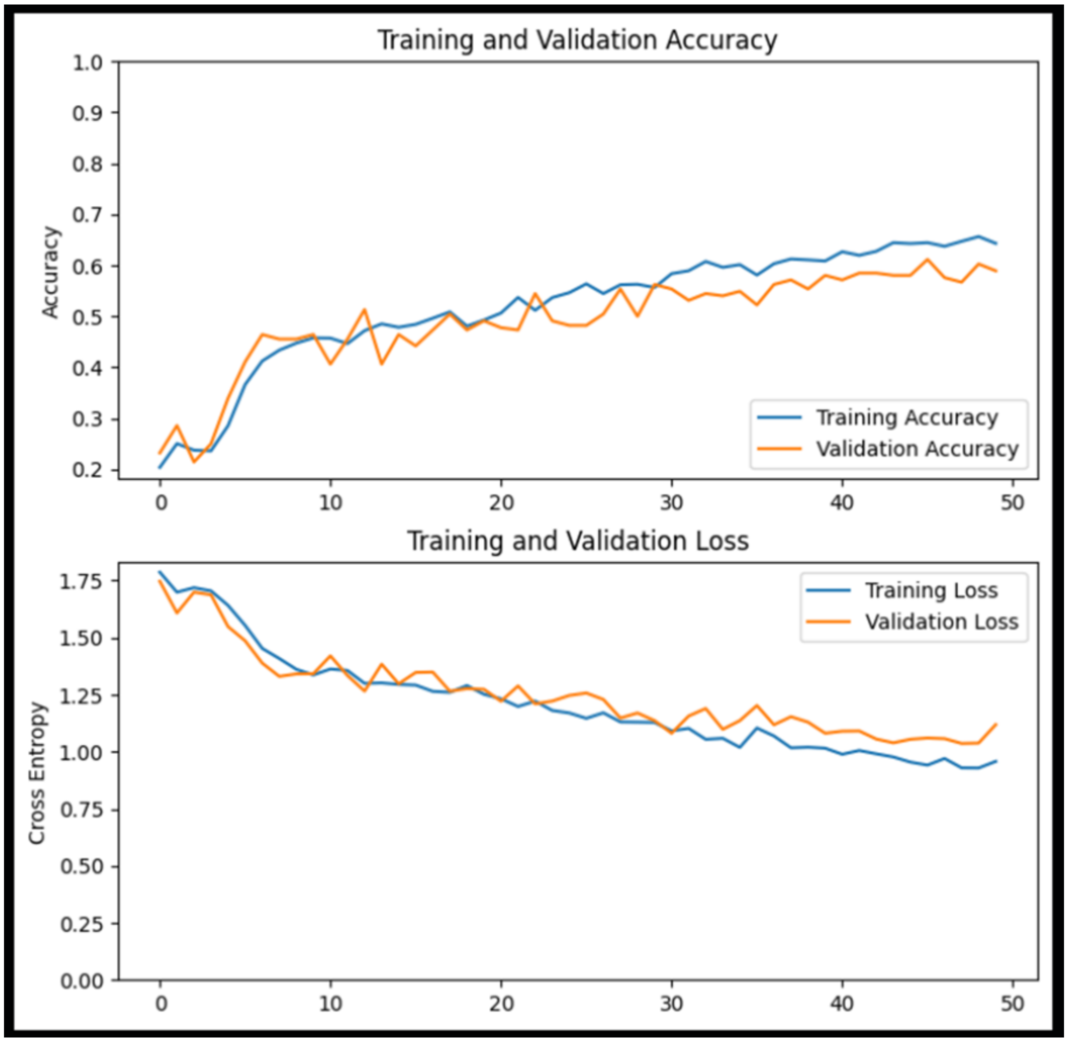
```python
def plot_train_curve(train_losses):
    plt.figure(figsize=(10, 4))
    plt.plot(train_losses)
    plt.xlabel('Iteration')
    plt.ylabel('Training loss')
    plt.title('NLL Loss')
    plt.show()
```



