A logo for a computer company

Description automatically generatedA logo with a symbol and text

Description automatically generated

Hand Gesture

Classification

Using ResNet, Xception, DenseNet Models

Neural Networks & deep learning

Team members:

|  |  |
| --- | --- |
| *Name* | *ID* |
| احمد نور یحي عبد المنعم | 20210122 |
| ابراھیم مصطفى علي وداعه الله | 20210009 |
| أحمد یحیي عبد المنعم | 20210129 |
| اسامه محمد سلیمان محمد قشطه | 20210141 |
| احمد سعد الدین محمد علي | 20210057 |
| كریم یونس ذكي | 20210689 |
| فادي ھاني عبد الملاك مسیحه | 20210656 |

Index

|  |  |
| --- | --- |
| Page Number | Content |
| 3 | Introduction to the Project and System overview |
| 3 | About the Datasets |
| 4 | ResNet Model Architecture and Implementation |
| 7 | Xception Model Architecture and Implementation |
| 10 | DenseNet Model Architecture and Implementation |
| After each model | Results of the Models (Accuracy, Visualization, Loss Curve, Confusion Matrix) |
| 13 | Conclusion |

**Introduction to Hand Gesture Classification System Documentation**

This documentation provides a comprehensive overview of a hand gesture **Classification** system designed to identify and classify human gestures based on input images or video streams. The system utilizes three state-of-the-art deep learning models—ResNet, Xception, and DenseNet—to achieve high accuracy and robust performance across diverse datasets.

**Purpose and Scope**

Hand gesture **Classification** has applications in various domains, including human-computer interaction, augmented reality, sign language interpretation, and contactless control systems. This project aims to leverage the strengths of three advanced convolutional neural network architectures to create an efficient and scalable solution.

**Key Features**

* **ResNet (Residual Networks):** Known for its deep architecture and skip connections, ResNet addresses the vanishing gradient problem and ensures efficient feature learning.
* **Xception (Extreme Inception):** A depthwise separable convolution model that optimizes performance by reducing computational complexity while maintaining high accuracy.
* **DenseNet (Dense Convolutional Networks):** Encourages feature reuse through dense connectivity, making the model highly efficient for gradient propagation and feature sharing.

**System Overview**

The hand gesture **Classification** system processes input data through a preprocessing pipeline, extracts features using the selected model, and classifies gestures into predefined categories. Each model contributes unique strengths, making the system adaptable to various use cases and performance requirements

**About Dataset**

The hand gesture **Classification** dataset was created by subtracting the background from the hand images using OpenCV.The dataset contains of 10 classes: [call\_me, rock\_on, fingers\_crossed, okay, paper, peace, rock, scissor, thumbs, up]  
Each class consists of around 500 images.

A screenshot of a computer

Description automatically generated

[Kaggle Dataset URL](https://www.kaggle.com/datasets/roobansappani/hand-gesture-recognition)

**ResNet (Residual Network)**

ResNet, or Residual Network, is a groundbreaking deep learning architecture introduced by He et al. in 2015. It addresses the challenges of training very deep neural networks, particularly the **vanishing gradient problem**, which can hinder the effective flow of gradients during backpropagation in deep layers.

