Gesture project Documentation

ID	الاسم
20210222	بسام عماد حمدي
20210218	بثينه عصام محمد
20210210	ايه محمد علي
20210236	تريفينا رضا امين
20210201	ايمان ايهاب ابراهيم
20210252	جميل محمد جميل

Project overview

Project title: gesture image classification

Objective : in this project we train our dataset on 3 models to make classification for hand gesture and then we compared the evaluation of the 3 models

Data preprocessing : resizing, normalizing, split the data

Models: Resnet-34 , Densnet, Xception

Dataset Description

Dataset Source: https://www.kaggle.com/datasets/gti-upm/leapgestrecog/data

Dataset Details:

- 10 people and each person have 10 hand gesture which mean 10 classes and each class have 200 image which mean we have 20000 image
- Classes
    - Palm
    - I :I shape by hand
    - Fist
    - Fist moved : fist from the side
    - Thumb
    - Index
    - Ok
    - Palm moved: palm from the side
    - C
    - Down
- Data split
    - train=70%
    - validation=15%
    - test=15%

Steps of preprocessing

1. load the dataset from kaggel using the API method because it is really large
2. split the data to train and test and validation
3. resizing all the images
    a. check all the photos size which turn out that all images is same size (240, 640)
    b. the better size for all the images for the 3 models we will work on is (224, 224)
4. normalizing all the images
    a. make 3 folders that have train and test and val and each folder called the model will be trained on
    b. normalize the images 3 difference time based on each model and put them in their folder

5. make data augmentation for the **train** images to increase the diversity of the dataset by mirror and shearing and zooming and rotation

---

Project setup

Programming Language: Python

Frameworks:

- TensorFlow
- Keras

Libraries:

- **NumPy**: For numerical operations.
- **Pandas**: For data manipulation and analysis (e.g., dataset handling).
- **Matplotlib** / Seaborn: For data visualization.
- **OpenCV**: For image processing.
- **Scikit-learn**: For machine learning utilities.

Data Augmentation Tool:

Keras ImageDataGenerator: Used for image data augmentation during training.

Development Environment:

Google Colab: Used as the primary environment for coding, data analysis, and model training.

Models

ResNet

ResNet (Residual Networks) revolutionized the deep learning field when its authors introduced it in their 2015 paper. The architecture made a significant impact by addressing the problem of training very deep networks, where earlier models faced degradation in performance as the number of layers increased. The core idea behind ResNet is the introduction of **residual connections**, or shortcut connections, which allow the model to learn identity mappings, making it easier to train deeper networks.

The key advantage of ResNet is its ability to train extremely deep models **without encountering vanishing or exploding gradient problems**, enabling architectures with hundreds or even thousands of layers. As a result, ResNet became a fundamental building block in many state-of-the-art deep learning models.

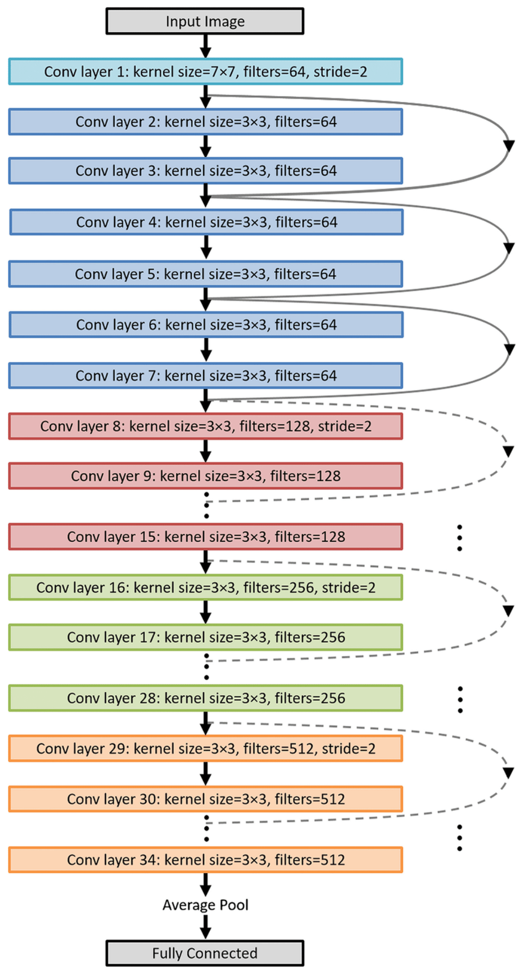
ResNet has several popular versions, such as **ResNet-18**, **ResNet-34**, **ResNet-50**, and others, each varying in depth and complexity. **The ResNet-34 model, with 34 layers, was chosen** for this implementation because it provides a good balance between complexity and performance, making it suitable for training from scratch while still achieving high accuracy.

Architecture Breakdown

The ResNet-34 model consists of several key components:

1. **Initial Convolutional Layer**: The model begins with a standard convolutional layer with a large filter size (usually 7x7), followed by batch normalization and ReLU activation. This layer captures broad features of the input image.
2. **Residual Blocks**: The fundamental building block of ResNet is the residual block. In a residual block, the input is passed through a series of convolutions (typically 3x3 filters) and then added to the output, creating a shortcut connection. This allows the model to learn residual mappings, which has been shown to improve the training of deeper networks.
   - Each residual block has two main layers: convolutional layers, followed by batch normalization and ReLU activation functions.
3. **Strides and Filters**: The strides and filters play a crucial role in controlling the down-sampling and feature extraction process. In ResNet-34, the first convolutional layer has a stride of 2 to reduce the spatial dimensions of the input. Subsequent blocks use a stride of 1 to maintain the resolution, except in certain blocks where down-sampling is required.
4. **Fully Connected Layer**: After passing through the residual blocks, the model ends with a global average pooling layer, which reduces the feature map to a single vector. This is followed by a fully connected layer for classification.

By choosing ResNet-34, we aimed to create a network that is deep enough to capture complex features while being computationally feasible to train from scratch. The combination of residual connections, convolutional layers with specific strides, and 3x3 filters ensures that the model performs well even on more challenging tasks.


Input Image
Conv layer 1: kernel size=7×7, filters=64, stride=2
Conv layer 2: kernel size=3×3, filters=64
Conv layer 3: kernel size=3×3, filters=64
Conv layer 4: kernel size=3×3, filters=64
Conv layer 5: kernel size=3×3, filters=64
Conv layer 6: kernel size=3×3, filters=64
Conv layer 7: kernel size=3×3, filters=64
Conv layer 8: kernel size=3×3, filters=128, stride=2
Conv layer 9: kernel size=3×3, filters=128
Conv layer 15: kernel size=3×3, filters=128
Conv layer 16: kernel size=3×3, filters=256, stride=2
Conv layer 17: kernel size=3×3, filters=256
Conv layer 28: kernel size=3×3, filters=256
Conv layer 29: kernel size=3×3, filters=512, stride=2
Conv layer 30: kernel size=3×3, filters=512
Conv layer 34: kernel size=3×3, filters=512
Average Pool
Fully Connected

The Archtucre

Initial Convolution Layer (Conv1):

- **Kernel Size**: 7x7
- **Number of Filters**: 64
- **Stride**: 2 (Downsampling the input image to 112x112)

Stage 1 (Conv2):

- **Block 1**:
  - 3x3 Convolution with **stride 1**
  - Output size: 56x56
- **Block 2**:
  - 3x3 Convolution with **stride 1**

    - Output size: 56x56
- **Block 3**:
    - 3x3 Convolution with **stride 1**
    - Output size: 56x56
- (No downsampling in this stage, stride = 1)

Stage 2 (Conv3):

- **Block 1**:
    - 3x3 Convolution with **stride 2** (Downsampling)
    - Output size: 28x28
- **Block 2**:
    - 3x3 Convolution with **stride 1**
    - Output size: 28x28
- **Block 3**:
    - 3x3 Convolution with **stride 1**
    - Output size: 28x28
- **Block 4**:
    - 3x3 Convolution with **stride 1**
    - Output size: 28x28
- (Downsampling in the first block, stride = 2)

Stage 3 (Conv4):

- **Block 1**:
    - 3x3 Convolution with **stride 2** (Downsampling)
    - Output size: 14x14
- **Block 2**:
    - 3x3 Convolution with **stride 1**
    - Output size: 14x14
- **Block 3**:
    - 3x3 Convolution with **stride 1**
    - Output size: 14x14
- **Block 4**:
    - 3x3 Convolution with **stride 1**
    - Output size: 14x14
- **Block 5**:
    - 3x3 Convolution with **stride 1**
    - Output size: 14x14
- **Block 6**:
    - 3x3 Convolution with **stride 1**
    - Output size: 14x14
- (Downsampling in the first block, stride = 2)

Stage 4 (Conv5):

- **Block 1**:
    - 3x3 Convolution with **stride 2** (Downsampling)
    - Output size: 7x7
- **Block 2**:
    - 3x3 Convolution with **stride 1**
    - Output size: 7x7
- **Block 3**:
    - 3x3 Convolution with **stride 1**
    - Output size: 7x7
- (Downsampling in the first block, stride = 2)

Final Layers:

- **Global Average Pooling**: The output from the final convolution block is averaged to a 1D vector.
- **Fully Connected Layer (Dense Layer)**: Outputs the final class predictions.

Training Process

The training process for the ResNet-34 model involved several key components, including the selection of appropriate hyperparameters and addressing challenges **and the time took to train was 1 and half hour** using TBU. Below, we outline each of these aspects:

Hyperparameters

The performance of the model is highly influenced by the following hyperparameters:

- **Batch Size**: The batch size used was **32**. A batch size of 32 is commonly used as it allows efficient memory usage while also providing a sufficient amount of data for stable gradient updates during each training step.
- **Number of Epochs**: The model was trained for **20 epochs**. This number was chosen to allow the model to learn sufficiently from the training data while preventing overfitting.
- **Loss Function**: We used **categorical_crossentropy** as the loss function, which is well-suited for multi-class classification problems. This loss function measures the difference between the predicted class probabilities and the true class labels.

