Project Overview

Task:

Classify raw materials (e.g., metals, wood, plastic) from images for industrial sorting and recycling.

Dataset:

Recyclable and Household Waste Classification

Architectures to Apply:

1. **ResNet** - Implemented from scratch.
2. **Xception** - Applied as a pre-trained model using Transfer Learning.
3. **DenseNet** - Applied as a pre-trained model using Transfer Learning.

Preprocessing the Dataset

1. Understanding the Dataset

- What is the Dataset?
  It contains images of raw materials like metals, wood, and plastics. These materials may be photographed in various environments, angles, and lighting conditions.
- Purpose:
  To classify materials for industrial sorting and recycling.

Step 1: Collect the Data

- Source:
  - Use publicly available datasets like those on Kaggle, ImageNet, or other repositories.
- Include a variety of materials (e.g., different types of metals, various types of wood, plastics like PET or HDPE).
- Labeling:
  - Label each image accurately (e.g., "metal," "wood," or "plastic").

Step 2: Organize the Dataset

- Folder Structure:
  Create a structured hierarchy to store images:
- Split the Dataset:
  Divide the dataset into:
- Training Set (80%): Used for training the models.
- Validation Set (10%): Used for tuning and preventing overfitting.
- Test Set (10%): Used for final evaluation.

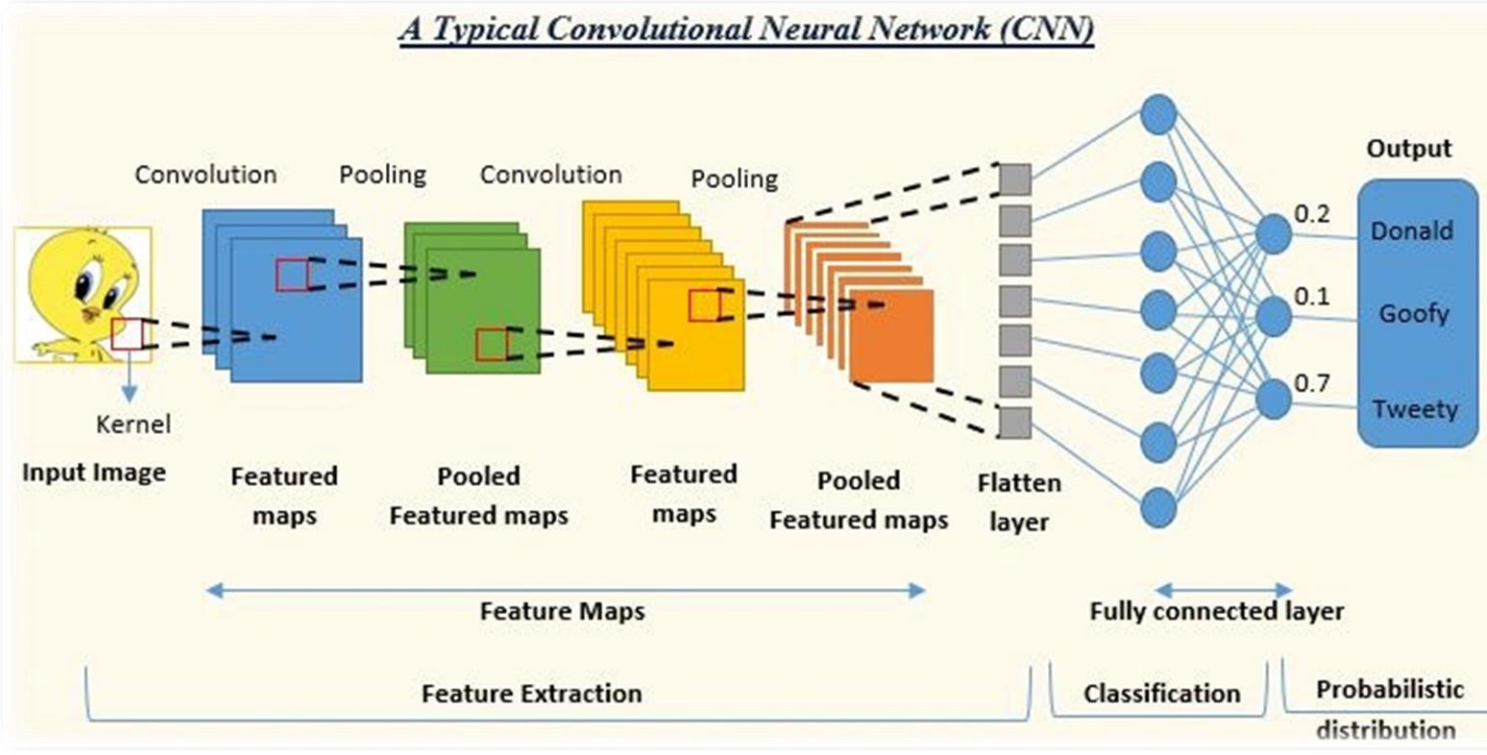
Step 3: Image Preparation

- Resize Images:
- Convert all images to a uniform size (e.g., 224x224 or 299x299 pixels).
- Normalize Images:
    - Scale pixel values to a standard range (e.g., 0 to 1). This improves model training stability.

Step 4: Ensure Dataset Quality

- Check for Imbalances:
    - Ensure that each class (metal, wood, plastic) has a similar number of images.
- Class Diversity:
    - For each class, include variations:
    - Metals: Include aluminum, steel, and copper.
    - Wood: Include softwood, hardwood, and processed wood

CNN



1. Input Layer

- Purpose: The input layer receives raw data, such as images, text, or tabular data, and serves as the entry point for the model.
- Characteristics:
- For image data: Input is usually a tensor of shape (height, width, channels), e.g., (224, 224, 3) for RGB images.
- For tabular data: Input is a vector of numerical features.
