**Deep Learning Documentation**

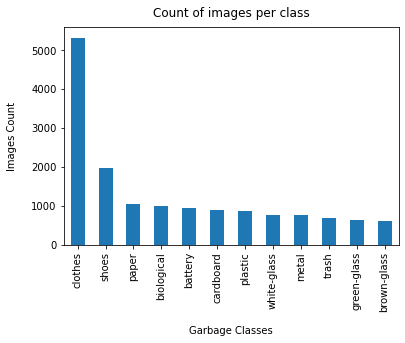
|  |  |  |
| --- | --- | --- |
| NO. | Name | ID |
| 1 | عبدالله خالد عبدالله | 20210543 |
| 2 | محمد ايمن عبدالفتاح | 20210750 |
| 3 | عمر عبدالله دياب | 20210609 |
| 4 | امنيه خليل محمد | 20210181 |
| 5 | اميره اشرف عبد الحميد | 20210184 |
| 6 | روان عزت محمد عزت | 20210341 |

**Our Dataset Details:**

Link: ( <https://www.kaggle.com/datasets/mostafaabla/garbage-classification?select=garbage_classification> )

Total Images: 15,150 images

Total number of Classes: 12 different classes of household garbage; paper, cardboard, biological, metal, plastic, green-glass, brown-glass, white-glass, clothes, shoes, batteries, and trash.



**ResNet-50**

**Paper link: (** [**https://arxiv.org/pdf/1512.03385**](https://arxiv.org/pdf/1512.03385) **)**

ResNet-50, or Residual Network-50, is a deep convolutional neural network architecture that is highly effective in image classification and other computer vision tasks. It was introduced as part of the ResNet family in the landmark paper “Deep Residual Learning for Image Recognition” by Kaiming He et al. in 2015. ResNet-50 is renowned for its ability to train extremely deep networks without succumbing to the vanishing gradient problem, a major challenge in training deep networks.

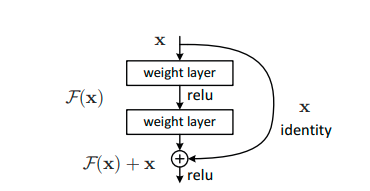
**Key Features of ResNet-50:**

## Depth:

* ResNet-50 consists of 50 layers, including convolutional layers, pooling layers, and fully connected layers.
* The architecture uses 16 residual blocks, which help manage the complexity of training such a deep network.

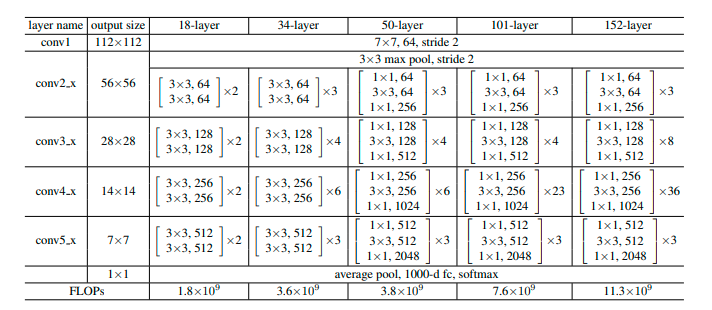
## Residual Learning:

* The core innovation is the introduction of *residual blocks*. These blocks bypass or skip certain layers by using skip connections.
* Mathematically, a residual block computes *F(x)+x,* where:
* *F(x)*: The learned function.
* *x*: The input to the block.
* These skip connections help the network focus on learning residual mappings (the difference between the desired and initial mappings) rather than the entire transformation.



## Bottleneck Design:

* To reduce computational complexity, ResNet-50 adopts a bottleneck design in its residual blocks:
* A 1x1 convolution reduces dimensions.
* A 3x3 convolution performs feature extraction.
* A 1x1 convolution restores the original dimensions.
* This three-layer design allows efficient computation while maintaining network capacity.
* **50-layer ResNet**: We replace each 2-layer block in the 34-layer net with this 3-layer bottleneck block, resulting in a 50-layer ResNet (Table 1). We use option B for increasing dimensions. This model has 3.8 billion FLOPs



## Activation Function:

* The ReLU (Rectified Linear Unit) activation function is used after most convolutional layers, introducing non-linearity and enabling better learning of complex patterns.

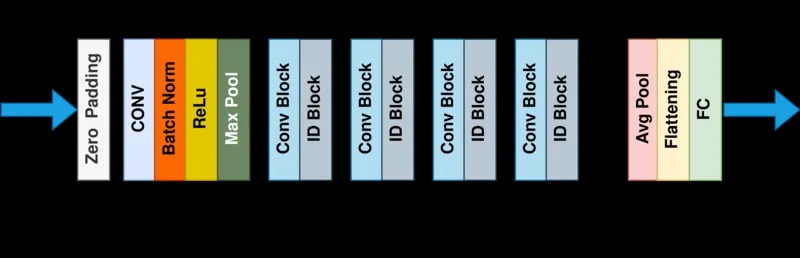
## Batch Normalization:

* Batch normalization is applied to stabilize learning and improve convergence by normalizing inputs to each layer.