Training Challenges

During the training process, several challenges were encountered:

- **Overfitting**: Overfitting occurred as the model's training accuracy improved significantly, but the validation accuracy plateaued. To mitigate this, **data augmentation** was applied (as described previously), which helped the model generalize better to unseen data.
    - **Why**? Because the data is not really have diversity and actually maybe yes it is large but it's simple not hard colors or edges that what make the overfitting and you can say it is overkill because the model is so complex for the simple data and even after using the data augmentation the change was very little

Pros and cons in General

Pros:

1. **Residual Connections**:
    - ResNet introduces residual connections (skip connections), which allow the model to learn identity functions. These connections help prevent the vanishing gradient problem by allowing gradients to flow more easily through the network, enabling the training of deeper models without performance degradation.
2. **Better Performance with Depth**:
    - One of the major advantages of ResNet is its ability to maintain or even improve performance as the network depth increases. Traditional neural networks struggle with very deep architectures due to vanishing gradients, but ResNet overcomes this with its residual connections.
3. **Efficient Training**:

- The use of residual connections helps in training deeper networks more efficiently. With ResNet, deeper models such as ResNet-50 or ResNet-101 can be trained with relatively better performance and less training time compared to other architectures of similar depth.

4. **Versatility**:
   - ResNet can be applied to a variety of tasks, such as image classification, object detection, and semantic segmentation. It has been used in many state-of-the-art systems and has demonstrated high generalization capabilities.
5. **Pretrained Models**:
   - ResNet provides access to pretrained models, which can be fine-tuned on specific datasets, leading to faster training times and better performance in transfer learning scenarios.
6. **Scalability**:
   - ResNet's design makes it scalable. It can be used to create models of varying depths, such as ResNet-18, ResNet-34, ResNet-50, ResNet-101, and even deeper versions, giving it flexibility for different problem complexities.

Cons:

1. **Increased Computational Cost**:
   - While ResNet allows for deeper architectures, this also comes with increased computational demands. The need for more resources (memory and processing power) becomes more evident with very deep models, especially for large-scale datasets.
2. **Training Complexity**:
   - Despite its advantages, training very deep ResNet models (e.g., ResNet-101 or ResNet-152) can be computationally expensive and may require specialized hardware like GPUs or TPUs. Additionally, deeper models can still suffer from issues like overfitting on small datasets.
3. **Potential Overfitting**:
   - Although ResNet performs well on large datasets, it can still overfit on smaller datasets. This is particularly problematic when the model is too deep for the dataset or if data augmentation techniques aren't applied effectively.
4. **Diminishing Returns**:
   - While deeper versions of ResNet can improve performance, the gains often become less significant beyond a certain depth. For example, going from ResNet-34 to ResNet-50 provides notable improvements, but moving from ResNet-50 to ResNet-101 or ResNet-152 may not always show drastic improvements in performance.
5. **Complexity in Hyperparameter Tuning**:
   - The ResNet architecture can sometimes be tricky to tune due to the various hyperparameters involved, such as the number of blocks, learning rates, and batch sizes, especially for deeper networks.

Pros Faced During Implementation:

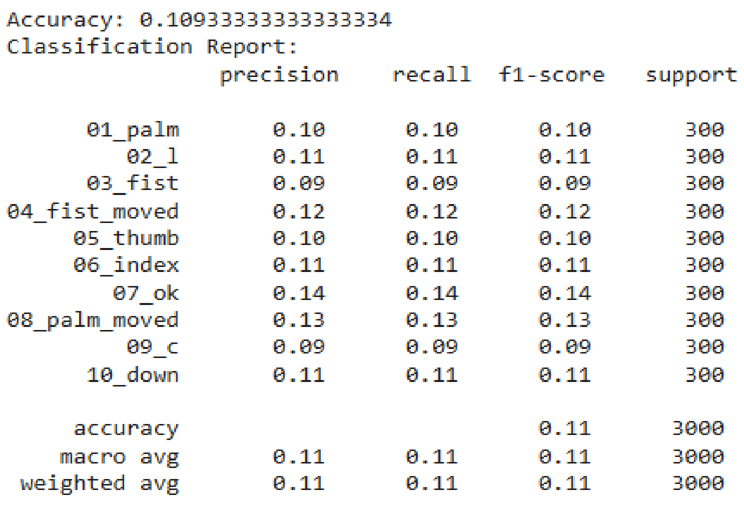
1. **Improved Accuracy**:
   - The use of the ResNet-34 architecture helped the model quickly learn features in the data, resulting in good performance during training and high initial accuracy.
2. **Efficient Use of Layers**:
   - The model was able to benefit from deeper layers without encountering issues like vanishing gradients due to the residual connections. This allowed the model to generalize better compared to other shallow architectures.
3. **Ease of Implementation**:
   - ResNet, especially in frameworks like TensorFlow and Keras, is relatively easy to implement due to the availability of pretrained models and the efficient implementation of residual blocks.

Cons Faced During Implementation:

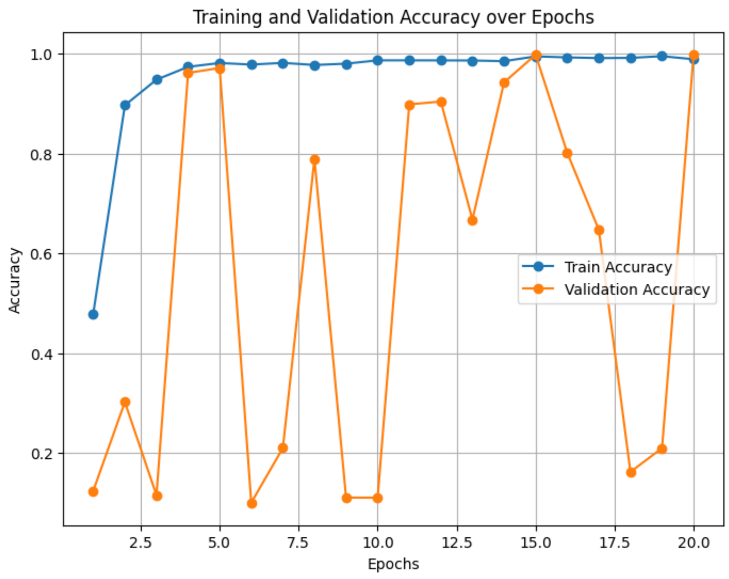
1. **Overfitting**:
   - One of the biggest challenges encountered was overfitting, as the model performed significantly better on the training dataset compared to the validation dataset. This was addressed by applying data augmentation and regularization techniques, but it still posed challenges in achieving high validation accuracy.
2. **Resource Consumption**:
   - Although ResNet-34 is relatively smaller compared to deeper versions, the training process still consumed significant computational resources, especially when working with large datasets. Training times were long, and the process required access to GPUs to complete efficiently.
3. **Difficulty with Validation Loss**:
   - While the training accuracy improved steadily, the validation loss did not decrease as expected. This suggests that the model may have learned to fit the training data too closely, indicating potential issues like underfitting or not enough regularization.