- For sequential data: Input is typically a sequence of vectors (e.g., words or time-series data).

---

2. Hidden Layers

- Hidden layers process the input data through transformations, extracting and learning patterns. These layers may include various types, each with a unique purpose.

2.1 Dense (Fully Connected) Layers

- Purpose:
- Transform data into higher-dimensional spaces.
- Combine and interpret learned features.

2.2 Convolutional Layers

- Purpose:
- Extract spatial features from images (e.g., edges, shapes, textures).
- Operation:
  Apply filters (kernels) to small regions of the input, sliding across the height and width to produce feature maps.

2.3 Pooling Layers

- Purpose:

    - Reduce the spatial dimensions of feature maps (height and width) while retaining key information.
    - Prevent overfitting and reduce computational cost.
- Types:
    - Max Pooling: Retains the maximum value in each region.
    - Average Pooling: Computes the average value in each region.

2.4 Activation Layers

- Purpose:
  Introduce non-linearity into the network, enabling it to learn complex patterns.
- Common Activations:
    - ReLU (Rectified Linear Unit): Helps handle vanishing gradients.
    - Sigmoid: Outputs values between 0 and 1. Useful for binary classification.
    - Softmax: Converts raw scores into probabilities, used in multi-class classification.

2.5 Dropout Layers

- Purpose:
    - Prevent overfitting by randomly "dropping out" (setting to 0) a fraction of neurons during training.
- How It Works:
    - A dropout rate (e.g., 0.5) specifies the proportion of neurons to drop.

2.6 Batch Normalization Layers

- Purpose:

    - Normalize inputs to each layer, stabilizing and accelerating training.
- Operation:
  Adjusts activations to have zero mean and unit variance, then applies a learned scaling and shifting.

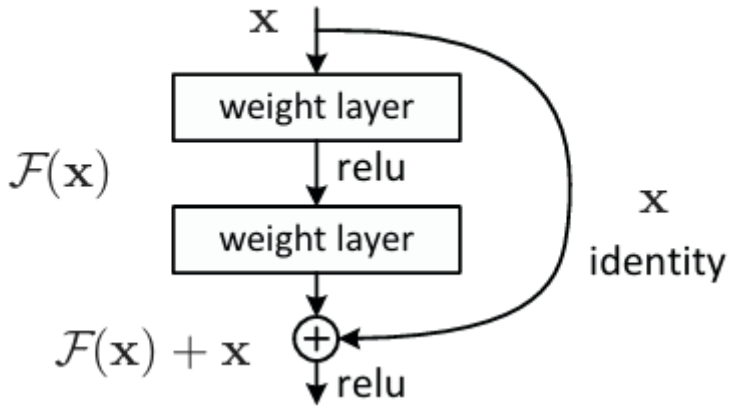
---

3. Output Layer

- Purpose:
  Produces the final predictions based on the processed data.
- Design:
- For binary classification: Single neuron with a sigmoid activation.
- For multi-class classification: Neurons equal to the number of classes, with a softmax activation.
- For regression tasks: Single neuron with no activation (linear output).

Models

ResNet (Residual Network)



What is ResNet?