A diagram of a model architecture

Description automatically generated

A table with numbers and lines

Description automatically generated with medium confidence

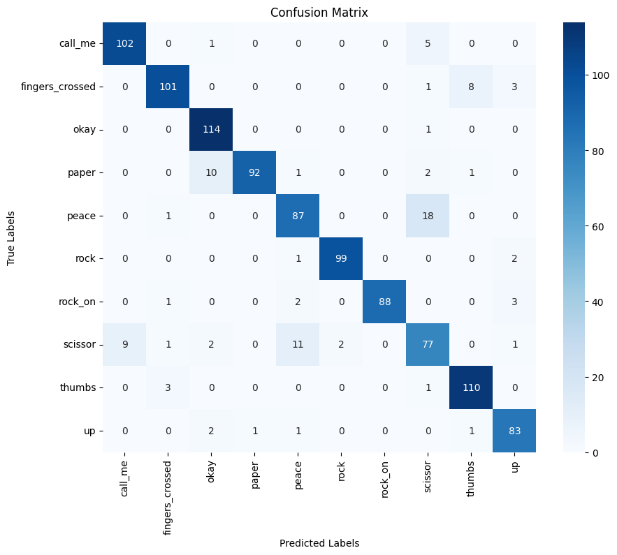
Our model utilizes a custom implementation of ResNet-50, built from scratch to leverage the powerful feature extraction capabilities of this architecture.

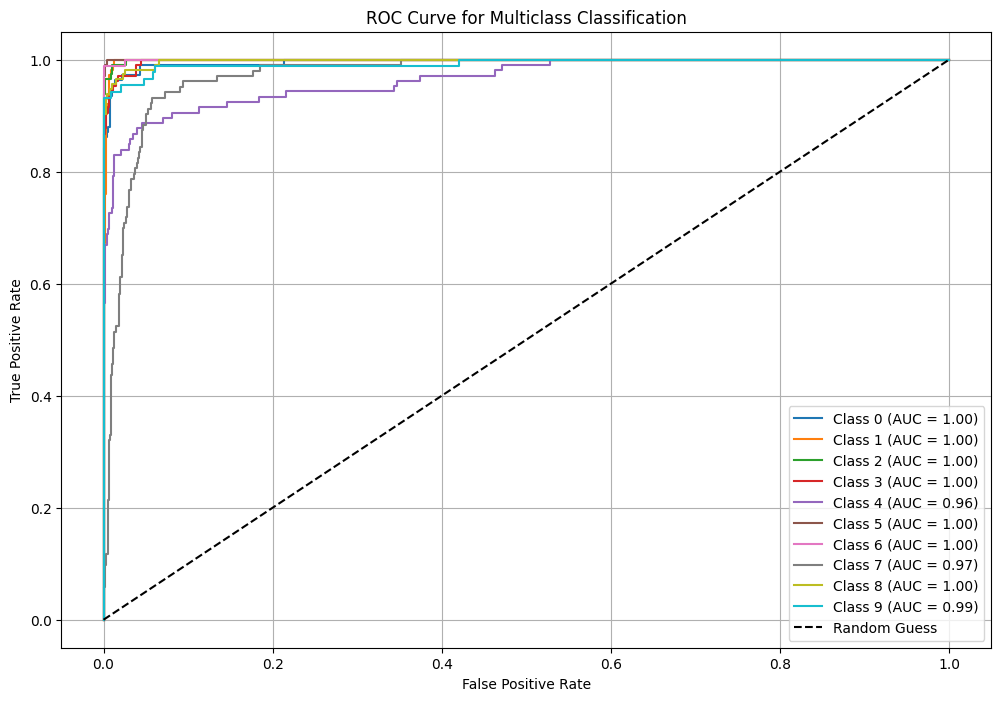
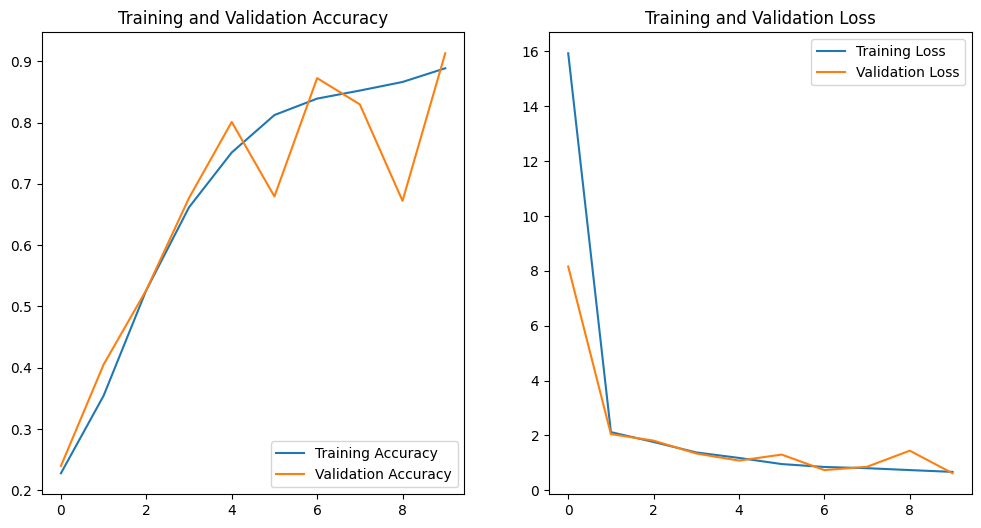
For preprocessing, we resize the input images and normalize their pixel values to the range of 0 to 255, ensuring consistency and optimal performance during training and inference.