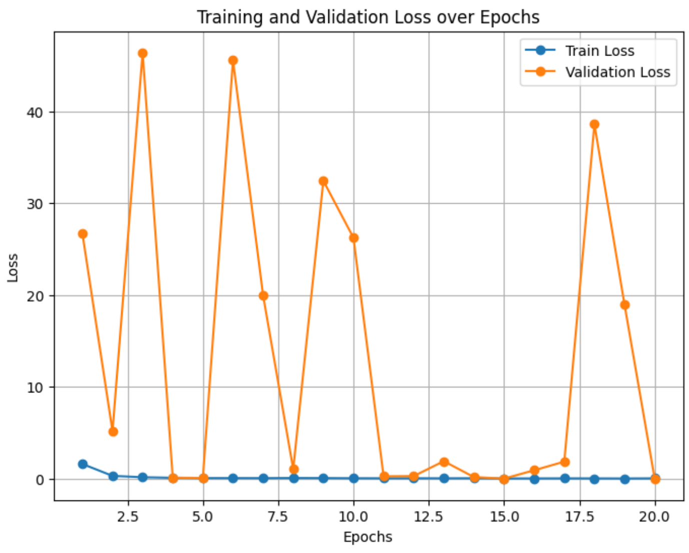
Evaluation

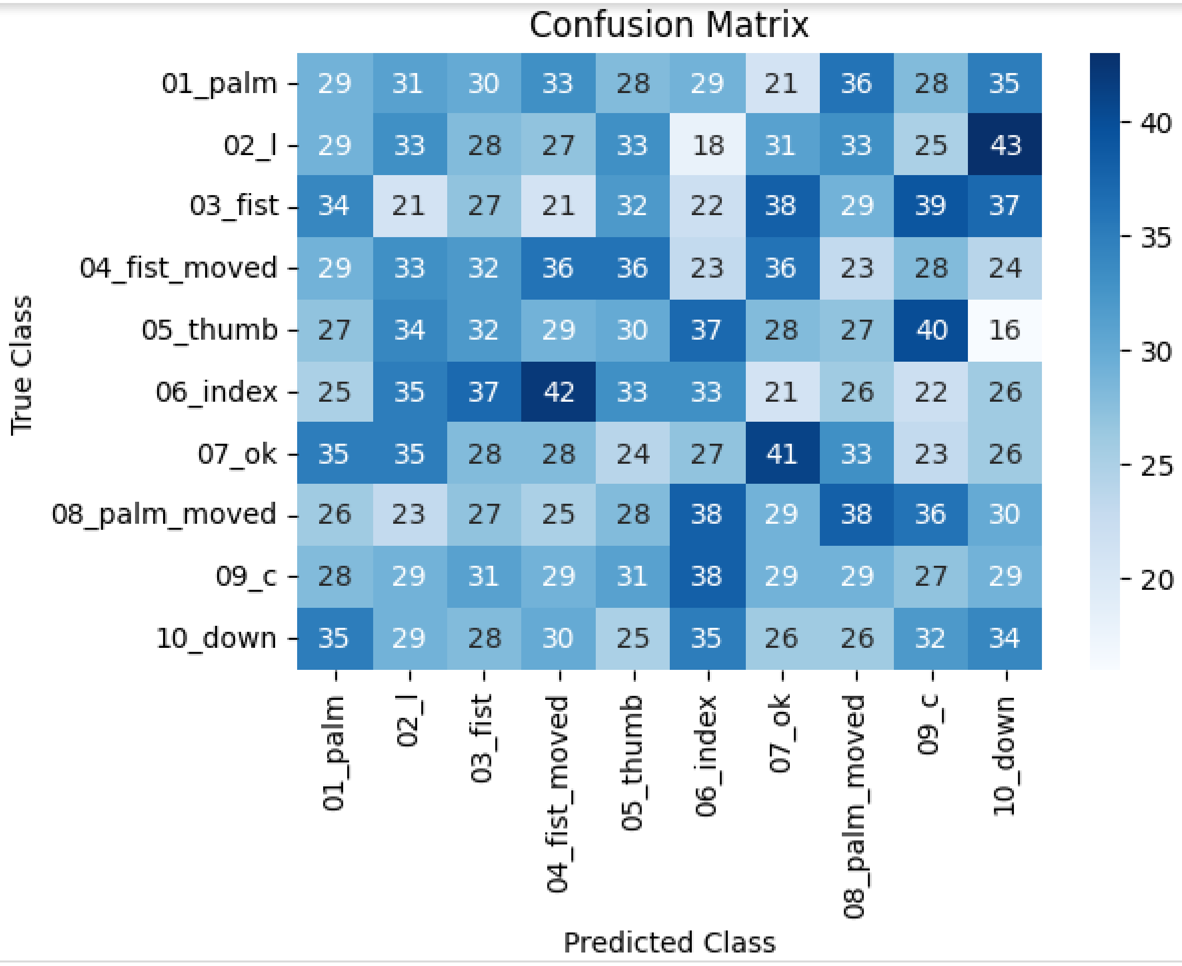
accuracy and classification Report



```
Accuracy: 0.10933333333333334
Classification Report:
                precision    recall  f1-score   support

      01_palm       0.10      0.10      0.10       300
         02_l       0.11      0.11      0.11       300
      03_fist       0.09      0.09      0.09       300
04_fist_moved       0.12      0.12      0.12       300
     05_thumb       0.10      0.10      0.10       300
     06_index       0.11      0.11      0.11       300
        07_ok       0.14      0.14      0.14       300
08_palm_moved       0.13      0.13      0.13       300
         09_c       0.09      0.09      0.09       300
      10_down       0.11      0.11      0.11       300

     accuracy                           0.11      3000
    macro avg       0.11      0.11      0.11      3000
 weighted avg       0.11      0.11      0.11      3000
```

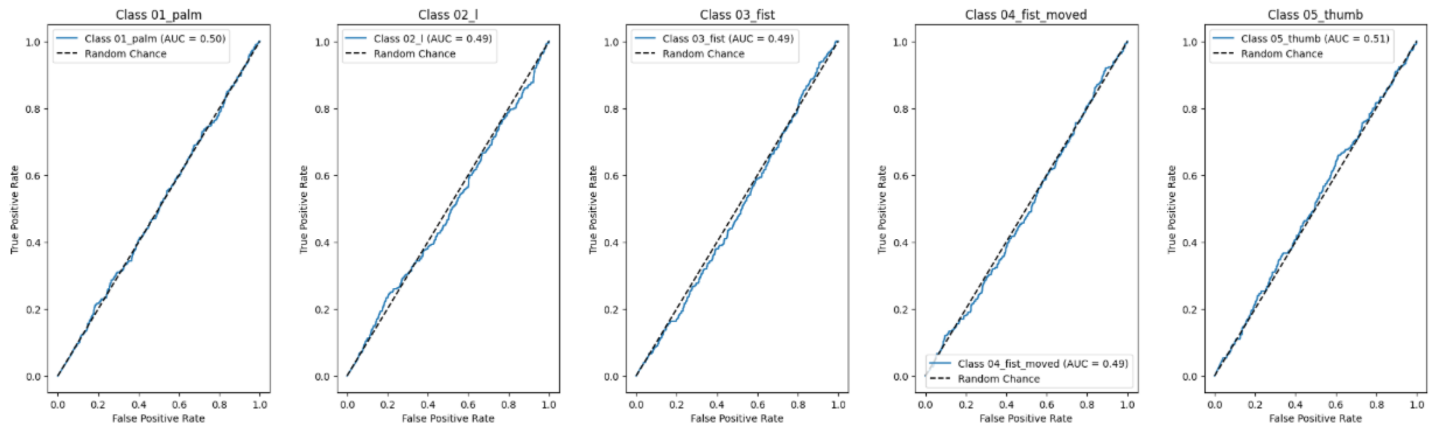




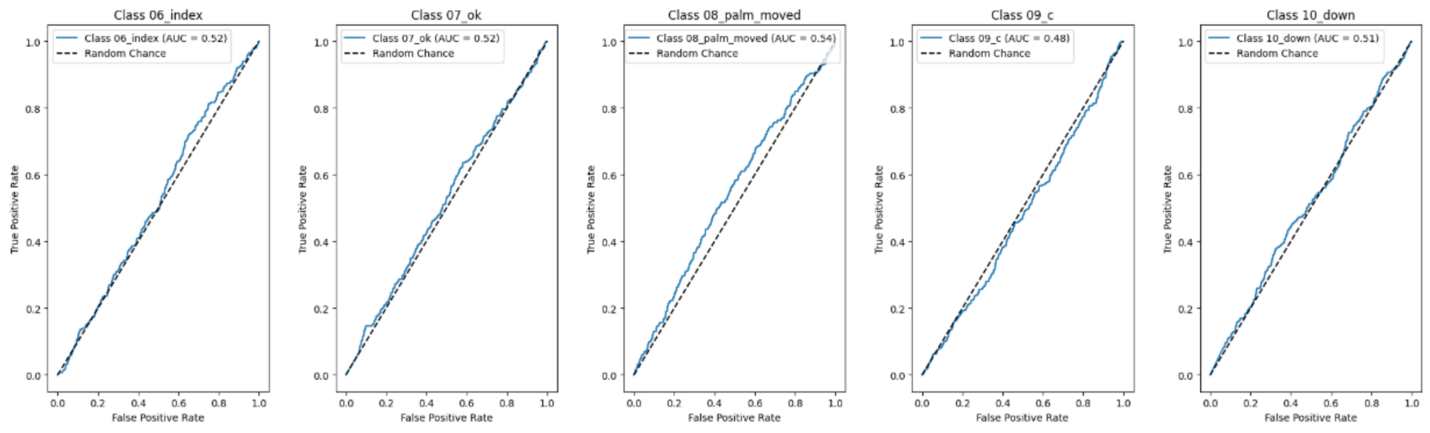


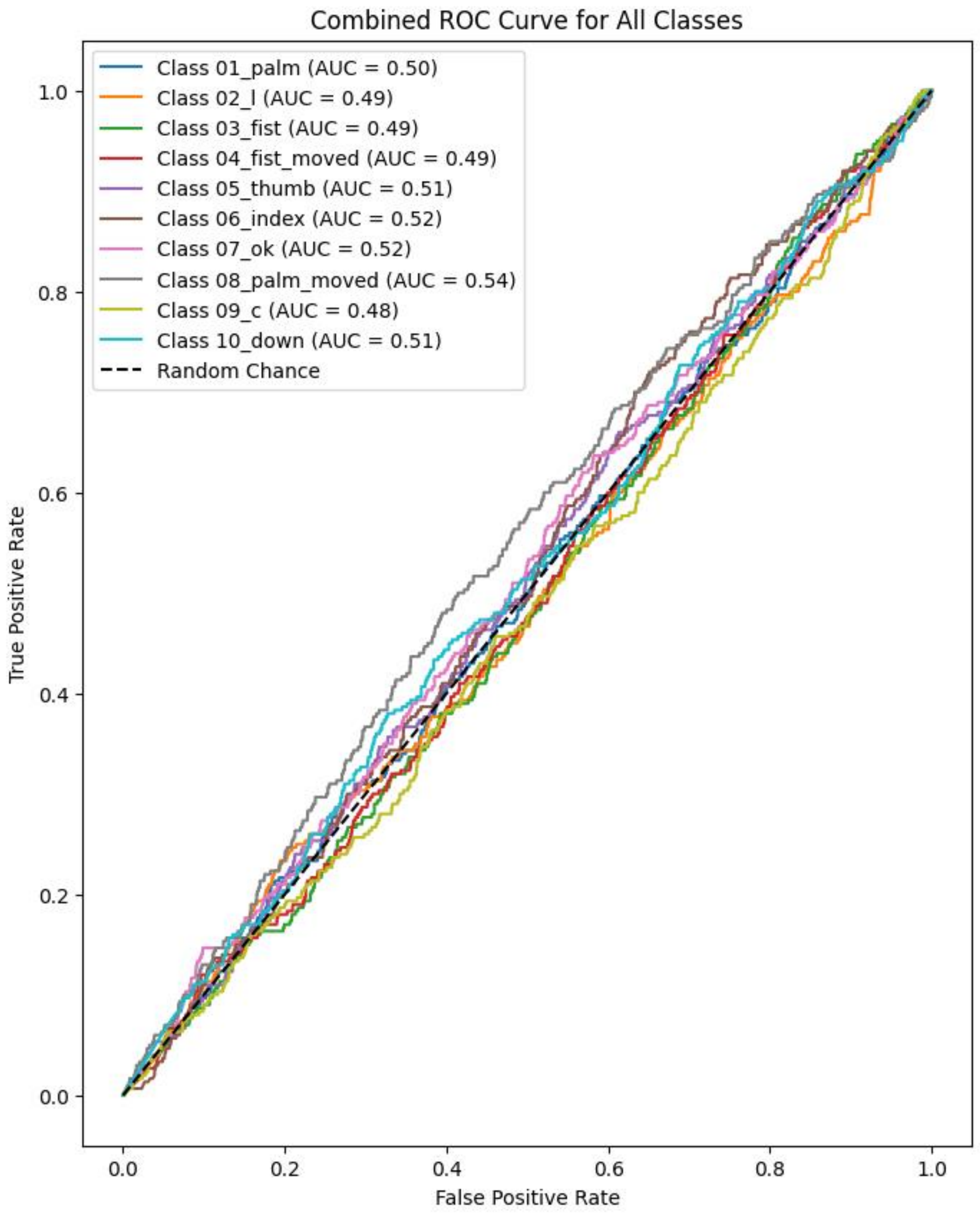
True Class \ Predicted Class	01_palm	02_l	03_fist	04_fist_moved	05_thumb	06_index	07_ok	08_palm_moved	09_c	10_down
01_palm	29	31	30	33	28	29	21	36	28	35
02_l	29	33	28	27	33	18	31	33	25	43
03_fist	34	21	27	21	32	22	38	29	39	37
04_fist_moved	29	33	32	36	36	23	36	23	28	24
05_thumb	27	34	32	29	30	37	28	27	40	16
06_index	25	35	37	42	33	33	21	26	22	26
07_ok	35	35	28	28	24	27	41	33	23	26
08_palm_moved	26	23	27	25	28	38	29	38	36	30
09_c	28	29	31	29	31	38	29	29	27	29
10_down	35	29	28	30	25	35	26	26	32	34