ResNet addresses the challenge of **training very deep neural networks** by introducing **skip (residual) connections**. These connections allow the model to bypass some layers, solving the problems of **vanishing gradients** and **degradation** in deep networks.

Key Innovation

- **Skip Connections (Residual Learning):**
  Instead of learning the mapping, ResNet learns the **residual function**.
- These skip connections allow the gradients to flow **easily** during backpropagation, even in very deep networks (e.g., 50 or 100+ layers).

Architecture

- **Convolutional Blocks:**
  - A block typically contains:
    - **Convolutional layers** (3x3 filters).
    - **Batch Normalization (BN)** for stable learning.
    - **ReLU activation** for non-linearity.

    - Skip connections are added to "skip" these blocks.
- **Types of ResNet Blocks:**
    - **Identity Block:** Input and output dimensions are the same.
    - **Convolutional Block:** Input and output dimensions differ (used when downsampling).
- **Variants:**
    - **ResNet-18/34:** Uses basic residual blocks.
    - **ResNet-50/101/152:** Uses bottleneck blocks (1x1 → 3x3 → 1x1 convolutions for efficiency).

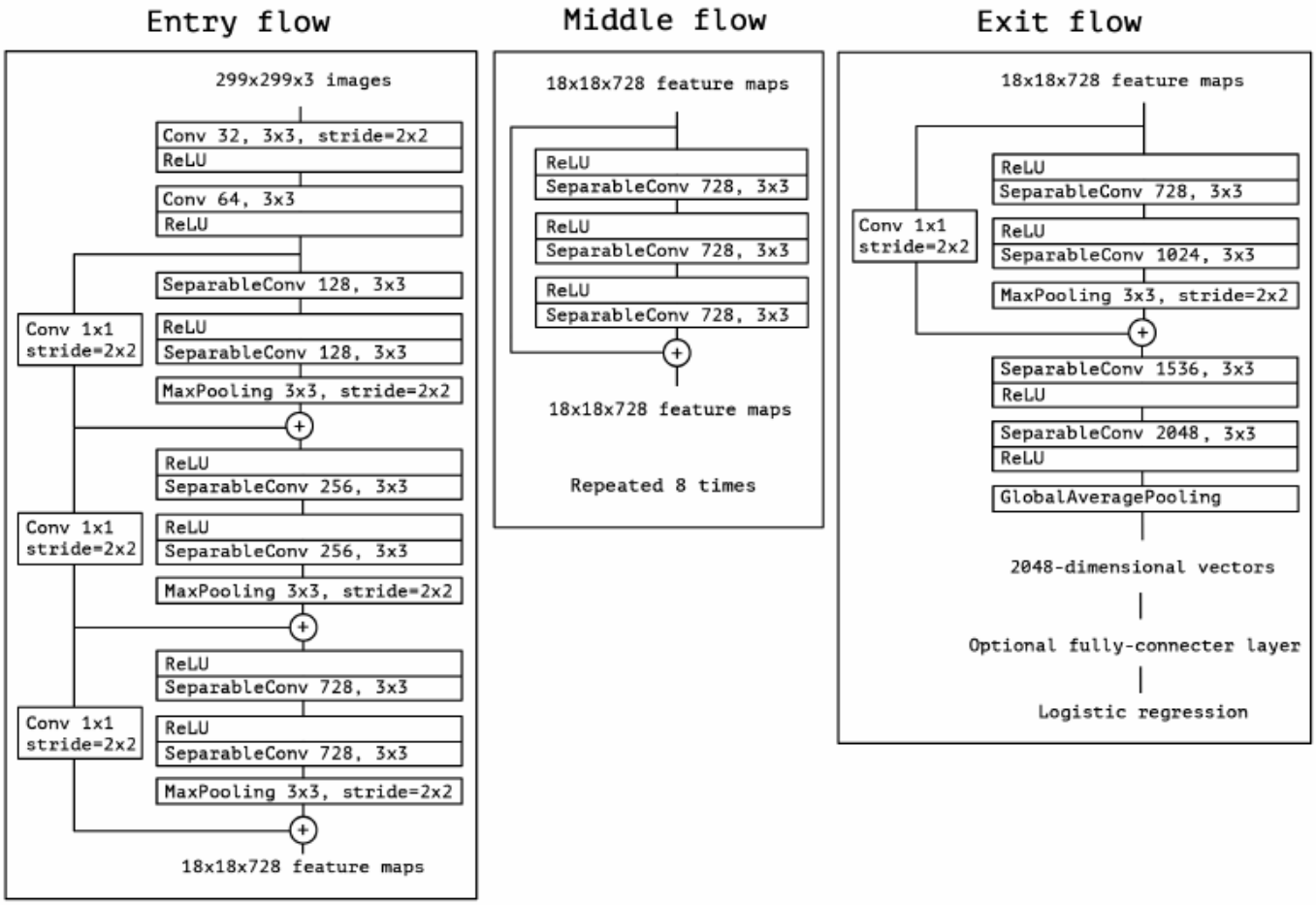
Use Cases

- **Image Classification**: Top results on ImageNet.
- **Object Detection**: Used as backbone in Faster R-CNN and Mask R-CNN.
- **Segmentation**: Used in U-Net and DeepLab for medical images and semantic segmentation.

Pros and Cons

Pros	Cons
Solves vanishing gradients in deep networks.	Computationally expensive for very deep models.
Enables training very deep networks (100+ layers).	Overhead of skip connections in smaller tasks.
Efficient learning of complex features.	

Xception (Extreme Inception)



What is Xception?

Xception stands for **"Extreme Inception"** and is a highly efficient architecture that replaces traditional convolutions with **depthwise separable convolutions**.

Key Innovation

1. **Depthwise Separable Convolutions**:
   Xception factorizes a **standard convolution** into two steps:
   - **Depthwise Convolution**: Applies a single filter to each channel separately.
   - **Pointwise Convolution**: Applies a 1x1 convolution to combine the outputs.
     This reduces the number of parameters while maintaining accuracy.

Example: Standard Convolution vs. Separable Convolution

- **Standard Convolution**: Filters * Channels → High computational cost.
- **Depthwise Separable**: Reduces computations by splitting depth and spatial dimensions.

2. **Fully Convolutional Model**:
   Xception has no dense (fully connected) layers before the softmax output, making it suitable for image tasks.

Architecture

- Xception has 36 convolutional layers structured into 14 **modules**.
- Each module contains:
  - Depthwise separable convolutions.
  - Batch normalization.
  - ReLU activations.
- **Shortcut Connections** (like ResNet) are added where needed.