## Output:

* The output layer is a fully connected layer with softmax activation for multi-class classification.

**ResNet-50 Architecture Overview**



## Input Layer:

* Takes an image of size 64X64X3 (height, width, channels).

## Initial Convolution + Pooling:

* A *7×7* convolution with 64 filters (stride = 2) is followed by a max pooling layer (*3×3*, stride = 2).

## Four Stages of Residual Blocks:

* **Stage 1:**
* 3 residual blocks, each with 64, 64, and 256 filters.
* **Stage 2:**
* 4 residual blocks, each with 128, 128, and 512 filters.
* **Stage 3:**
* 6 residual blocks, each with 256, 256, and 1024 filters.
* **Stage 4:**
* 3 residual blocks, each with 512, 512, and 2048 filters.

## Global Average Pooling:

* Reduces the spatial dimensions to *1×1*

## Output:

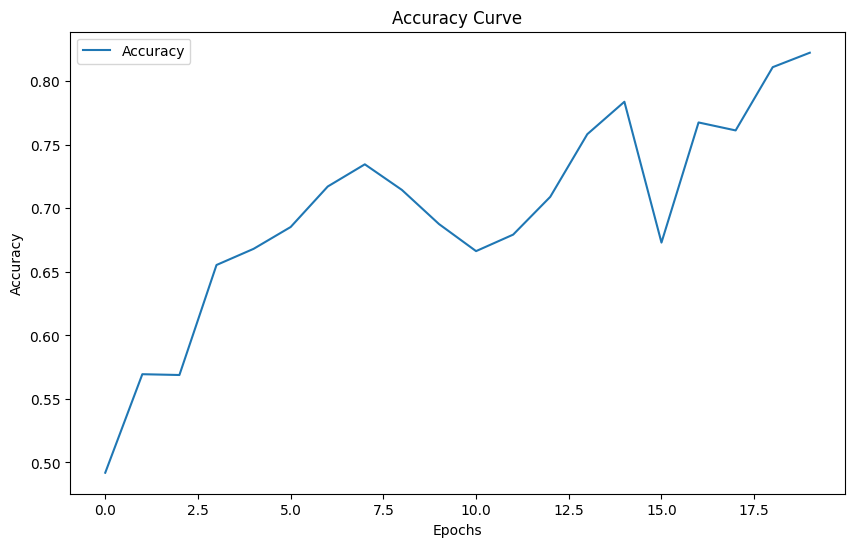
* The output layer is a fully connected layer with softmax activation for classification.

## Model Details:

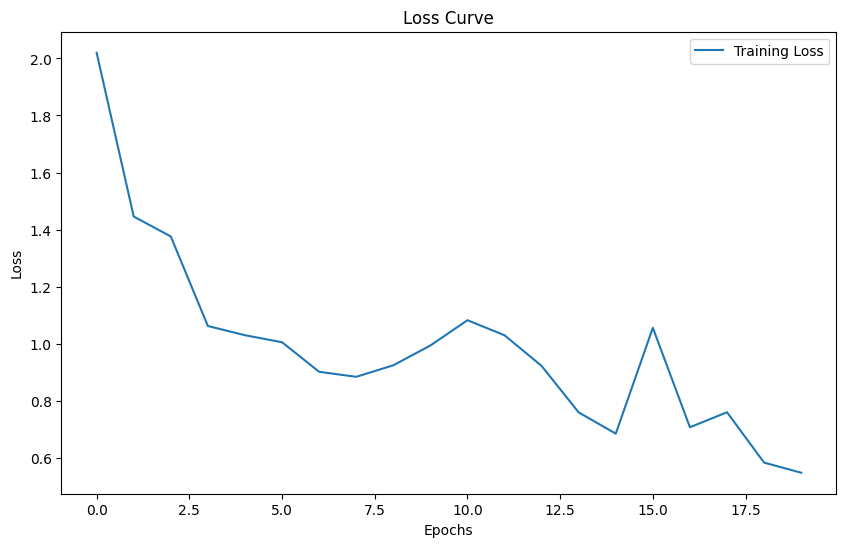
* Train Images: 80%
* Test Images: 20%
* Epochs=20
* Batch\_size=32
* validation\_split=10%
* Optimizer = “Adaptive Moment Estimation “
* Loss ="sparse\_categorical\_crossentropy"