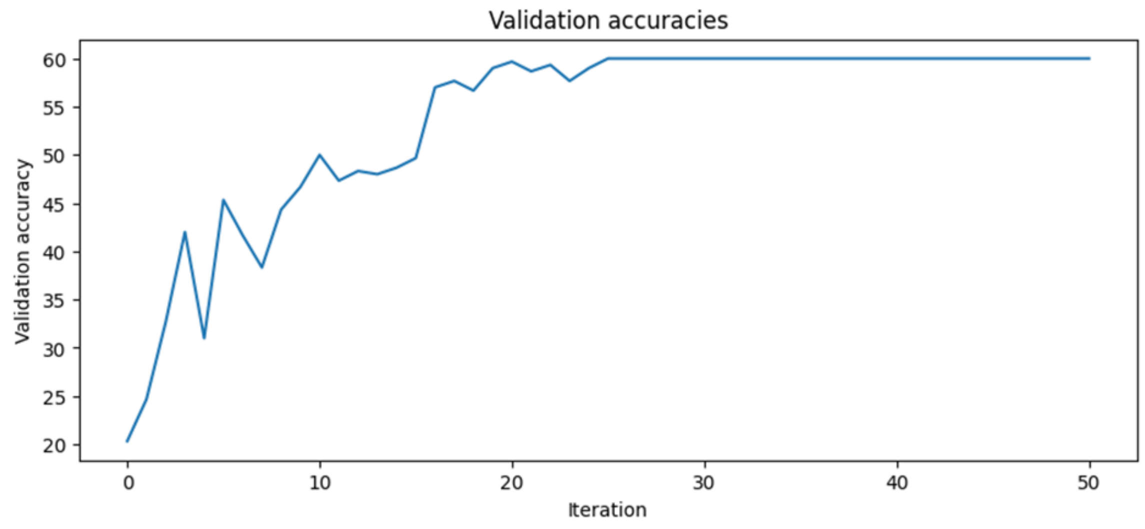
```python
def plot_val_curve(val_losses):
    plt.figure(figsize = (10,4))
    plt.plot(val_losses)
    plt.xlabel('Iteration')
    plt.ylabel('Validation loss')
    plt.title('NLL Loss')
    plt.show()
```

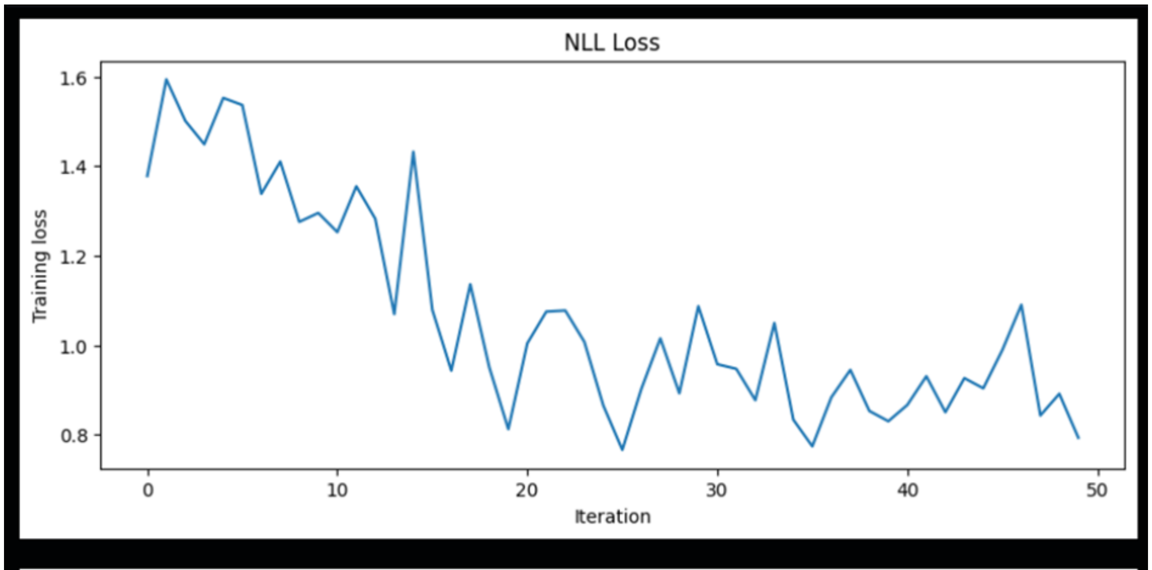
Results and Graphs:

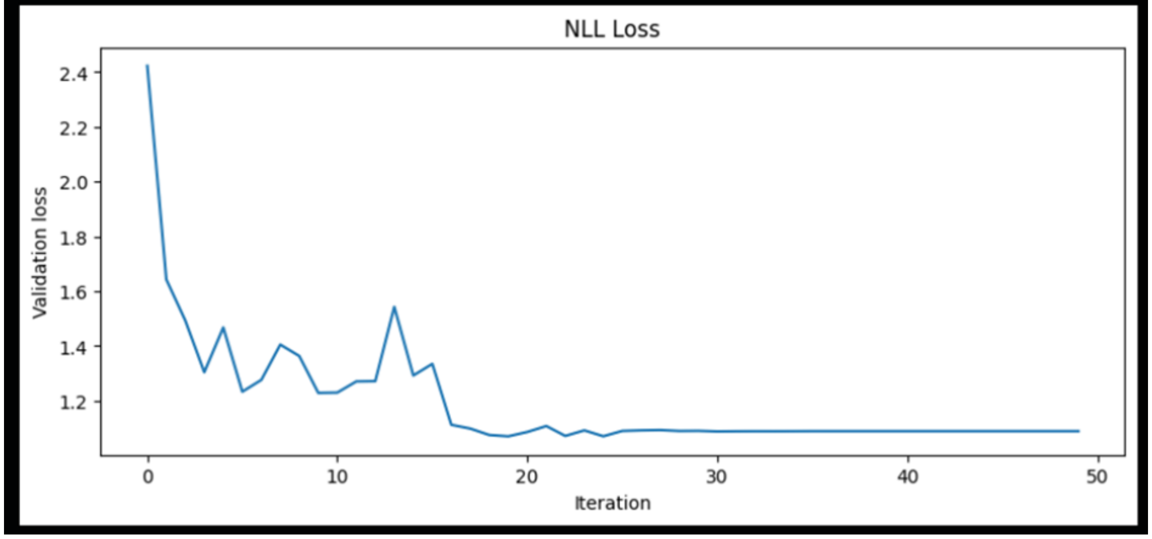
Results of CNN:



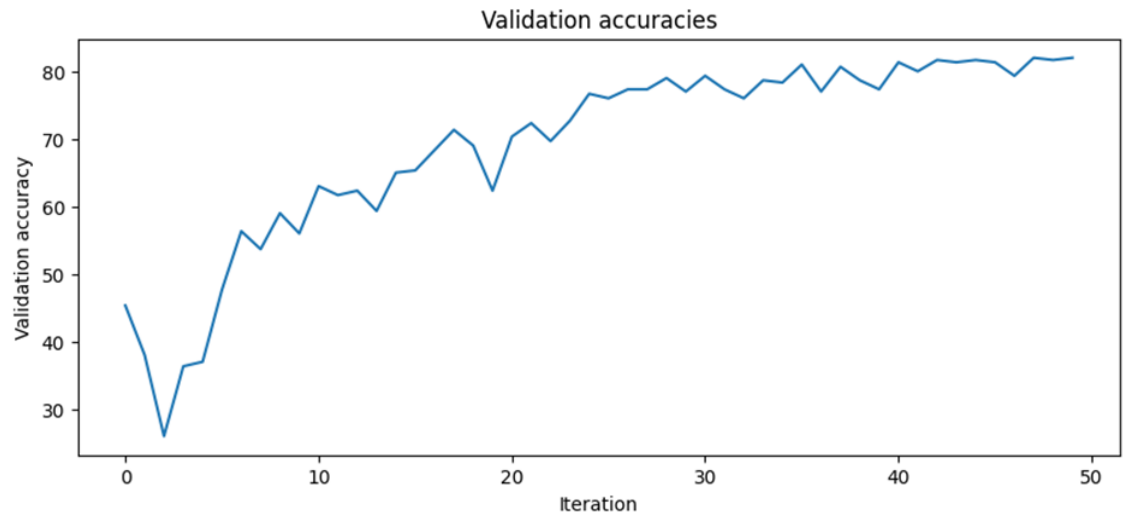
Results of MobileNet1V1:

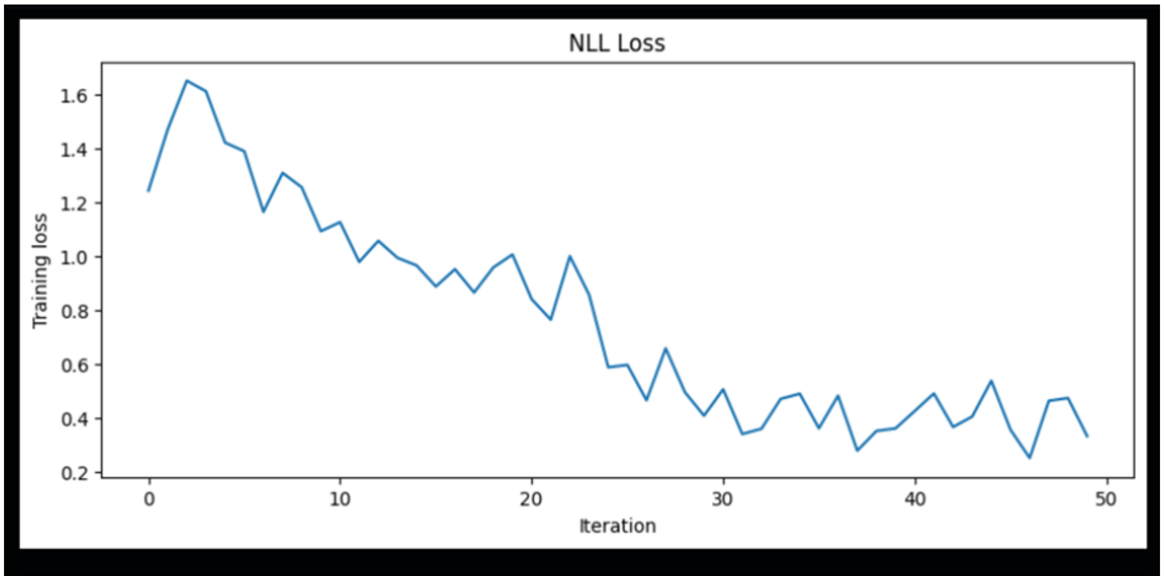


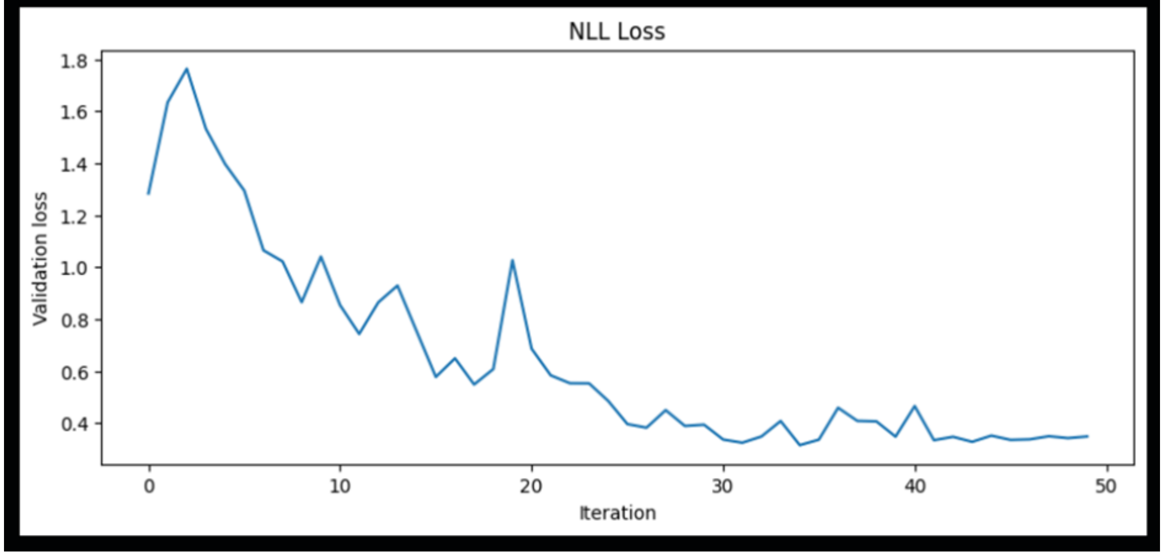




Results of ResNet18:







Test Accuracy :

CNN model: 58.93%

ResNet18 Model: 82%

MobileNetV1: 60%

Evaluation Metrics with Justification:

Accuracy: The classification accuracy is the ratio of the number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Correct\ prediction}{Total\ cases} * 100\%$$

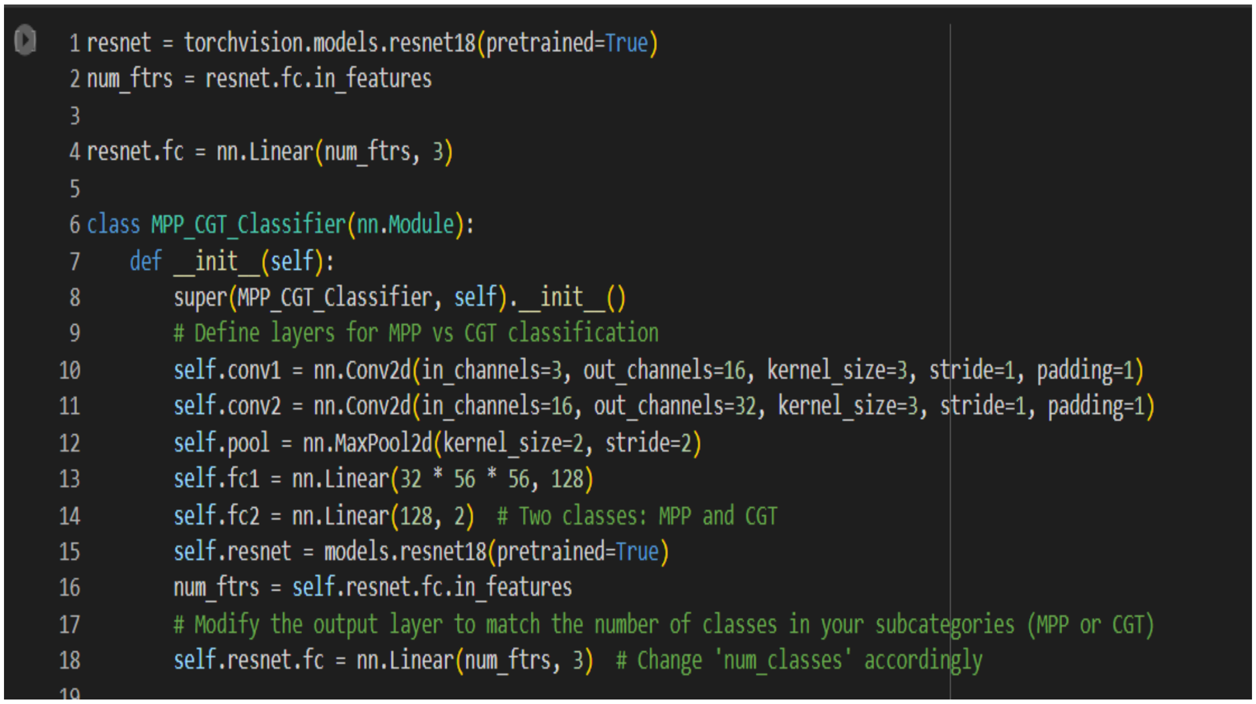
$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} * 100\%$$

Accuracy is suitable to be chosen as the only measure of classification for the problem of garbage classification since we are classifying images consisting of a single type of trash in one image, therefore more complicated evaluation metrics are not necessary, for the simple case of a single type of garbage being classified at a time, accuracy is a good enough evaluation metric for the same.