We create a structured DataFrame to manage the dataset, where each entry includes the image path and its corresponding label. This streamlined preprocessing pipeline ensures efficient data handling and prepares images for effective learning by the ResNet-50 model.

|  |  |
| --- | --- |
| Pros | Cons |
| * Customizable and Flexible Architecture * Ability to Modify Underlying Data * Solves Vanishing Gradient and Exploding Gradient Problems | * Time-Intensive Training * High Failure Rate if Not Properly Configured * Memory and Compute Intensive * overfitting due large number of parameters. |

Accuracy and Evaluation:

A screenshot of a computer screen

Description automatically generated

**Key Concept - Skip Connections**

* Instead of forcing the network to learn everything from scratch, ResNet adds shortcuts (skip connections) that allow information to bypass some layers.
* These connections help the network focus on learning only the difference (or "residual") between the input and the desired output.

**Residual Block**

* The core unit of ResNet is a "residual block," which processes data using layers of transformations (e.g., convolutions) and then combines it with the original input.
* This makes it easier for the network to train because the layers don't need to change the input much if it's already useful.

**Depth Without Problems**

* Traditional deep networks often struggle with vanishing gradients and overfitting as they get deeper.
* ResNet avoids these problems by making it easier for gradients to flow backward during training, thanks to the skip connections.

**Types of ResNet**

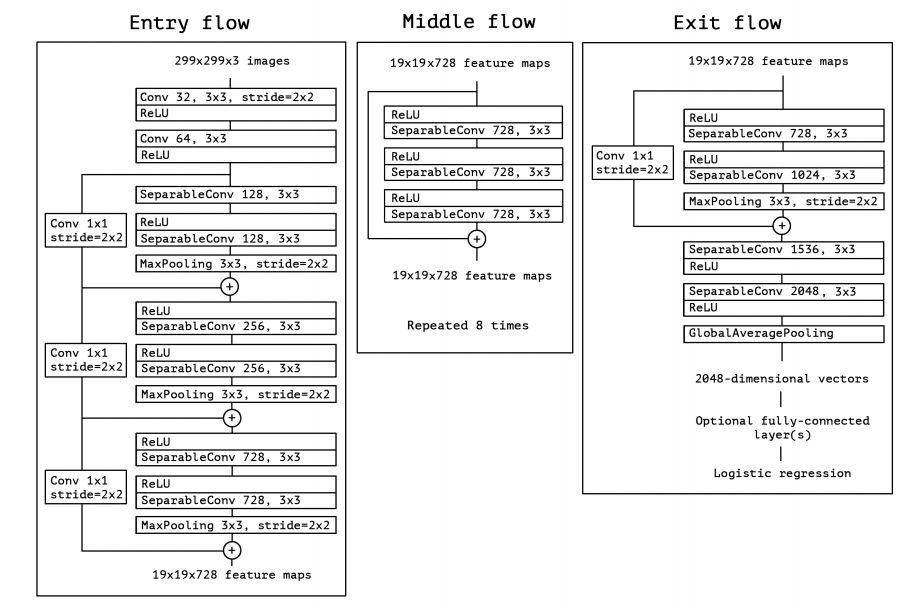
* ResNet comes in various sizes (ResNet-18, ResNet-34, ResNet-50, etc.), with the number indicating the total layers in the network.
* Larger versions use more advanced residual blocks for efficiency.