## Graphs

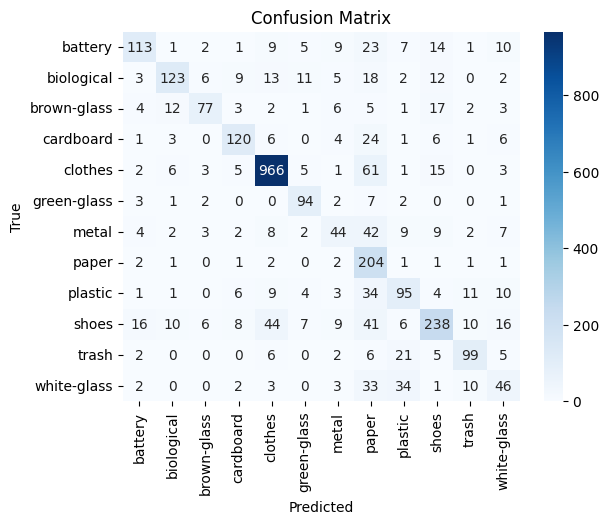
* Accuracy: 0.715114414691925



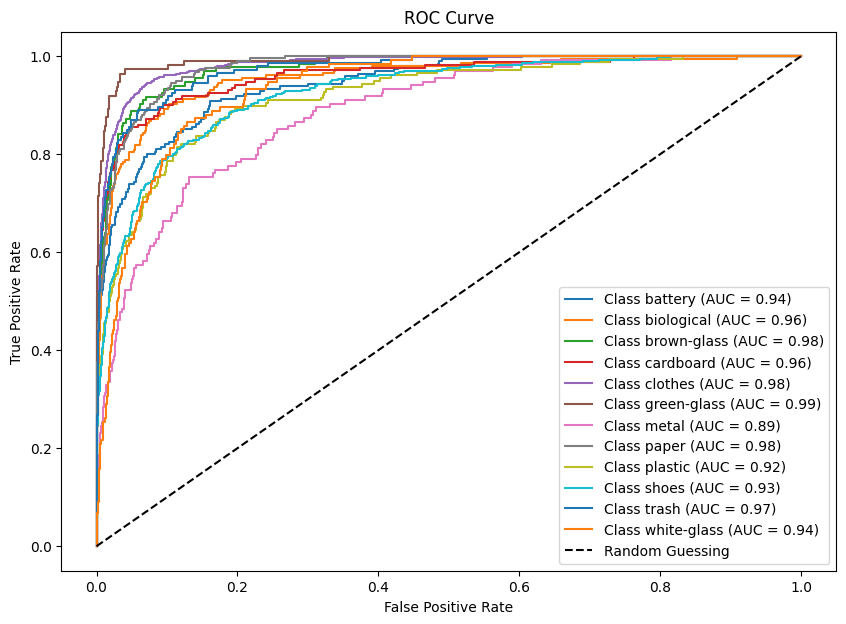
* Loss:



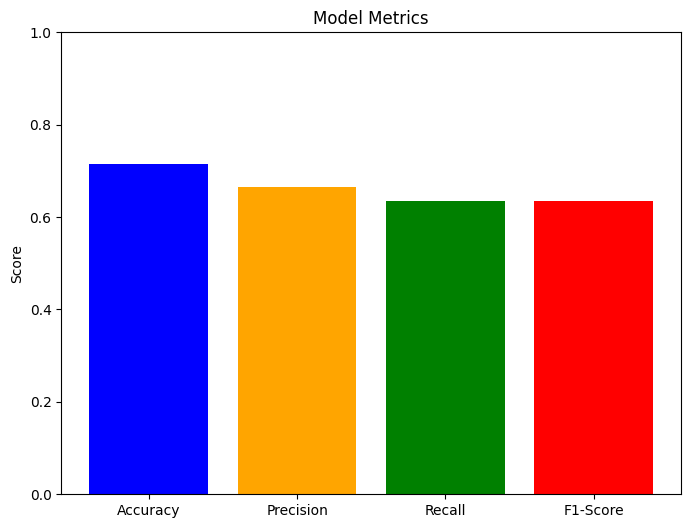
* Confusion Matrix:



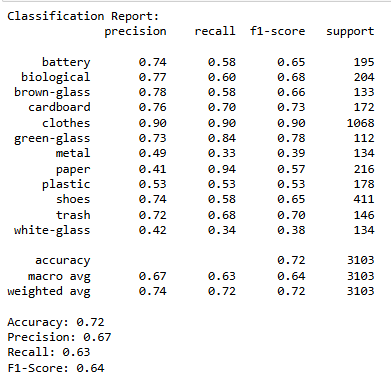
* ROC Curve:



* Model Metrics:



* Classification Report:



**Xception**  
**Paper Link: (** [**https://arxiv.org/pdf/1610.02357**](https://arxiv.org/pdf/1610.02357) **)**

Xception, short for **“Extreme Inception,”** is a deep convolutional neural network architecture that builds on the principles of the Inception model while taking it a step further. It was proposed by François Chollet in 2017 in the paper *“Xception: Deep Learning with Depthwise Separable Convolutions.”* Xception replaces traditional convolution operations with **depthwise separable convolutions**, making it highly efficient and performant for image classification and other computer vision tasks.