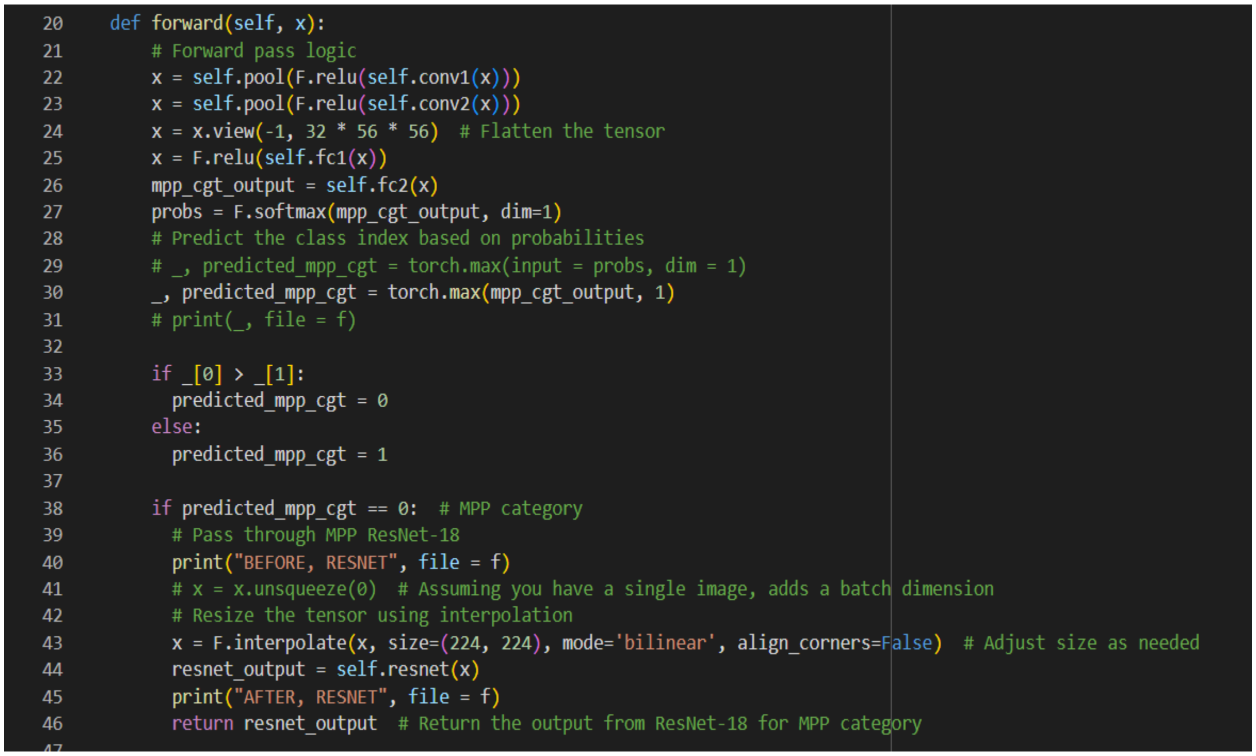
Based on the performance of models ResNet18 outperforms others according to chosen evaluation metrics with an accuracy of 82% is selected as best model for waste classification.

Hybrid Model:

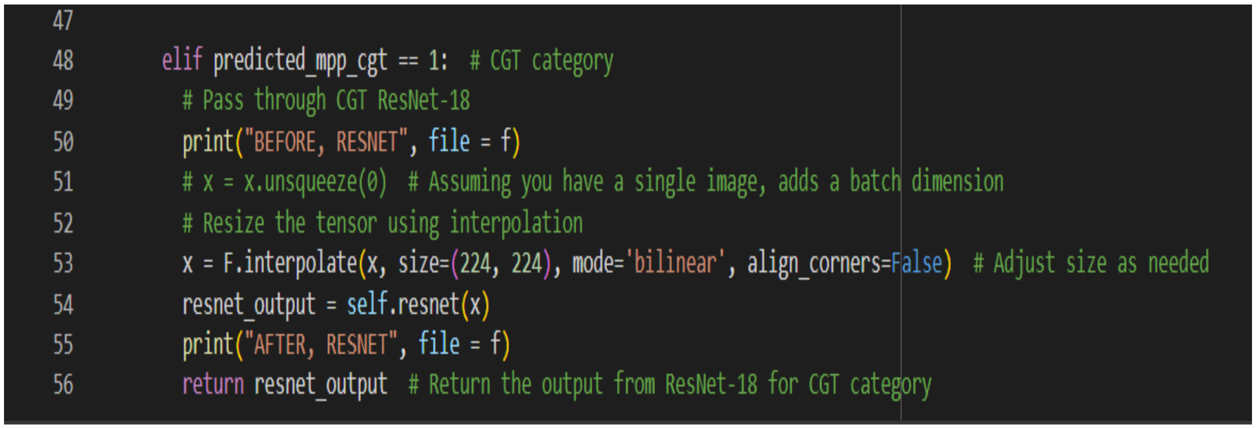
We tried to blend the ResNet-50, Regular CNN, and MobileNet models into a single hybrid model. However, the task's complexity and limited resources posed challenges in seamlessly integrating these diverse architectures.



```
resnet = torchvision.models.resnet18(pretrained=True)
num_ftrs = resnet.fc.in_features

resnet.fc = nn.Linear(num_ftrs, 3)

class MPP_CGT_Classifier(nn.Module):
    def __init__(self):
        super(MPP_CGT_Classifier, self).__init__()
        # Define layers for MPP vs CGT classification
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(32 * 56 * 56, 128)
        self.fc2 = nn.Linear(128, 2)  # Two classes: MPP and CGT
        self.resnet = models.resnet18(pretrained=True)
        num_ftrs = self.resnet.fc.in_features
        # Modify the output layer to match the number of classes in your subcategories (MPP or CGT)
        self.resnet.fc = nn.Linear(num_ftrs, 3)  # Change 'num_classes' accordingly
```



```
    def forward(self, x):
        # Forward pass logic
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 32 * 56 * 56)  # Flatten the tensor
        x = F.relu(self.fc1(x))
        mpp_cgt_output = self.fc2(x)
        probs = F.softmax(mpp_cgt_output, dim=1)
        # Predict the class index based on probabilities
        # _, predicted_mpp_cgt = torch.max(input = probs, dim = 1)
        _, predicted_mpp_cgt = torch.max(mpp_cgt_output, 1)
        # print(_, file = f)

        if _[0] > _[1]:
          predicted_mpp_cgt = 0
        else:
          predicted_mpp_cgt = 1

        if predicted_mpp_cgt == 0:  # MPP category
          # Pass through MPP ResNet-18
          print("BEFORE, RESNET", file = f)
          # x = x.unsqueeze(0)  # Assuming you have a single image, adds a batch dimension
          # Resize the tensor using interpolation
          x = F.interpolate(x, size=(224, 224), mode='bilinear', align_corners=False)  # Adjust size as needed
          resnet_output = self.resnet(x)
          print("AFTER, RESNET", file = f)
          return resnet_output  # Return the output from ResNet-18 for MPP category
```



```
    elif predicted_mpp_cgt == 1:  # CGT category
      # Pass through CGT ResNet-18
      print("BEFORE, RESNET", file = f)
      # x = x.unsqueeze(0)  # Assuming you have a single image, adds a batch dimension
      # Resize the tensor using interpolation
      x = F.interpolate(x, size=(224, 224), mode='bilinear', align_corners=False)  # Adjust size as needed
      resnet_output = self.resnet(x)
      print("AFTER, RESNET", file = f)
      return resnet_output  # Return the output from ResNet-18 for CGT category
```

Conclusion:

Based on the evaluation metrics, ResNet18 stands out as the best-performing model for waste classification, achieving an accuracy of 82%. This indicates its superior ability to recognize and classify different types of trash in the given dataset. The successful implementation of the Integrated Waste Management System, leveraging advanced technologies like deep learning and computer vision, showcases its potential to revolutionize waste management practices, making them more efficient, cost-effective, and environmentally friendly. The adoption of such automated solutions not only safeguards the environment and public health but also aligns with long-term sustainability goals.