ROC AUC


Class 01_palm
Class 01_palm (AUC = 0.50)
Random Chance
Class 02_l
Class 02_l (AUC = 0.49)
Random Chance
Class 03_fist
Class 03_fist (AUC = 0.49)
Random Chance
Class 04_fist_moved
Class 04_fist_moved (AUC = 0.49)
Random Chance
Class 05_thumb
Class 05_thumb (AUC = 0.51)
Random Chance
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

m


Class 06_index
Class 06_index (AUC = 0.52)
Random Chance
Class 07_ok
Class 07_ok (AUC = 0.52)
Random Chance
Class 08_palm_moved
Class 08_palm_moved (AUC = 0.54)
Random Chance
Class 09_c
Class 09_c (AUC = 0.48)
Random Chance
Class 10_down
Class 10_down (AUC = 0.51)
Random Chance
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

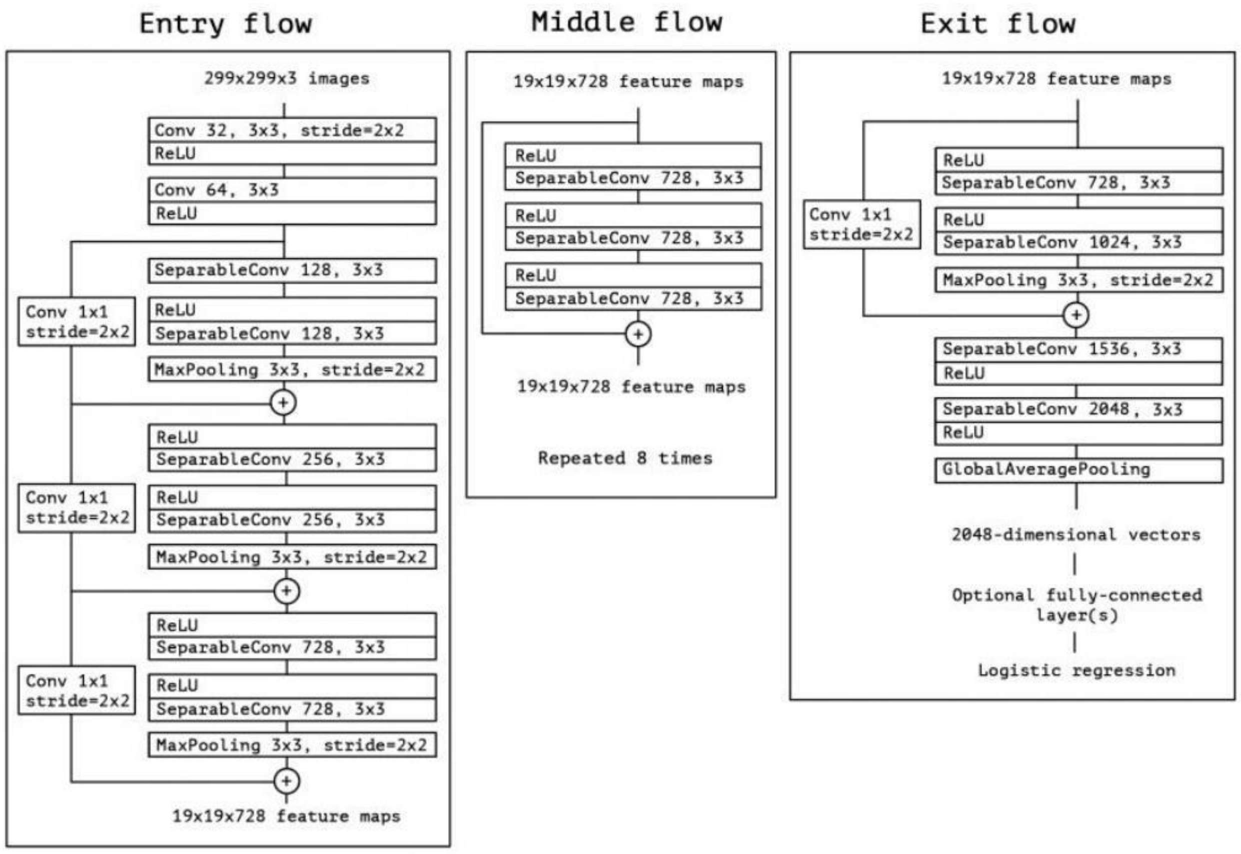

Combined ROC Curve for All Classes
Class 01_palm (AUC = 0.50)
Class 02_l (AUC = 0.49)
Class 03_fist (AUC = 0.49)
Class 04_fist_moved (AUC = 0.49)
Class 05_thumb (AUC = 0.51)
Class 06_index (AUC = 0.52)
Class 07_ok (AUC = 0.52)
Class 08_palm_moved (AUC = 0.54)
Class 09_c (AUC = 0.48)
Class 10_down (AUC = 0.51)
Random Chance
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

XceptionModel

Xception, short for "Extreme Inception," is a deep convolutional neural network architecture introduced by Francois Chollet in 2016. It builds upon the Inception architecture, sharing some design principles but introducing depthwise separable convolutions, which distinguish it as more computationally efficient. When compared to ResNet or standard Inception, Xception typically offers improved computational efficiency and comparable or better accuracy in various image recognition tasks, while maintaining a simpler and more modular structure. It builds upon the Inception architecture but replaces Inception modules with depthwise separable convolutions, making the model both more efficient and performant.

Architecture

1. **Depthwise Separable Convolutions**: Xception leverages depthwise separable convolutions, which decompose a standard convolution into:
    - A depthwise convolution: Applies a single filter per input channel.
    - A pointwise convolution: Combines the outputs of depthwise convolutions.
2. **Linear Stack**: Unlike Inception, Xception arranges layers in a straightforward sequential structure, simplifying implementation.
3. **Global Average Pooling**: Removes fully connected layers and replaces them with a global average pooling layer for better generalization and reduced overfitting.



Pros

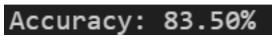
- **Efficient Computations**: Depthwise separable convolutions significantly reduce the computational cost compared to traditional convolutions.
- **High Performance**: Demonstrates strong results across various classification and detection tasks.
- **Simplified Design**: Replaces the complex Inception modules with a clean, modular architecture.

Cons

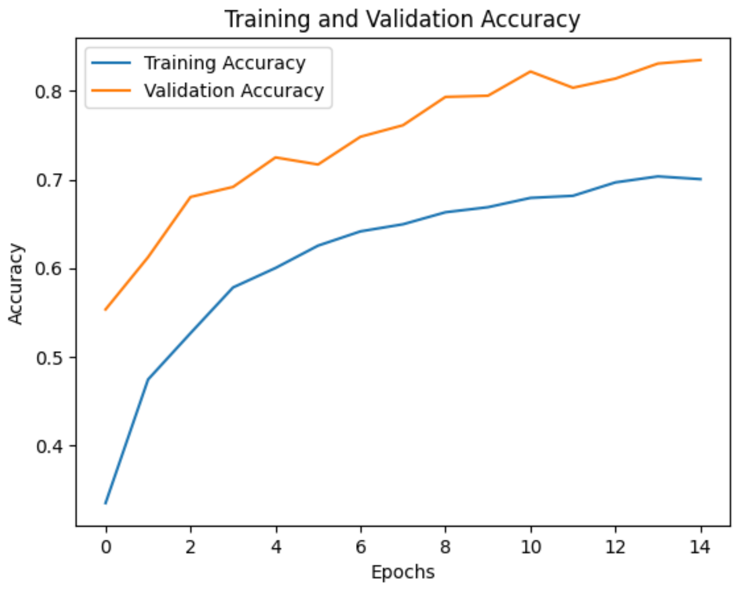
- **Sensitive to Hyperparameters**: Performance heavily depends on proper initialization and tuning. Common hyperparameters to consider include the learning rate, weight decay, batch size, and the number of epochs. Additionally, optimizer selection (e.g., Adam or SGD) and dropout rate may require fine-tuning to achieve optimal results.
- **Training Complexity**: Requires careful data preprocessing and augmentation for optimal results.
- **Less Robust for Small Datasets**: May overfit when used on small datasets without proper regularization.