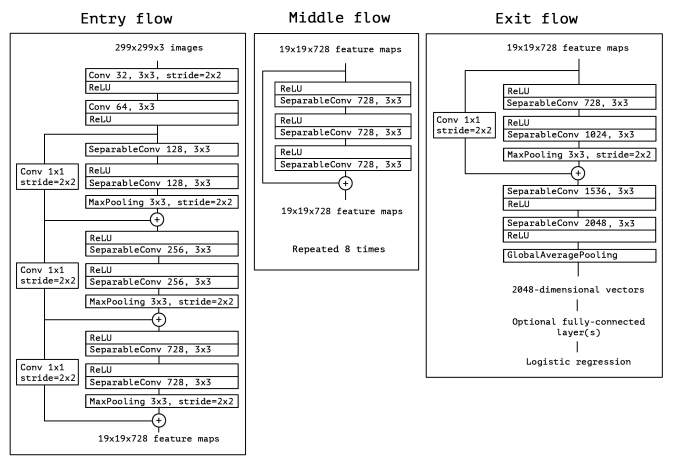
**Key Features of Xception:**

## Depthwise Separable Convolutions:

* Xception replaces standard convolution layers with depthwise separable convolutions.
* A depthwise separable convolution splits the convolution operation into two parts:
* **Depthwise Convolution**: Applies a single convolutional filter to each input channel independently.
* **Pointwise Convolution**: Combines the outputs from the depthwise convolution by applying a 1x1 convolution across all channels.
* This separation reduces the computational cost and number of parameters compared to traditional convolutions.

## Full Convolutional Stack:

* Unlike Inception modules, Xception abandons the branching structure and uses a sequence of depthwise separable convolutions.
* This results in a simpler architecture that captures spatial and cross-channel correlations more effectively.



## Linear Stacking:

* Unlike Inception, which uses modules with multiple branches, Xception stacks depthwise separable convolutions in a linear sequence, simplifying the architecture.

## ReLU Activation and Batch Normalization:

* Each convolution layer is followed by batch normalization for stability and ReLU activation for non-linearity.

## Residual Connections:

* Xception integrates residual connections to improve gradient flow during backpropagation, like ResNet.

**Xception Architecture Overview**

## Input:

* Xception accepts images of size 71x71x3 (height, width, color channels).

## Flow of Operations:

**Entry Flow:**

* The entry flow extracts basic features from the input image.
* **Sequence:**
* A ***3×3*** convolution with stride = 2 for downsampling.
* Followed by depthwise separable convolutions with increasing filter sizes.
* Ends with residual connections to aid in feature learning.

**Middle Flow:**

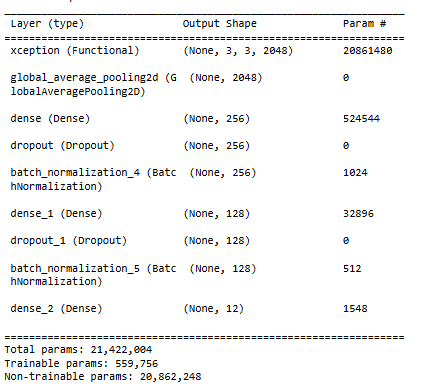
* The middle flow forms the core of the architecture and consists of **8 identical modules**.
* Each module is composed of depthwise separable convolutions with residual connections.
* No downsampling occurs in this stage, preserving the spatial resolution.

**Exit Flow:**

* The exit flow extracts higher-order features and reduces spatial dimensions further.
* **Sequence:**
* Depthwise separable convolutions with large filter sizes.
* Global average pooling to reduce the feature map to a single vector.
* Ends with a fully connected layer for classification.

## Global Average Pooling:

* The network employs **global average pooling** before the softmax output layer, reducing feature maps into a single vector.



## Output:

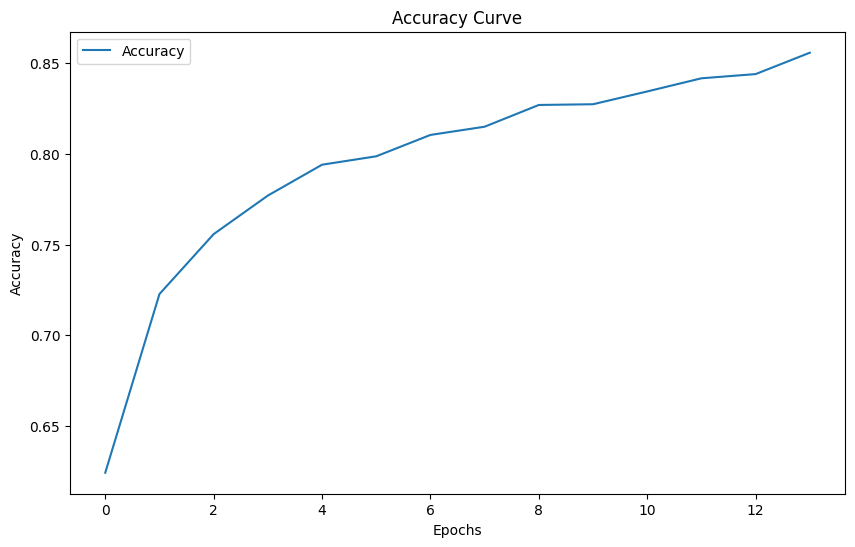
* The output layer is a fully connected layer with softmax activation for classification.