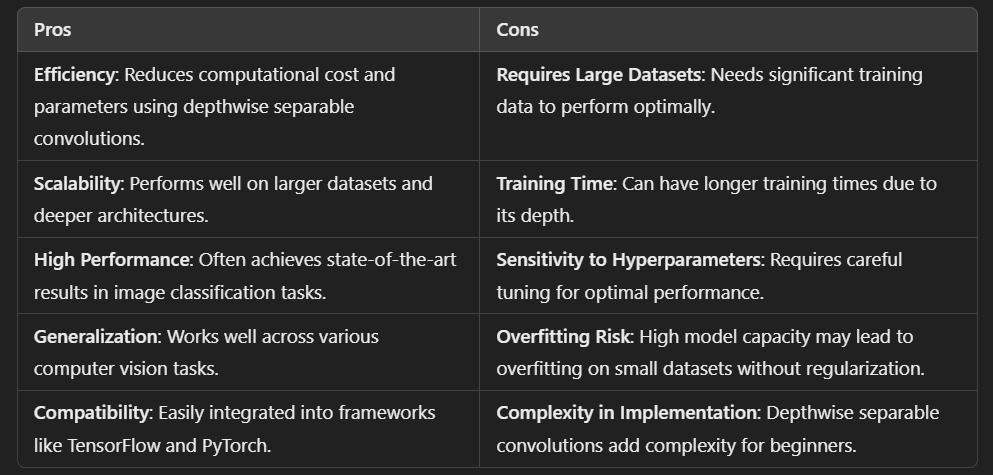
**Why It’s Powerful**

* ResNet made it possible to train networks with over 100 layers while maintaining high performance.
* It achieves excellent results in tasks like image classification and object detection and is widely used in AI applications