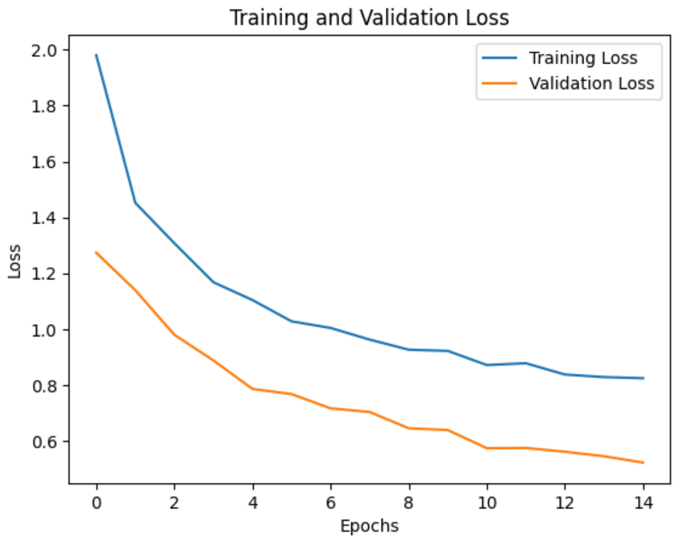
- ## Evaluation

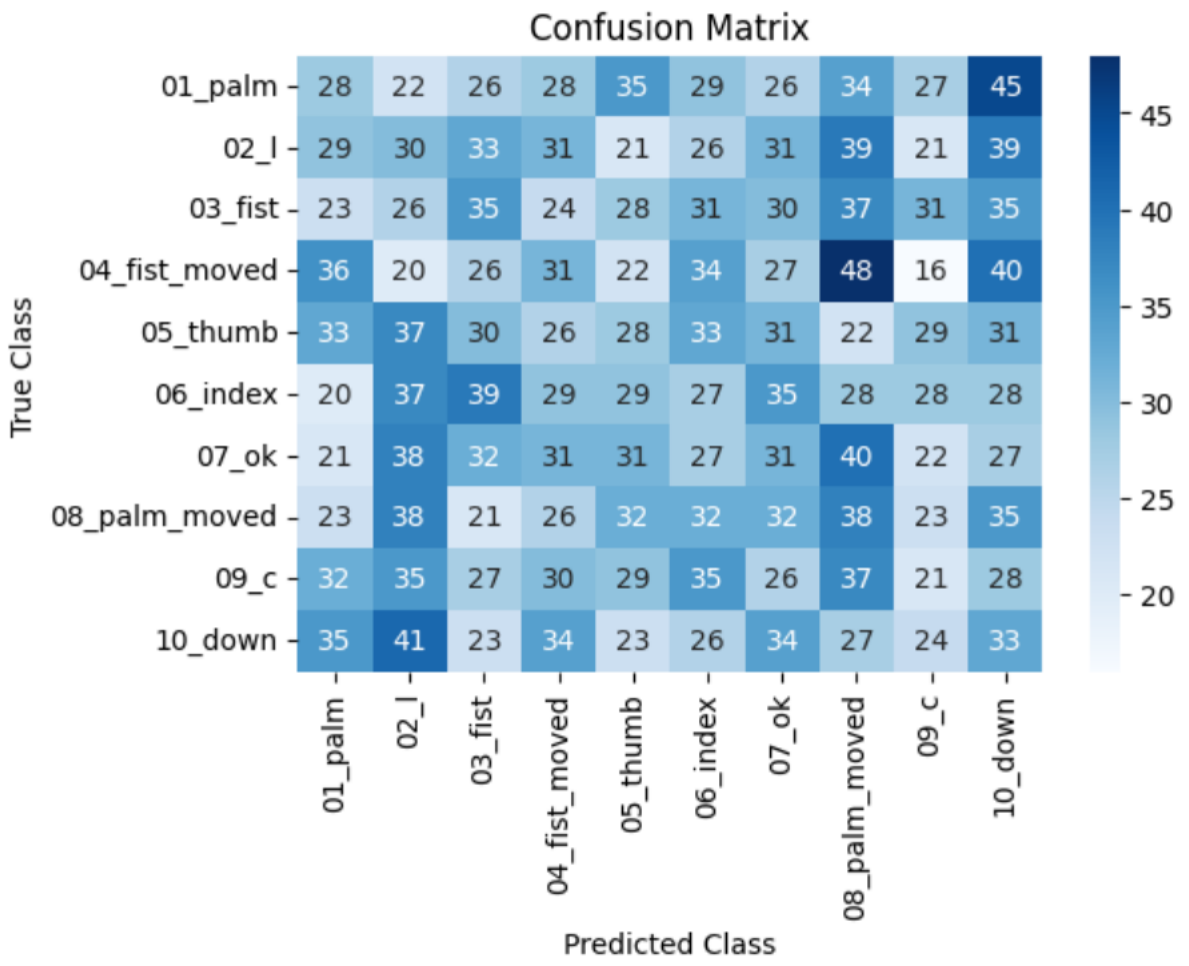
  accuracy and classification Report



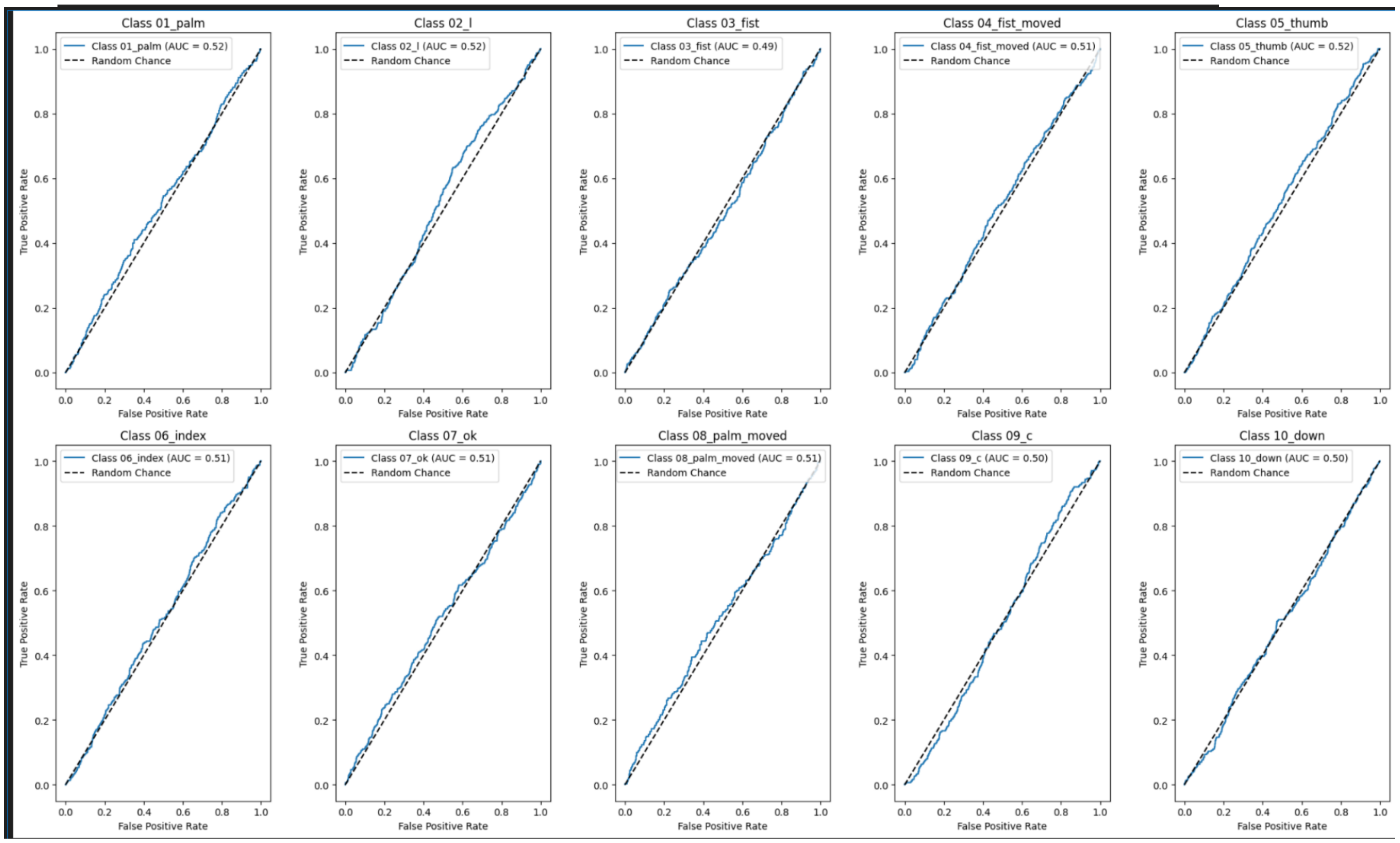
```
Accuracy: 83.50%
```

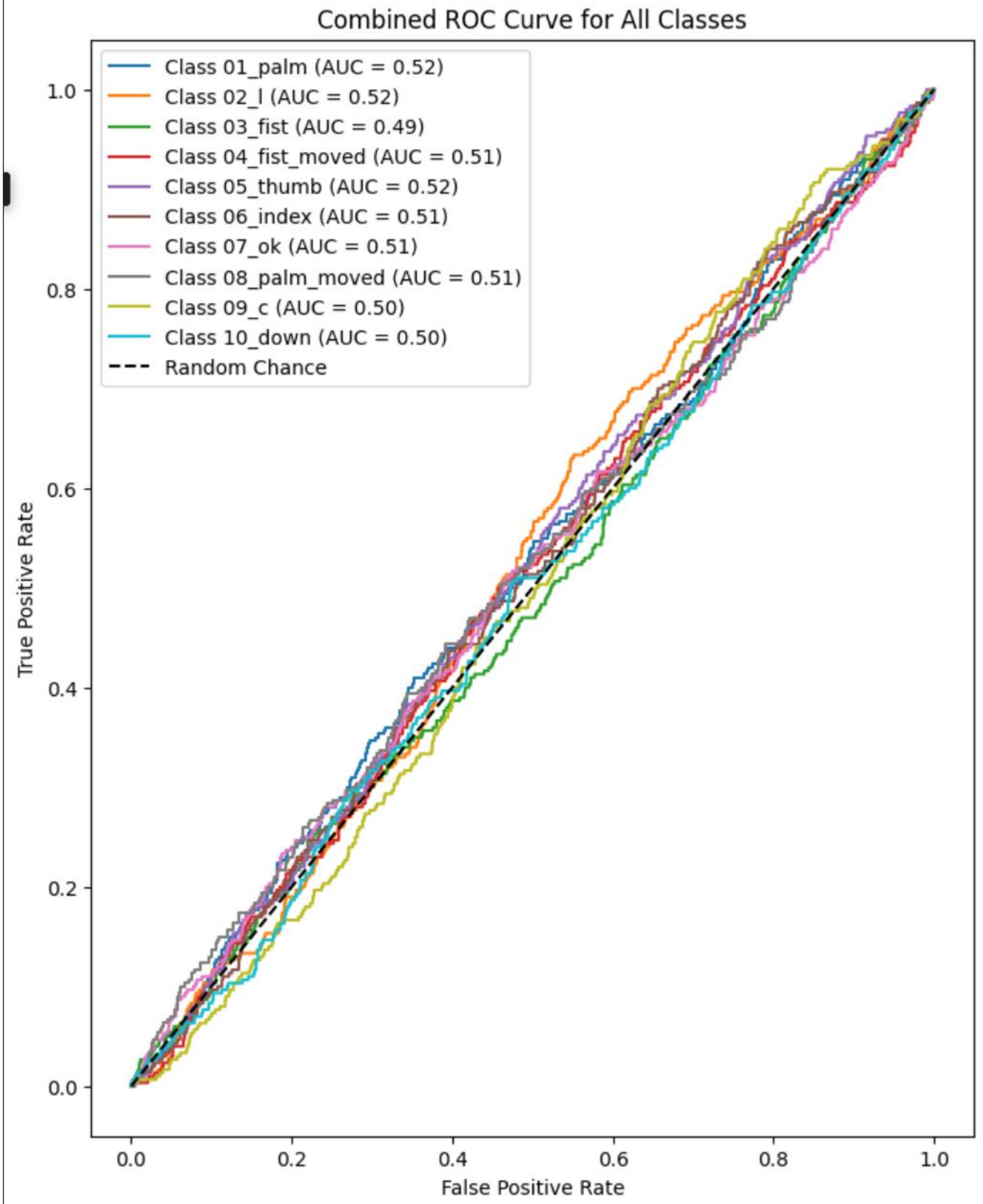






True Class \ Predicted Class	01_palm	02_l	03_fist	04_fist_moved	05_thumb	06_index	07_ok	08_palm_moved	09_c	10_down
01_palm	28	22	26	28	35	29	26	34	27	45
02_l	29	30	33	31	21	26	31	39	21	39
03_fist	23	26	35	24	28	31	30	37	31	35
04_fist_moved	36	20	26	31	22	34	27	48	16	40
05_thumb	33	37	30	26	28	33	31	22	29	31
06_index	20	37	39	29	29	27	35	28	28	28
07_ok	21	38	32	31	31	27	31	40	22	27
08_palm_moved	23	38	21	26	32	32	32	38	23	35
09_c	32	35	27	30	29	35	26	37	21	28
10_down	35	41	23	34	23	26	34	27	24	33




Combined ROC Curve for All Classes
Class 01_palm (AUC = 0.52)
Class 02_l (AUC = 0.52)
Class 03_fist (AUC = 0.49)
Class 04_fist_moved (AUC = 0.51)
Class 05_thumb (AUC = 0.52)
Class 06_index (AUC = 0.51)
Class 07_ok (AUC = 0.51)
Class 08_palm_moved (AUC = 0.51)
Class 09_c (AUC = 0.50)
Class 10_down (AUC = 0.50)
Random Chance
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

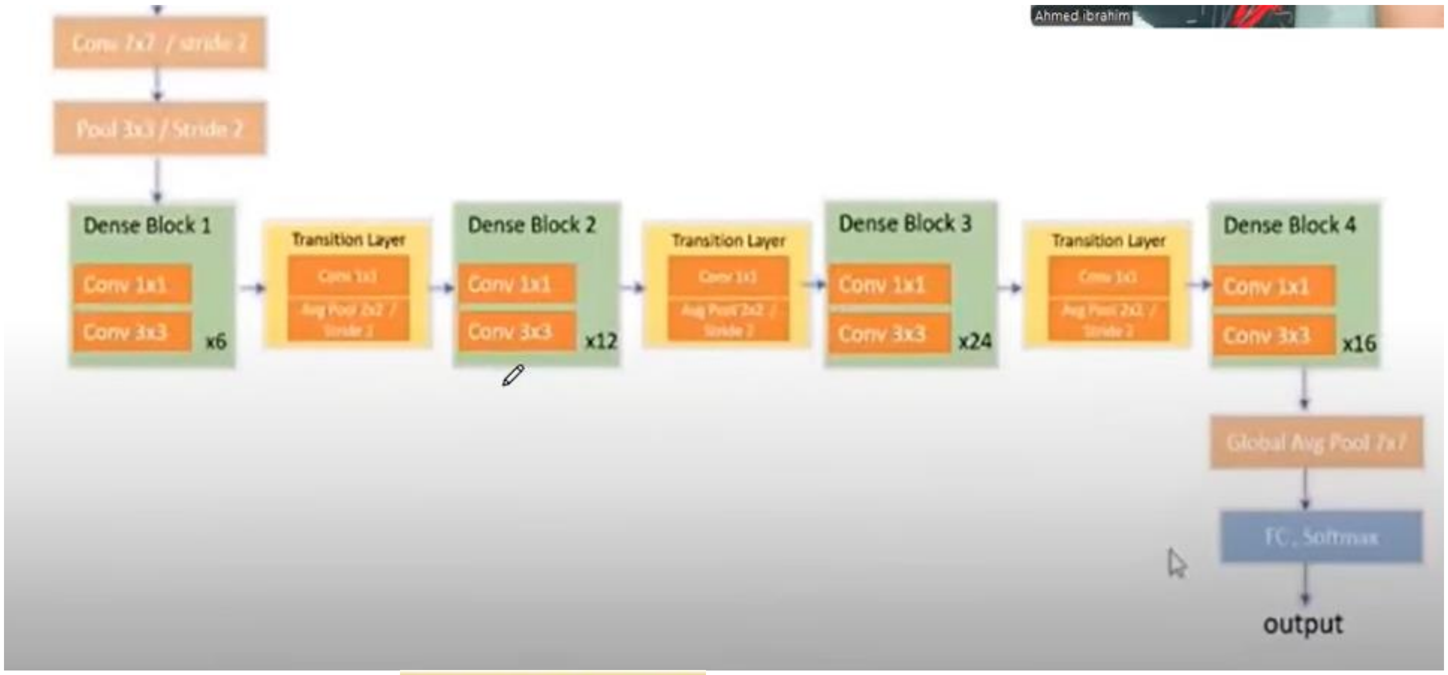
DenseNet Model

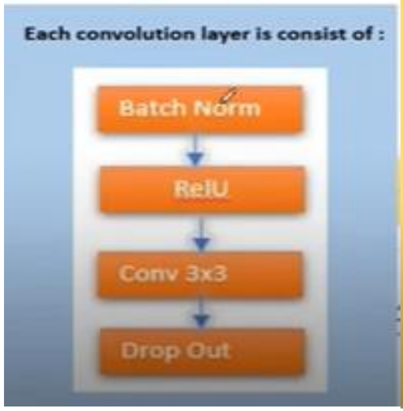
Overview

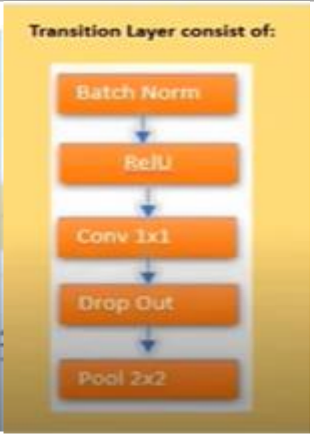
DenseNet (Densely Connected Convolutional Network) is a neural network architecture introduced by Huang et al. in 2017. DenseNet has shown notable advantages in various domains, particularly in medical imaging, where its efficient feature reuse and strong gradient flow have been leveraged for tasks such as disease diagnosis and segmentation. It is also well-suited for other applications requiring detailed pattern recognition, such as remote sensing and biometrics. It connects each layer to every other layer in a dense block, promoting feature reuse and efficient gradient flow.