## Model Details:

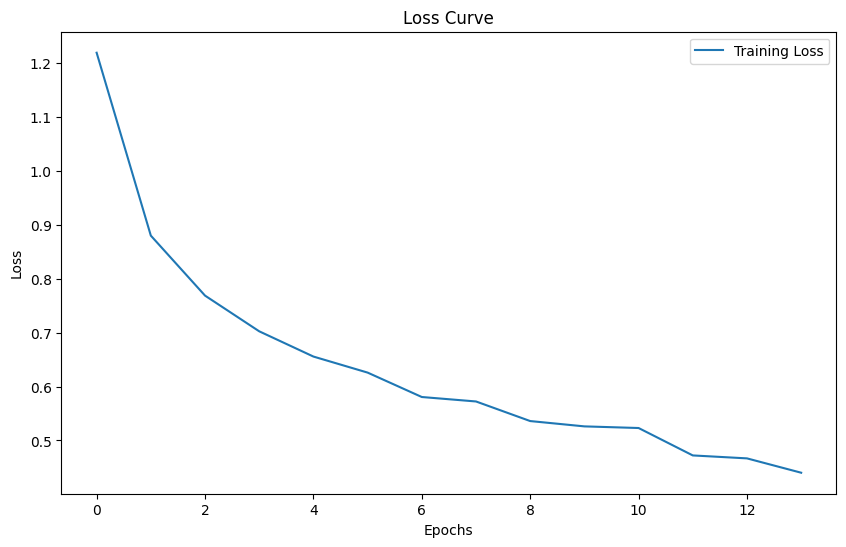
* Train Images: 80%
* Test Images: 20%
* Epochs=20
* Batch\_size=32
* validation\_split=20%
* Optimizer = “Adaptive Moment Estimation “
* Loss ="sparse\_categorical\_crossentropy"

## Graphs

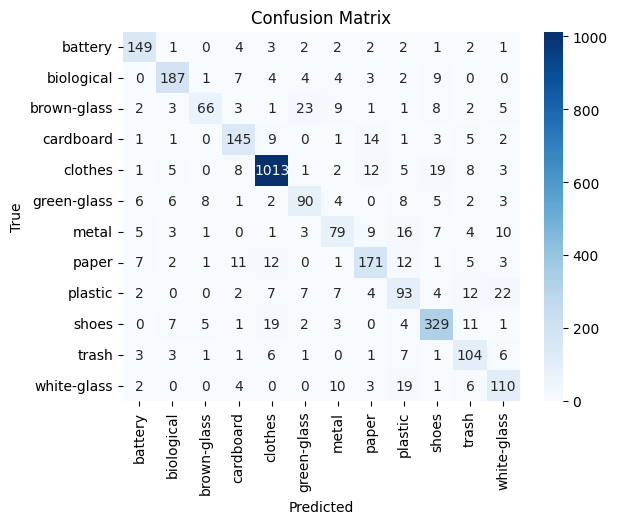
* Accuracy: 0.8172736167907715



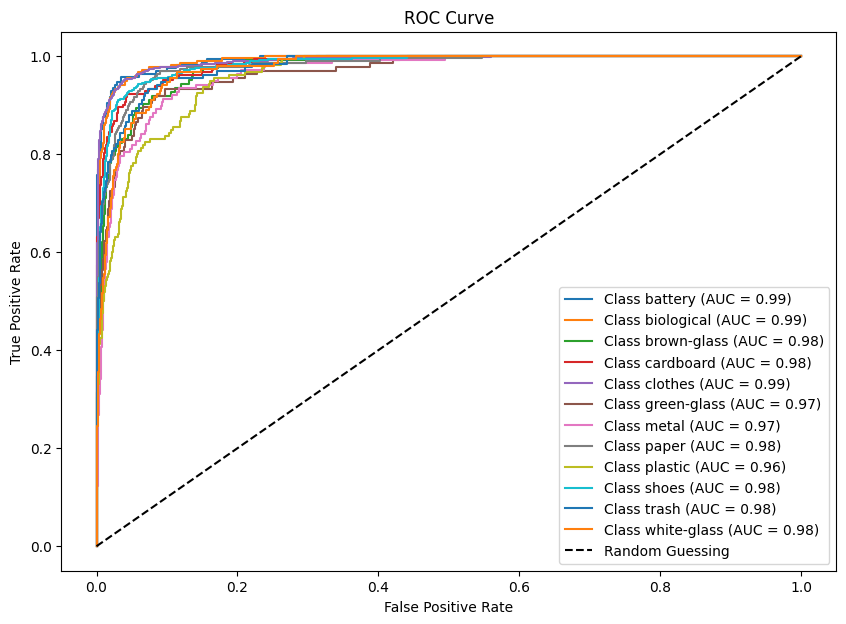
* Loss Curve:



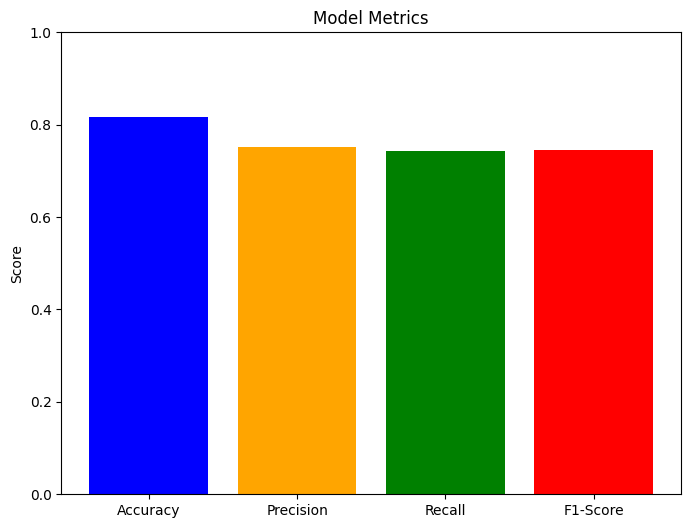
* Confusion Matrix:



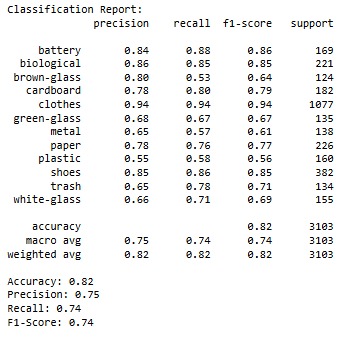
* ROC Curve:



* Model Metrics:



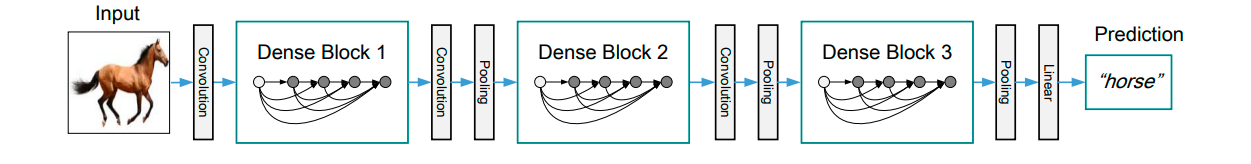
* Classification Report:



**DenseNet**

**Paper Link: (** [**https://arxiv.org/pdf/1608.06993**](https://arxiv.org/pdf/1608.06993) **)**

Convolutional neural networks (CNNs) have been at the forefront of visual object recognition. From the pioneering LeNet to the widely used VGG and ResNets, the quest for deeper and more efficient networks continues. A significant breakthrough in this evolution is the Densely Connected Convolutional Network, or DenseNet, introduced by Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. DenseNet's novel architecture improves information flow and gradient propagation, offering numerous advantages over traditional CNNs and ResNets.

A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

**Key Features of DenseNet:**

## Dense Connectivity:

* Unlike traditional CNNs where each layer receives input from only the previous layer, in DenseNet, every layer is directly connected to every other subsequent layer.
* In DenseNet, each layer is connected to every other layer in a **feed-forward** manner. This means the input of a layer is the concatenation of the feature maps from all previous layers.
* If there are *L* layers, each layer has { *L(L+1)/2* } direct connections, promoting feature reuse.
* Mathematically, the output of the l^th layer is defined as: Where H(l) is a composite function (batch normalization, ReLU, convolution), and [x0, x1, . . ., x l−1] represents the concatenation of feature maps from all preceding layers.



## Efficient Feature Reuse:

* Layers in DenseNet do not re-learn redundant features but instead build on the features learned by previous layers.
* This results in better parameter efficiency and reduced chances of overfitting.

## Reduced Parameters:

* DenseNet requires fewer parameters compared to traditional CNNs because it does not need to relearn redundant features. Instead, it uses narrow layers (e.g., fewer filters per convolution) and avoids deep architectures with redundant learning.

## Alleviation of Vanishing Gradient:

* The dense connectivity ensures that gradients flow more effectively during backpropagation, mitigating the vanishing gradient problem in very deep networks.

## DenseNet Variants

Link: ( <https://www.geeksforgeeks.org/densenet-explained/> )

|  |  |  |  |
| --- | --- | --- | --- |
| Variant | Layers | Parameters | Typical Use Cases |
| DenseNet-121 | 121 | 7.98M | General-purpose image classification, object detection |
| DenseNet-169 | 169 | 14.15M | Advanced image recognition, medical image analysis |
| DenseNet-201 | 201 | 20.01M | High-accuracy tasks, detailed feature extraction |
| DenseNet-264 | 264 | 33.34M | Complex visual tasks, extensive datasets |

**DenseNet-121 Architecture Overview**

DenseNet is divided into multiple Dense Blocks separated by Transition Layers.

## Dense Block:

* A dense block is a sequence of convolutional layers where each layer receives the outputs of all preceding layers as input.
* **Number of Layers per Dense Block:**

The number of layers in each dense block for DenseNet-121 is:

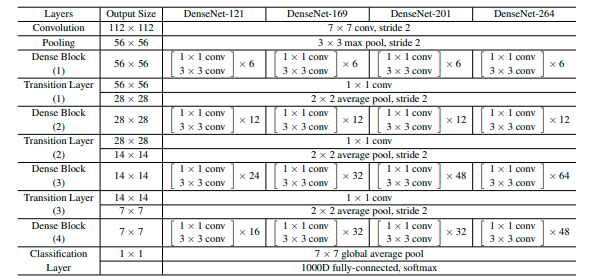
* **Dense Block 1:** 6 layers.
* **Dense Block 2:** 12 layers.
* **Dense Block 3:** 24 layers.
* **Dense Block 4:** 16 layers.
* Each layer performs:
* **Batch Normalization (BN):** Normalizes the inputs for stable and faster training.
* **ReLU Activation:** Introduces non-linearity.
* ***3×3* Convolution:** Extracts spatial features.