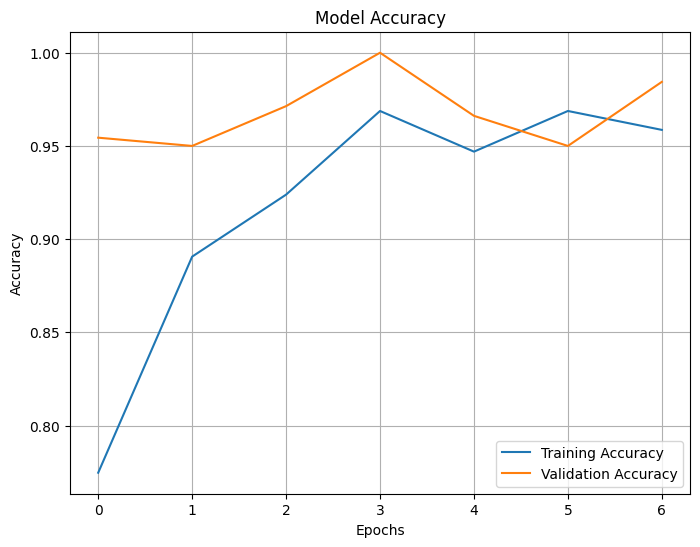
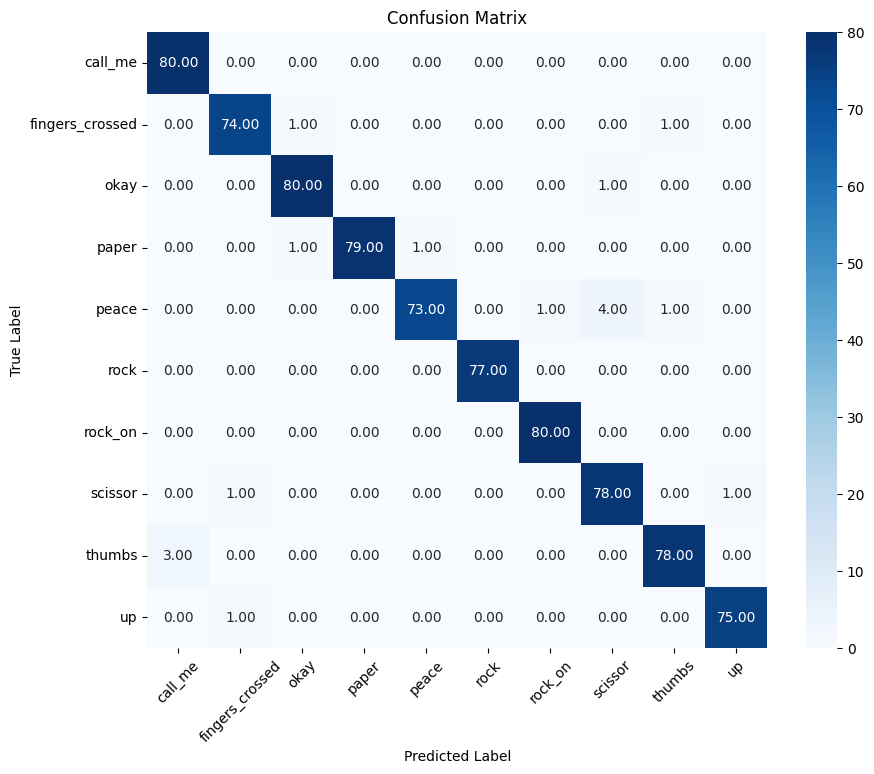
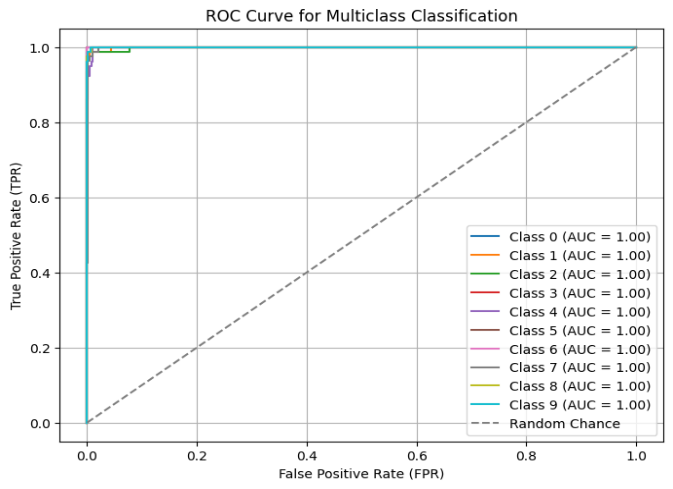
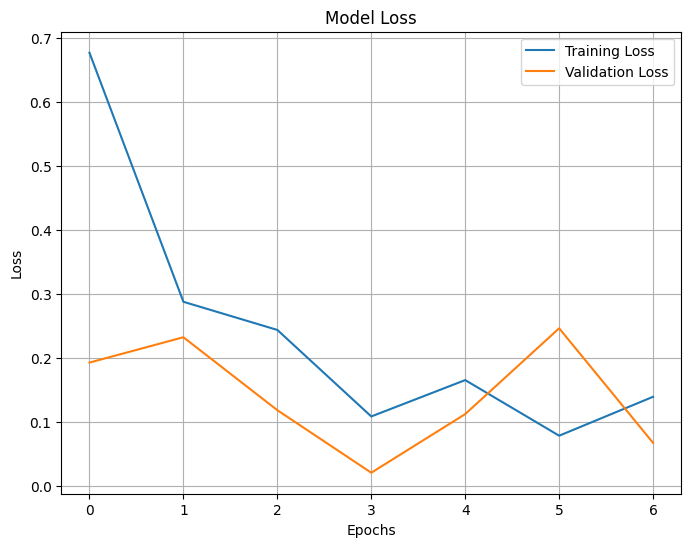
Xception