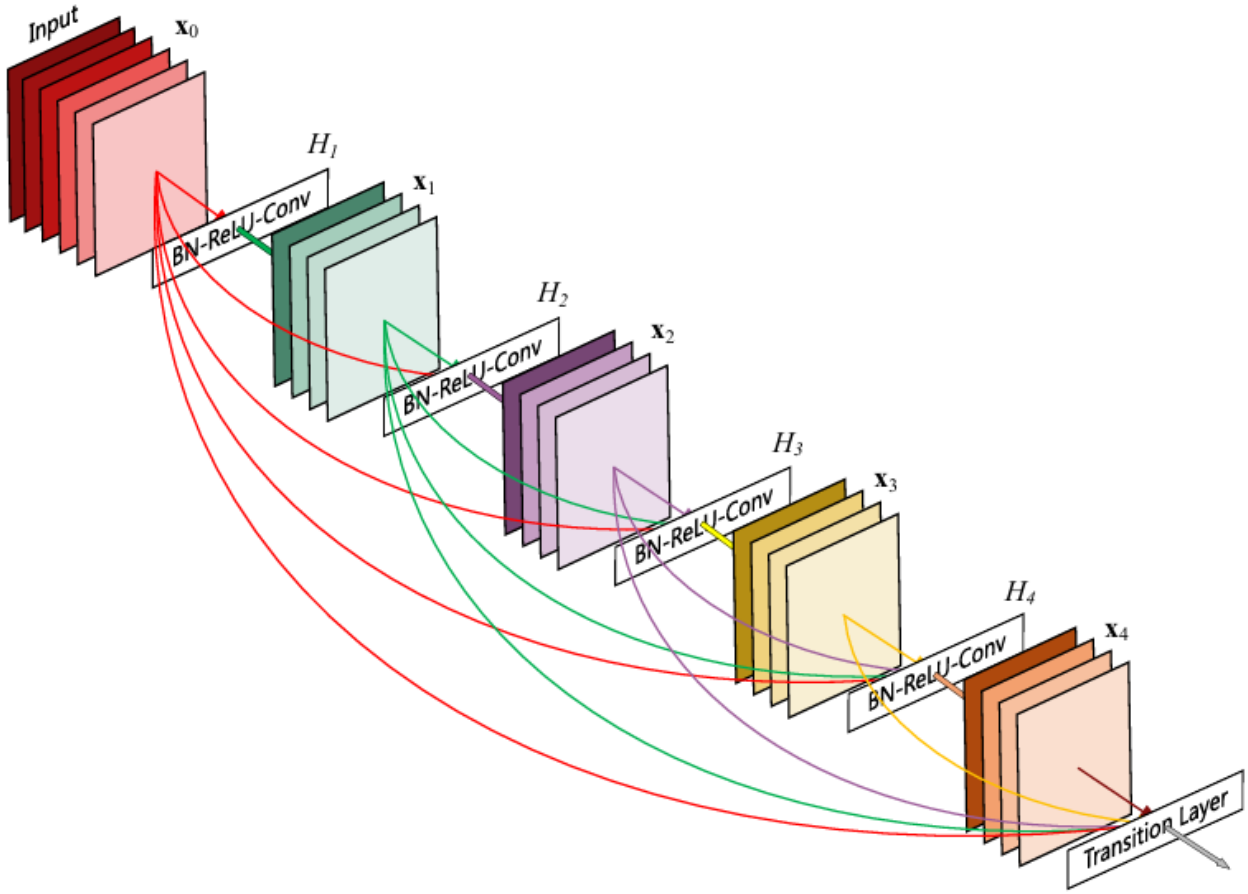
Use Cases

- **Efficient Image Classification**: Xception achieves similar or better accuracy than Inception V3 with fewer parameters.
- **Mobile and Embedded Applications**: Lightweight due to its efficient computations.
- Transfer learning for tasks like **object detection** and **segmentation**.

Pros and Cons

Pros	Cons
Lightweight and computationally efficient.	Still slower than other lightweight models like MobileNet.
Achieves high accuracy with fewer parameters.	Not as widely used as ResNet for general tasks.
Suitable for small datasets and transfer learning.	

DenseNet (Densely Connected Network)



What is DenseNet?

DenseNet focuses on enhancing feature reuse by **connecting each layer to every other layer** in a feedforward fashion.

Key Innovation

1. **Dense Connectivity**:
   - In DenseNet, the output of each layer is **passed to all subsequent layers**.
   - Each layer receives the feature maps from all previous layers as input.
   - This creates a highly **connected** network.
2. **Feature Reuse**:
   - Features from earlier layers are reused, leading to less redundancy.

    - Reduces the need for a very large number of filters.
3. **Fewer Parameters**:
    - DenseNet achieves state-of-the-art accuracy with fewer parameters compared to ResNet.

Architecture

- DenseNet is divided into **Dense Blocks**.
    - Within each block, layers are densely connected.
    - Between blocks, **Transition Layers** (convolutions + pooling) are used to reduce dimensions.
- **Variants:**