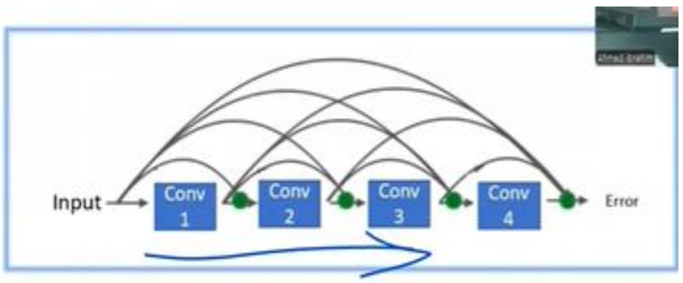
Architecture

1. **Dense Connections**: Each layer receives the feature maps of all preceding layers as input, ensuring maximum information flow. This feature is particularly beneficial for tasks with limited data or high-dimensional inputs, as it allows the model to efficiently utilize features from all layers and improves generalization.
2. **Dense Blocks**: Groups of densely connected layers where each layer outputs a fixed number of feature maps ("growth rate").
3. **Transition Layers**: Reduces dimensionality using convolutions and pooling layers, controlling model complexity.
4. **Global Average Pooling**: Like Xception, it replaces fully connected layers with global average pooling.





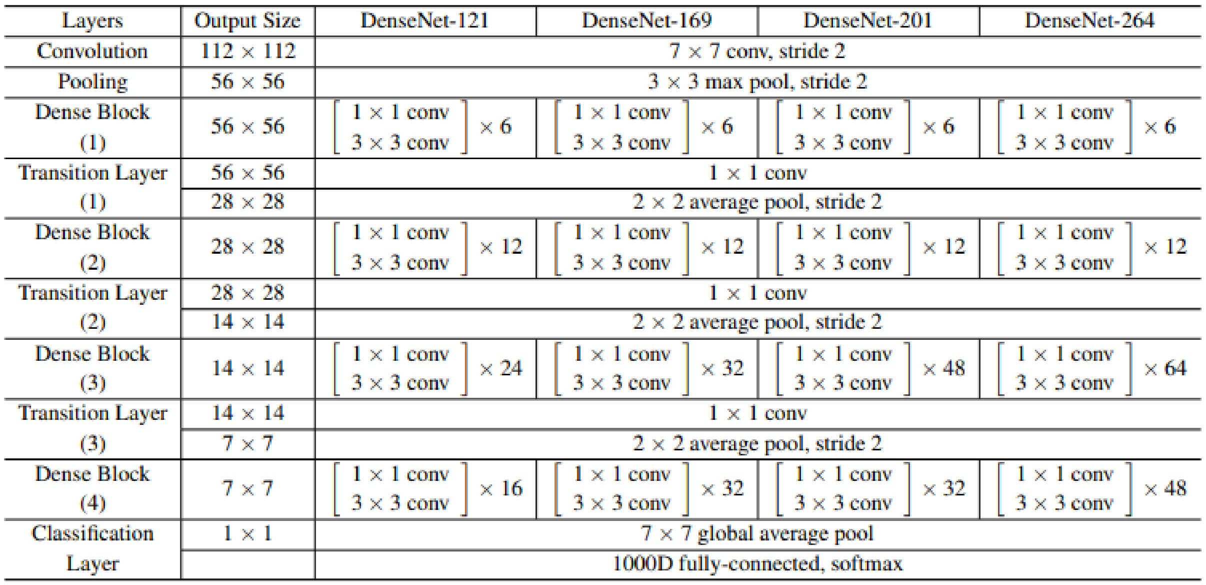




Types of DenseNet

DenseNet comes in several variants based on the number of layers and complexity:

1. **DenseNet-121**: A lightweight version with 121 layers, suitable for tasks requiring a balance between accuracy and computational efficiency.
2. **DenseNet-169**: Deeper than DenseNet-121, with 169 layers, providing improved performance but requiring more resources.
3. **DenseNet-201**: Offers 201 layers, designed for tasks where higher accuracy is prioritized over efficiency.
4. **DenseNet-264**: The deepest standard version with 264 layers, delivering top-tier accuracy but demanding significant computational and memory resources.