Xception (Extreme Inception) is a deep convolutional neural network architecture introduced by François Chollet in 2017. It is an extension of the Inception architecture and is designed to improve efficiency and performance by utilizing depthwise separable convolutions. Depthwise separable convolutions break down a standard convolution into two separate steps: depthwise convolution (spatial convolution applied independently to each input channel) and pointwise convolution (a 1x1 convolution that combines the outputs from the depthwise step). This separation significantly reduces computational complexity and improves the model's efficiency. Xception can be seen as a streamlined and more efficient version of Inception, where Inception modules are replaced with depthwise separable convolutions. It is widely used in computer vision tasks like image classification, object detection, and segmentation.





Accuracy and Evaluation:



A screenshot of a computer screen

Description automatically generatedA graph with a line

Description automatically generated

This code performs a hand gesture recognition task using deep learning, specifically using the Xception model for image classification. Here's a breakdown of its functionality:

1. Data Preparation:

- The script first organizes the dataset (hand gesture images) into training, validation, and test directories. It uses train\_test\_split to split the images into 70% for training, 15% for validation, and 15% for testing, while preserving the structure for each category (gesture).

2. Model Building:

- The code loads the Xception model (pre-trained on ImageNet) as a base model, excluding the top layer (for transfer learning).

- Custom layers are added on top: a GlobalAveragePooling2D layer, a dense layer with 1024 units, and a final softmax layer with 10 output classes (for gesture recognition).

3. Model Compilation:

- The model is compiled using the Adam optimizer with a learning rate of 0.001 and categorical cross-entropy loss for multi-class classification.

4. Data Augmentation:

- The training dataset is augmented with random transformations (rotation, width/height shift, zoom, and horizontal flip) using ImageDataGenerator to improve generalization.

- The validation and test datasets are rescaled but not augmented.

5. Training:

- The model is trained on the training data using early stopping (patience = 3 epochs) to avoid overfitting.

- The training process is monitored using accuracy and loss metrics, and it runs for 10 epochs.

6. Evaluation:

- After training, the model is evaluated on the test dataset, and the test accuracy and loss are printed.

- Confusion Matrix and Classification Report are generated to assess performance across the individual classes.

- ROC Curve and AUC scores for each class are computed and plotted to evaluate the model's ability to distinguish between classes.

7. Visualization:

- The training and validation accuracy/loss are plotted across epochs to visually assess model performance.

- The Confusion Matrix is displayed as a heatmap to show the true vs predicted labels.

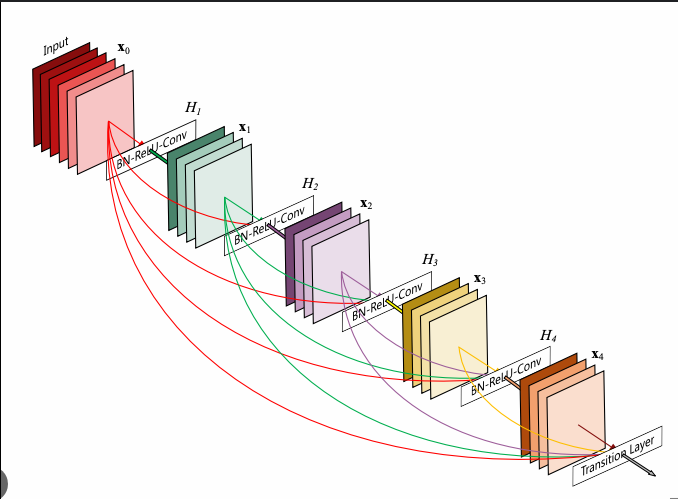
- The ROC Curve for each class is plotted to evaluate model performance for multi-class classification tasks.