#### Transition Layers:

* Dense blocks are connected by transition layers, which reduce the spatial dimensions and control the number of feature maps.
* Transition layers include:
* ***1×1* Convolution:** Reduces the number of channels.
* ***2×2* Average Pooling:** Halves the spatial dimensions.

#### Growth Rate (k):

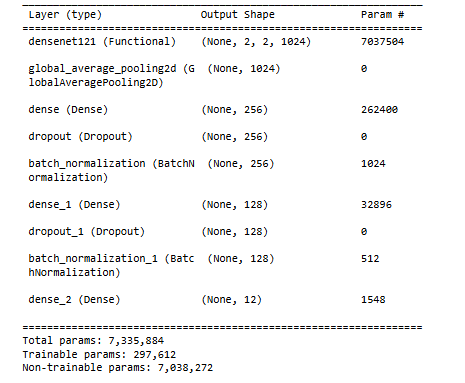
* The growth rate determines the number of new feature maps added by each layer in a dense block.
* If *k=32*, each layer contributes 32 additional feature maps, ensuring manageable model size while promoting feature reuse.



DenseNet architectures for ImageNet. The growth rate for all the networks is k = 32. Note that each “conv” layer shown in the table corresponds the sequence BN-ReLU-Conv.

#### Global Average Pooling and Classification:

* After the last dense block, global average pooling aggregates the feature maps, which are passed to the fully connected layer for classification.



## Output:

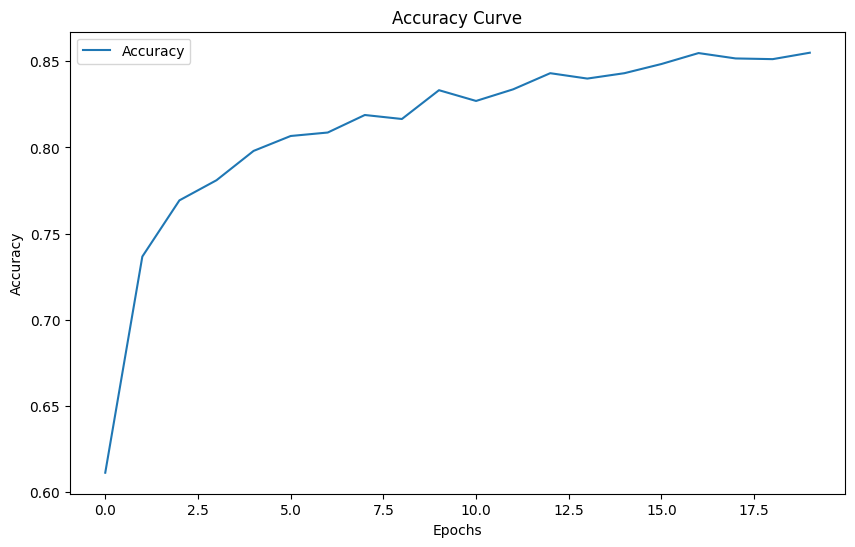
* The output layer is a fully connected layer with softmax activation for classification.

## Model Details:

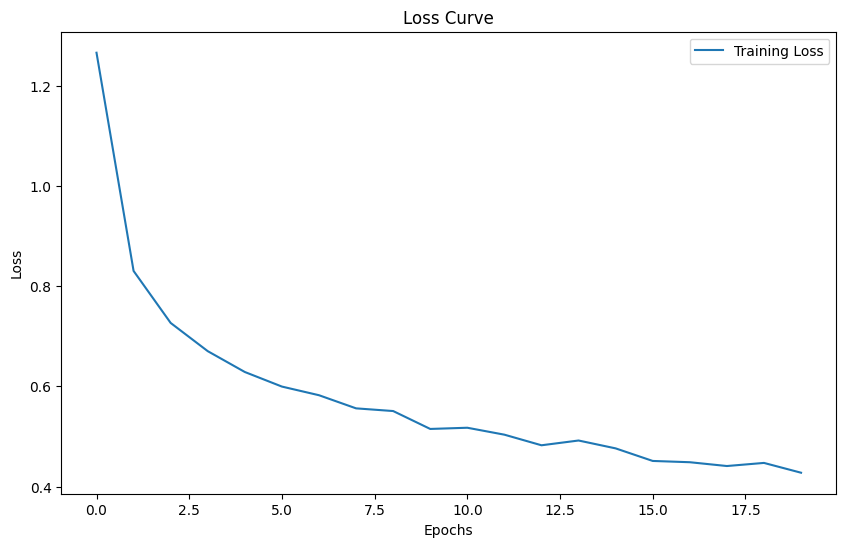
* Train Images: 80%
* Test Images: 20%
* Epochs=20
* Batch\_size=32
* validation\_split=20%
* Optimizer = “Adaptive Moment Estimation “
* Loss ="sparse\_categorical\_crossentropy"