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Pros

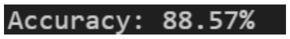
- **Efficient Feature Usage**: Reuse of features allows DenseNet to achieve better performance with fewer parameters.
- **Improved Gradient Flow**: Dense connections mitigate the vanishing gradient problem.
- **Compact Model**: Reduces the risk of overfitting due to efficient parameter usage.

Cons

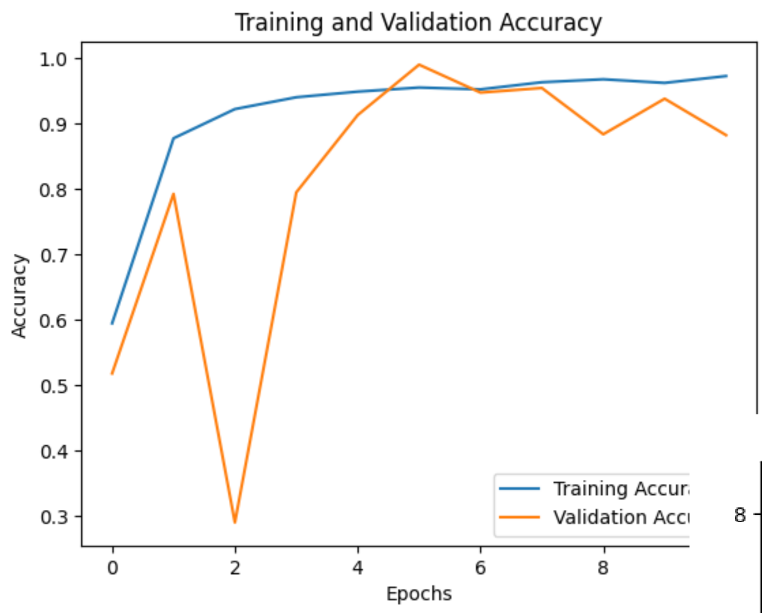
- **High Memory Usage**: Dense connections require storing feature maps from all previous layers, increasing memory demand. To mitigate this, strategies such as gradient checkpointing, mixed precision training, or reducing the batch size can be employed to optimize memory usage during training.
- **Slower Training**: Dense connectivity increases computation time during training.
- **Not Suitable for Very Large Datasets**: The computational cost may become prohibitive for extremely large datasets.