In Summary:

This code prepares a hand gesture recognition dataset, builds a transfer learning model using Xception, trains it with data augmentation, and evaluates its performance using various metrics like accuracy, loss, confusion matrix, and ROC curves. It also visualizes the results for better interpretability.

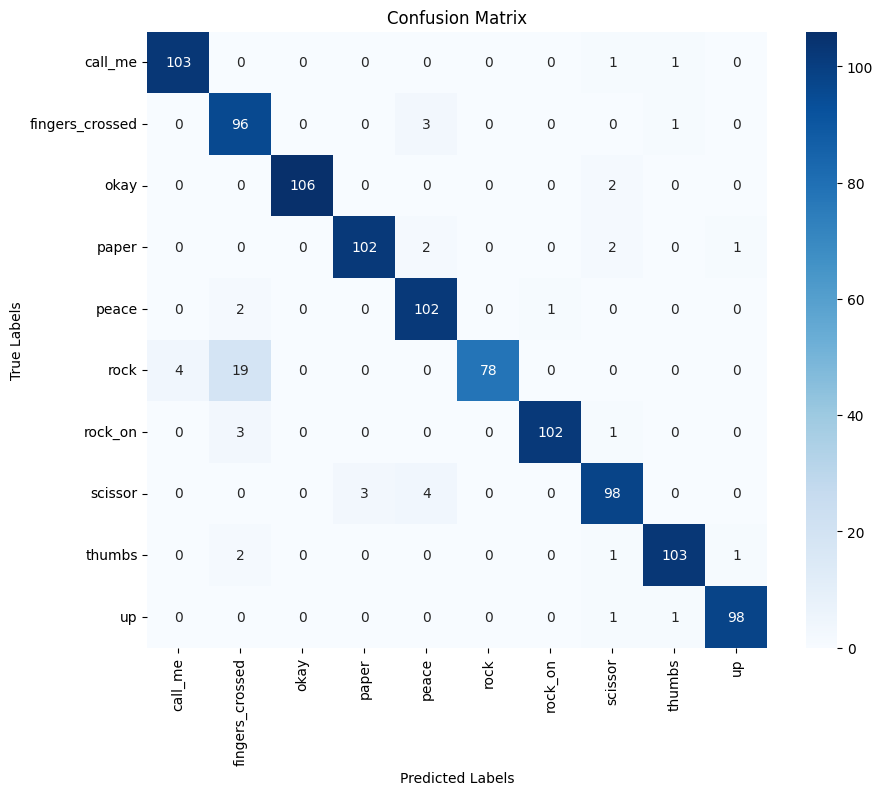
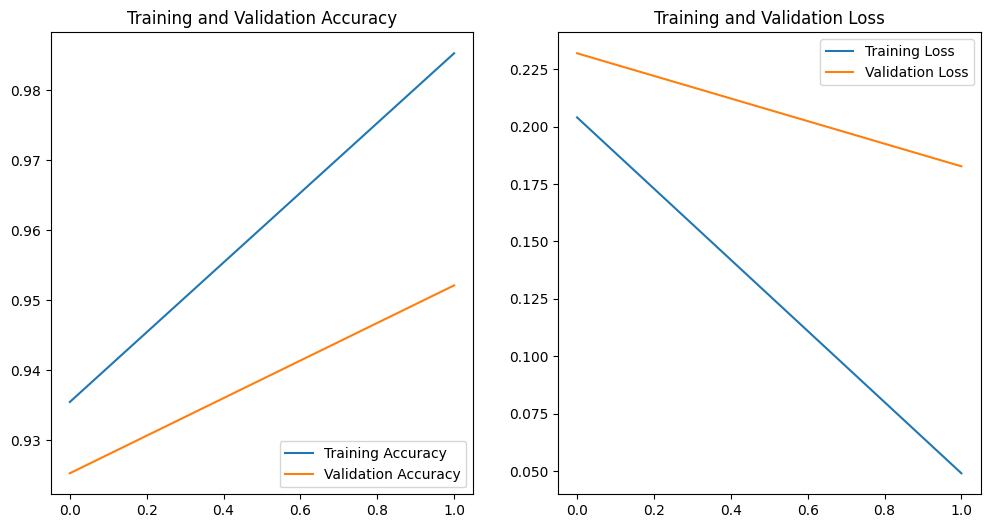
DenseNet