    - **DenseNet-121, DenseNet-169, DenseNet-201**: Different numbers of layers and blocks.

Use Cases

- **Image Classification**: Excellent for tasks requiring deep feature extraction.
- **Medical Imaging**: DenseNet is widely used for classification of X-rays, MRIs, etc.
- **Object Detection**: Used as a backbone in tasks like YOLO and Faster R-CNN.

Pros and Cons

Pros	Cons
Efficient feature reuse.	Increased memory consumption.
Fewer parameters compared to ResNet.	Computationally expensive on large inputs.
Reduces overfitting on small datasets.	

Comparison

Feature	ResNet	Xception	DenseNet
Main Innovation	Skip connections (Residual)	Depthwise separable convs	Dense connectivity
Depth Efficiency	Deep networks, 100+ layers	Efficient, lightweight	Deep with fewer parameters
Feature Reuse	Limited reuse	Limited reuse	High feature reuse
Computational Cost	Moderate	Low (efficient)	High memory use
Best For	General tasks, large data	Small datasets, efficiency	Feature extraction, medical

Results:

Model	Accuracy	Precision	Recall	F1-Score
ResNet	78	79	77.8	78
DenseNet	95	95	95	95
Xception	97	97	97	97

References

1 - **ResNet** stands for **Residual Network**, introduced by **Kaiming He et al.** in the paper:
"Deep Residual Learning for Image Recognition" (CVPR 2016).

2 - **XceptionNet** is an improvement over Inception V3, proposed by François Chollet in the paper:

"Xception: Deep Learning with Depthwise Separable Convolutions" (2017).

3 - **DenseNet** was introduced by **Gao Huang et al.** in the paper:
"Densely Connected Convolutional Networks" (CVPR 2017).