## Graphs

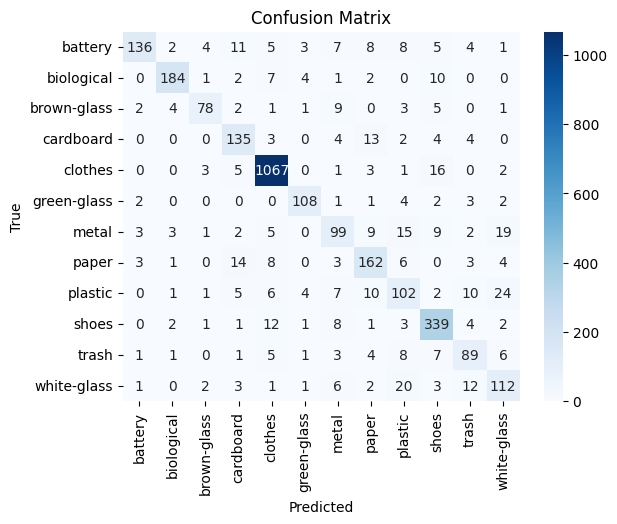
* Accuracy: 0.8414437770843506



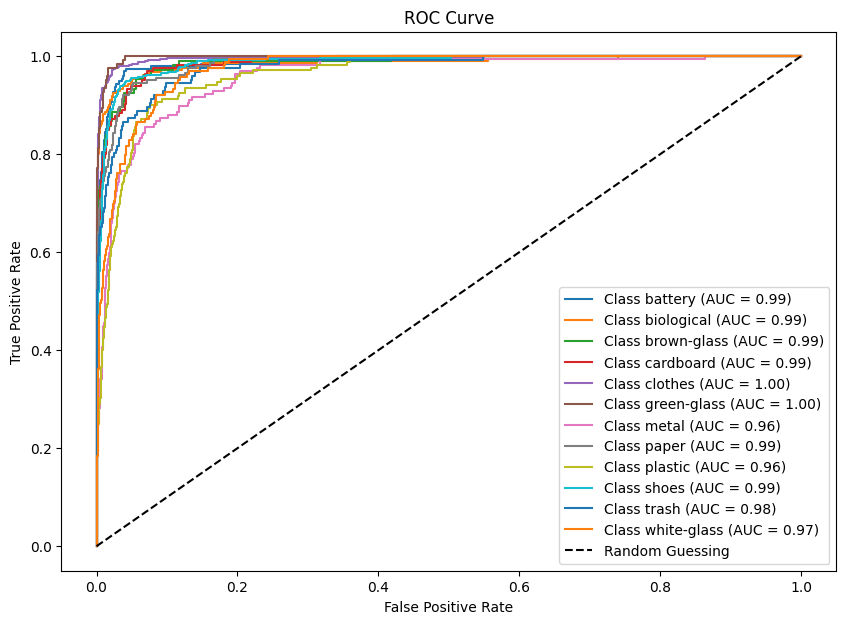
* Loss Curve:



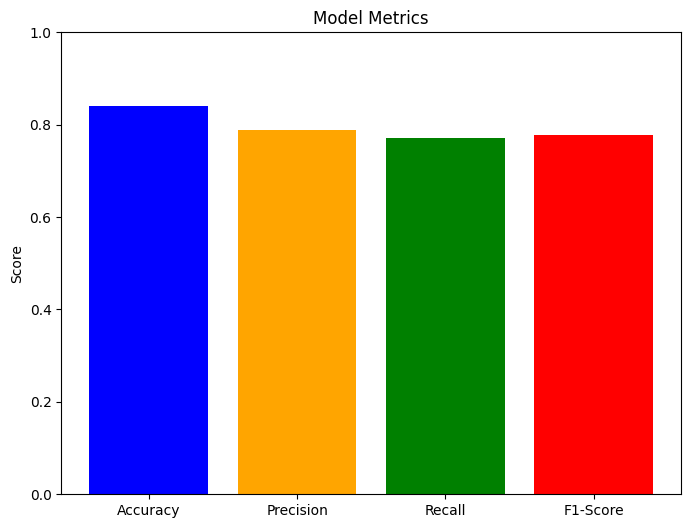
* Confusion Matrix:



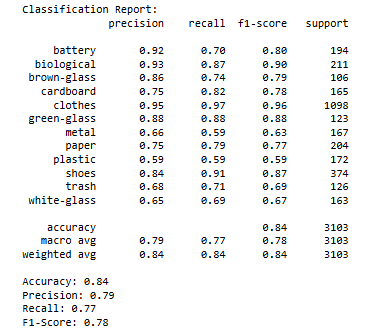
* ROC Curve:



* Model Metrics:



* Classification Report:



**Comparison between our 3 Models**

|  |  |  |  |
| --- | --- | --- | --- |
| Compare | ResNet-50 | Xception | DesneNet-121 |
| Accuracy | 0.72 | 0.82 | 0.84 |
| Connections | Residual | Depthwise separable | Dense |
| Training Speed | Slower | Moderate | Fast |
| Weaknesses | Residual connections improve training, they can increase model complexity slightly compared to simpler architectures. | Computational overhead during model initialization | Dense connectivity can lead to higher memory usage during training. |

So, according to our models DesneNet-121 is best performance but theoretically Xception is better performance