DenseNet (Dense Convolutional Network) is a type of deep learning architecture designed for image recognition and classification tasks. Its key idea is to connect each layer directly to every other layer in a "dense" manner. How It Works: Dense Connections: In DenseNet, each layer gets inputs from all previous layers and passes its output to all subsequent layers. This helps the model reuse features, making it efficient. Feature Propagation: Instead of learning redundant features, DenseNet focuses on combining features from earlier layers with new ones. Compact and Efficient: DenseNet uses fewer parameters compared to other architectures like ResNet because it doesn’t need as many filters, making it memory-efficient. Improved Gradient Flow: The dense connections allow gradients to flow easily back through the network, improving training stability. Benefits: Efficient use of parameters. Better accuracy with fewer layers. Reduces the risk of overfitting due to effective feature sharing. DenseNet is widely used in tasks like object detection, medical imaging, and gesture recognition because of its high performance and compact design.

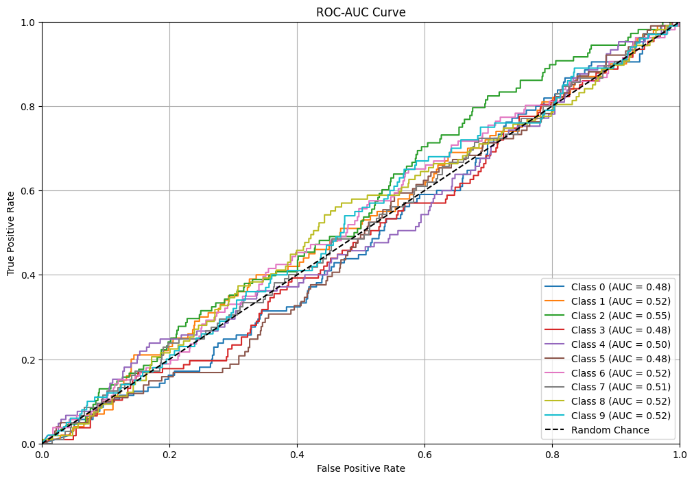


|  |  |
| --- | --- |
| Pros | Cons |
| * Better Accuracy: Reduces overfitting with efficient feature reuse. * Fewer Parameters: Less redundant information due to layer connections. * Efficient Gradient Flow: Dense connections help in training deeper networks. | * High Memory Usage: Many feature maps require more memory. * Slower Training: Increased computations due to multiple connections. * Complex Architecture: More challenging to implement compared to simpler networks. |

Accuracy and Evaluation:

A screenshot of a computer screen

Description automatically generated



Conclusion

Concatenates outputs of all previous layers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *MODEL* | |  | | --- | |  |   *KEY FEATURE* | *STRENGTHS* | *ACCURACY* | *STRENGTHS* |
| ResNet | Simple design  Residual connections | Reduces vanishing gradient issues | 91% | High memory usage Computationally intensive due to large depth |
| Xception | Depthwise  separable convolutions | Competitive accuracy with fewer parameters compared to ResNet | 98% | Performs poorly if depth wise separable convolutions are not optimized for hardware |
| DenseNet | Concatenates outputs of all previous layers | Reduces overfitting with fewer parameters | %95 | High computational overhead and memory usage due to concatenating all previous layer outputs |