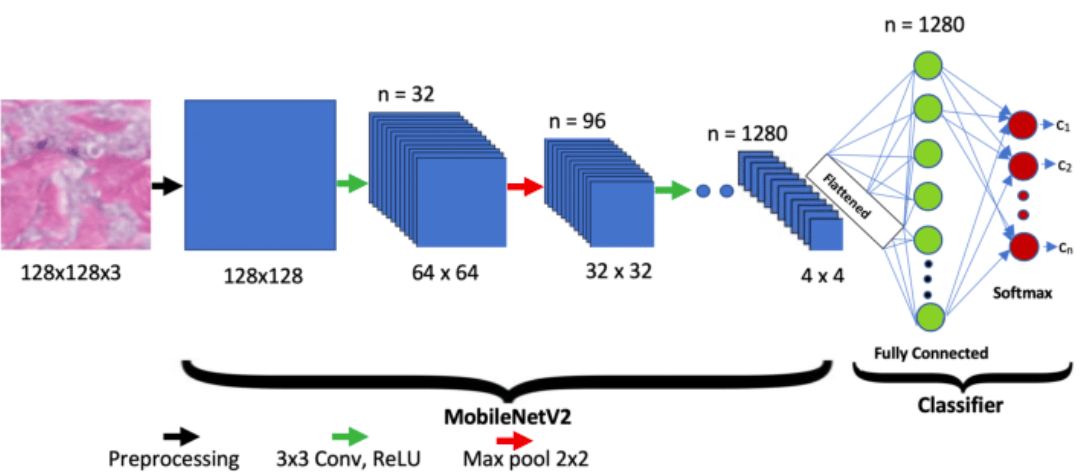
**RealWaste Image Classification with Transfer Learning**

This code implements the MobileNet\_V2 model for classifying images using transfer learning and fine-tuning with Keras.

**Objective:**

The objective of this project is to replicate the results obtained by pre-trained CNN models like MobileNet\_V2 using transfer learning.



# Fig 1. MobileNet\_V2 Architecture

**Data Preparation:**

* The dataset is divided into ‘train’ and ‘test’ dataset.
* The dataset contains 4752 images belonging to different categories.
* image\_size = (224,224): Resizes images to 224x224 pixels.
* shuffle=True: Randomises the order of images in the ‘train’ and ‘test’ datasets.
* seed=123: Fixes the images that will be in the two datasets.
* validation\_split=0.2: Divides the original dataset into ‘train’ and ‘test’ in the ratio 80:20.

**MobileNet\_V2 – Inverted Residuals and Linear Bottlenecks:**

Linear Bottlenecks in MobileNetV2

MobileNetV2 introduced the concept of "linear bottlenecks." Here’s how it works:

1. Expand: First, the network expands the data to a higher dimension using a 1x1 convolution. This means it increases the number of features (or channels) but keeps the spatial dimensions the same.
2. Depthwise Convolution: Then, a depthwise convolution is applied. This type of convolution filters each channel independently, which is computationally cheaper than traditional convolutions that mix channels.
3. Linear Bottleneck: Finally, the network reduces the data back to a smaller dimension using another 1x1 convolution. Importantly, this last step uses a linear activation (no non-linearity like ReLU is applied here).

### Inverted Residuals in MobileNetV2

**Inverted residuals** are another key concept introduced in the MobileNetV2 architecture. They are used to improve efficiency and performance of the network, particularly on mobile and edge devices. Let's break down what inverted residuals are and how they work.

#### Traditional Residuals

In traditional residual networks (ResNets), the basic idea is to add shortcut connections that skip one or more layers. This helps with training deep networks by allowing the gradient to flow through these shortcuts directly, which mitigates the vanishing gradient problem.

A typical residual block in ResNet looks like this:

1. Input
2. Convolutional layers (with non-linear activations like ReLU)
3. Add the input to the output of the convolutional layers (shortcut connection)
4. Output

A diagram of a block diagram

Description automatically generated

# Fig 2. Structure of MobileNet\_V2

* The architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers.
* ReLU6 as the non-linearity because of its robustness when used with low-precision computation.
* Always use kernel size 3 × 3 as is standard for modern networks and utilize dropout and batch normalization during training.
* Except for the first layer, we use constant expansion rate throughout the network. We find that expansion rates between 5 and 10 result in nearly identical performance curves, with smaller networks being better off with slightly smaller expansion rates and larger networks having slightly better performance with larger expansion rates.

**Additional Changes:**

* All the layers in MobileNet\_V2 are rendered “trainable”.
* Flatten() layer is added along with BatchNorm
* A Dropout of 0.2 is added to droupot 20% of nodes during training
* Two iterations of Dense layer (ReLU activation) + BatchNorm layer are added
* And lastly a Dense layer (Softmax Activation) is added to obtain the output as a single integer indicating the class of input image.

**Compilation:**

* The model is then compiled with:
  1. **‘sparse\_categorical\_crossentropy’**: Suitable for multi-class classification with integer labels.
  2. Adam optimizer with a low learning rate to fine-tune the pre-trained layers.
  3. **metrics=[‘Accuracy’]**: Monitors training and validation accuracy.
* The model is trained using the fit function with:

○ **train** dataset for training.

* Epochs = 25

○ **validation\_data=(test)**: Evaluates the model on the validation set after each epoch.

# Training Performance Visualization

* The code plots the accuracy and loss curves using the data of trained model.
* These plots help to visualise how the model performs during the training phase and helps to identify overfitting and underfitting issues.

# Part 1 - On-Shelf Training

**Results:**

● The model achieves about 78% in on-shelf transfer learning.

# Confusion Matrix

* The code iterates over the test data to collect predicted labels for each image.
* The confusion matrix visualizes the distribution of correct and incorrect predictions for each waste category.
* This helps identify classes that the model struggles with and allows for further analysis or targeted improvements.
* The confusion matrix helps to calculate the precision and recall values for each class and ultimately the f1 score. The f1 score indicates the performance of the model better than the validation accuracy. The f1 score is effective when the dataset is skewed i.e. the no. of training examples are not uniform for each class.

A diagram of a true confusion matrix

Description automatically generated with medium confidence

# Experiment No. 2 - Fine-Tuning

**Results:**

* Further, the accuracy increases to 83.79% using fine-tuning.

A graph of a graph

Description automatically generated

**Fig 6. Model accuracy curve for fine-tuned training**

* The model achieves training accuracy of 100% within 5 epochs with a learning rate of 2e-4.
* The validation accuracy increases gradually throughout the training from

40% to 83.79%.

● The validation loss remains almost constant at around 0.4 while the training loss decreases to as low as 4.83e-4.

# Confusion Matrix

A blue and white chart

Description automatically generated with medium confidence

# Overall

This code demonstrates a common approach for real-waste image classification using transfer learning and fine-tuning. It leverages a pre-trained InceptionV3 model and fine-tunes its final layers for the specific waste classification task. The code also includes data preparation, model compilation, training, performance visualization, and evaluation using a confusion matrix.