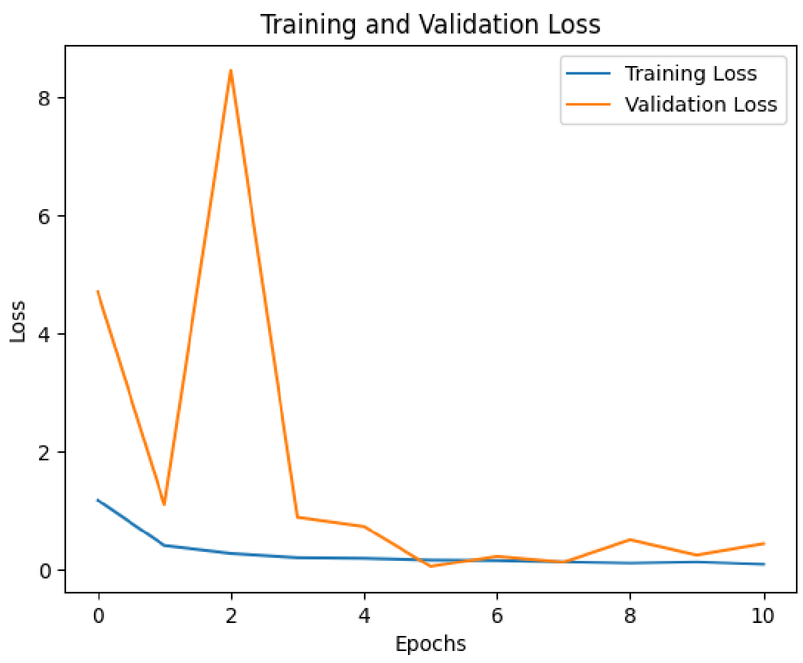
Evaluation

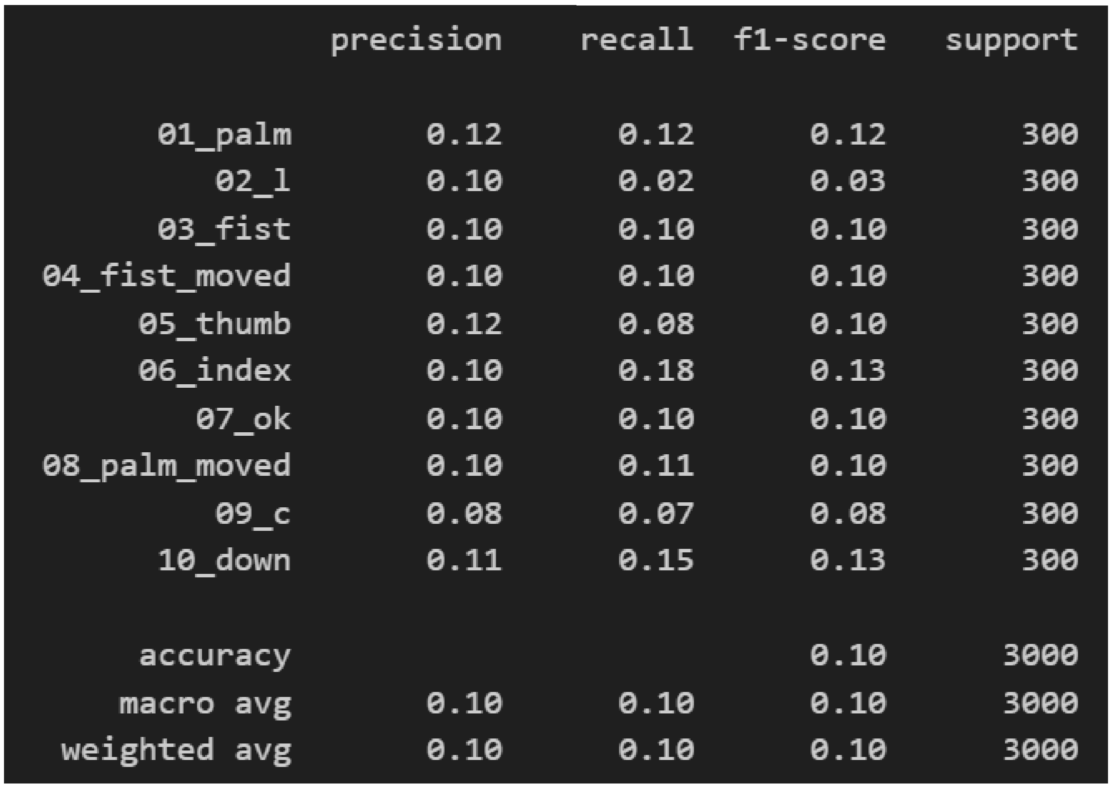
accuracy and classification Report



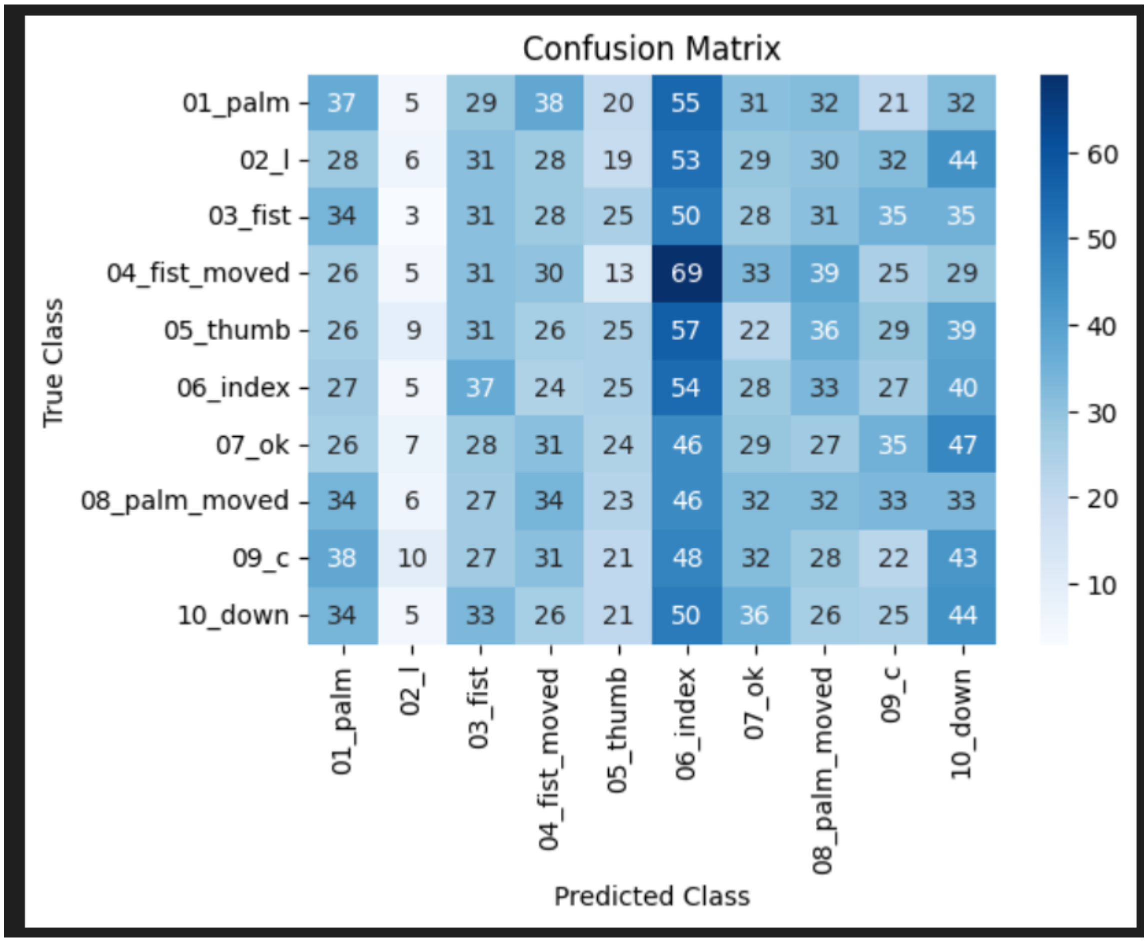
```
Accuracy: 88.57%
```



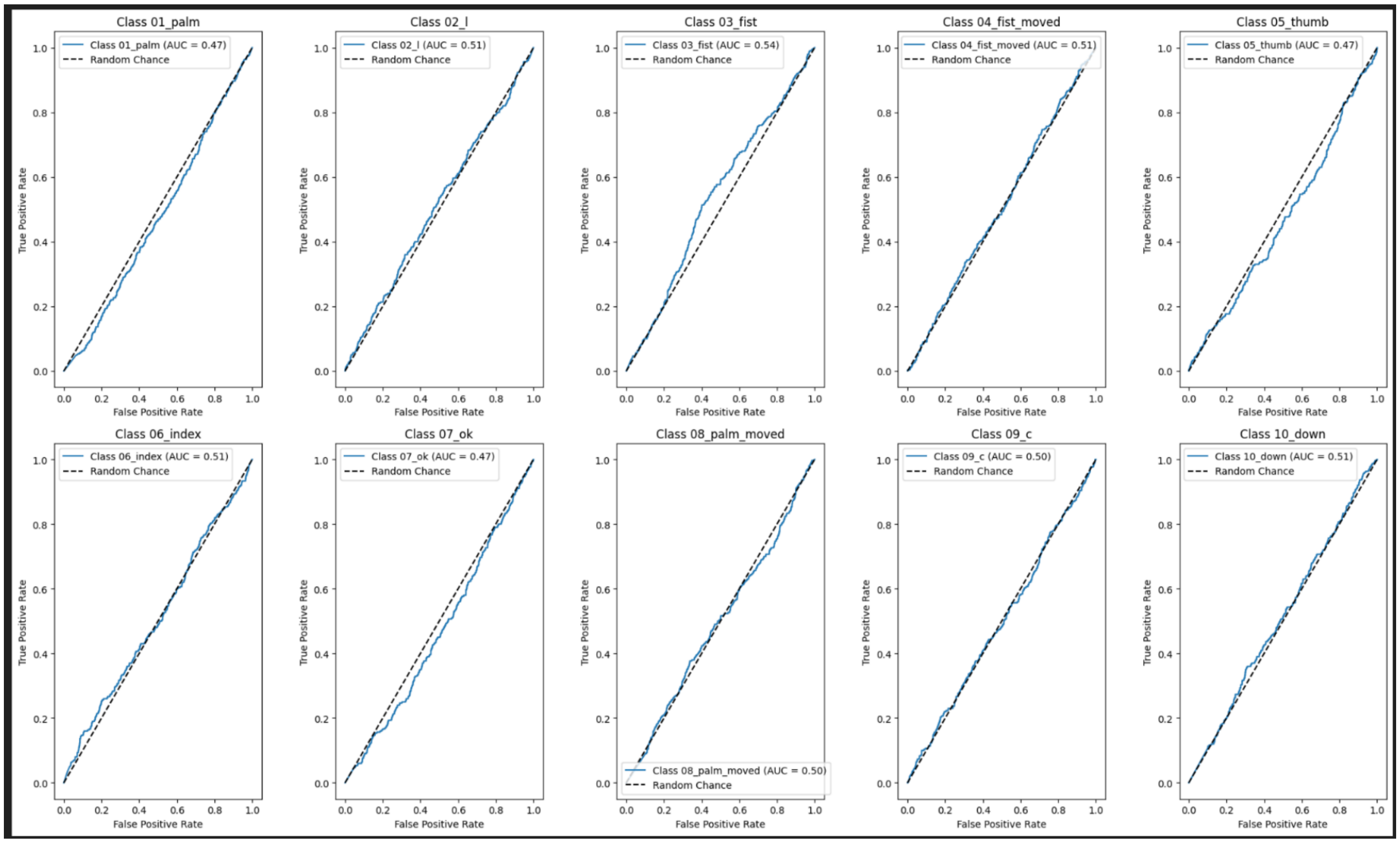


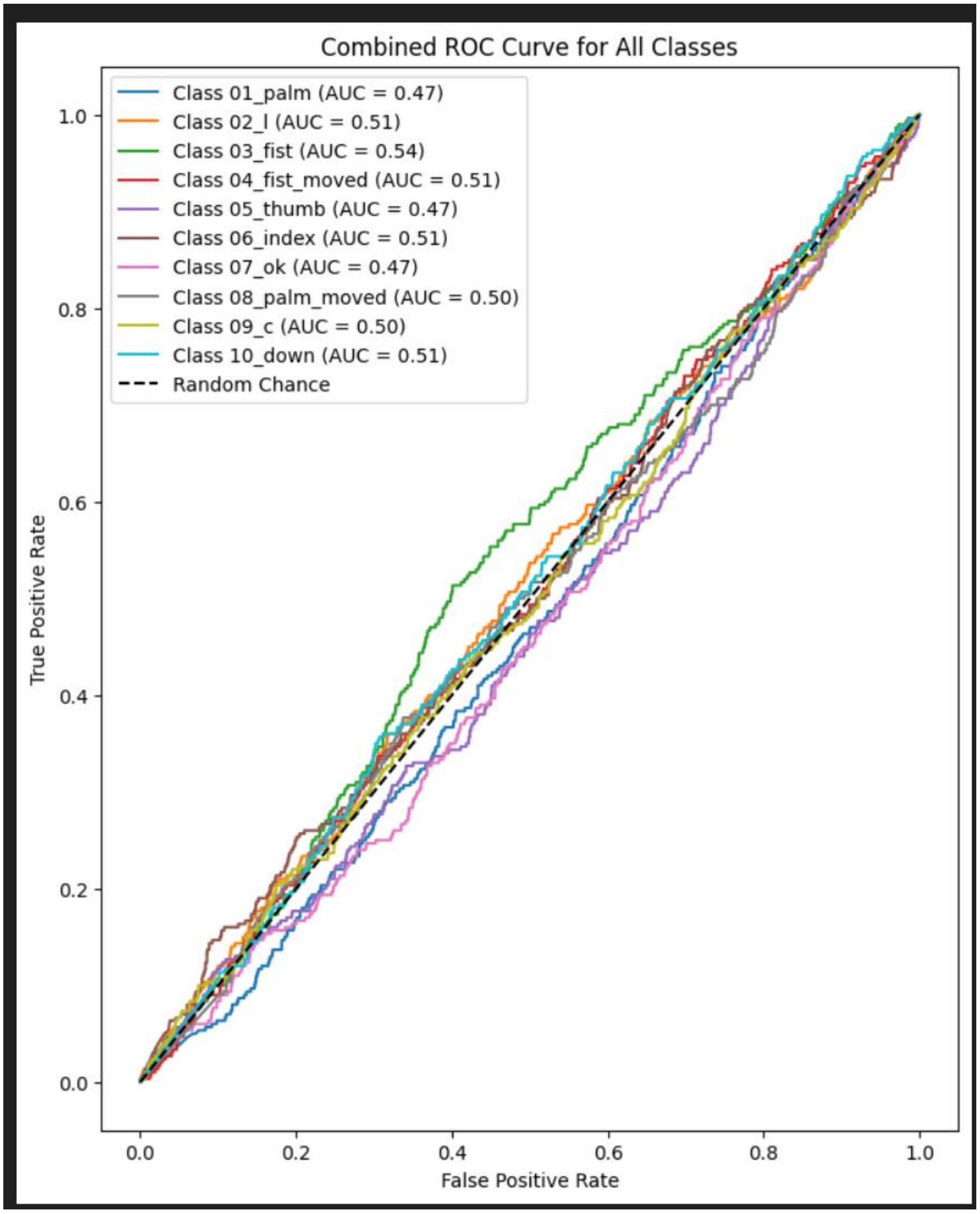


```
              precision    recall  f1-score   support

     01_palm       0.12      0.12      0.12       300
        02_l       0.10      0.02      0.03       300
     03_fist       0.10      0.10      0.10       300
04_fist_moved       0.10      0.10      0.10       300
    05_thumb       0.12      0.08      0.10       300
    06_index       0.10      0.18      0.13       300
       07_ok       0.10      0.10      0.10       300
08_palm_moved       0.10      0.11      0.10       300
        09_c       0.08      0.07      0.08       300
     10_down       0.11      0.15      0.13       300

    accuracy                           0.10      3000
   macro avg       0.10      0.10      0.10      3000
weighted avg       0.10      0.10      0.10      3000
```



True Class \ Predicted Class	01_palm	02_l	03_fist	04_fist_moved	05_thumb	06_index	07_ok	08_palm_moved	09_c	10_down
01_palm	37	5	29	38	20	55	31	32	21	32
02_l	28	6	31	28	19	53	29	30	32	44
03_fist	34	3	31	28	25	50	28	31	35	35
04_fist_moved	26	5	31	30	13	69	33	39	25	29
05_thumb	26	9	31	26	25	57	22	36	29	39
06_index	27	5	37	24	25	54	28	33	27	40
07_ok	26	7	28	31	24	46	29	27	35	47
08_palm_moved	34	6	27	34	23	46	32	32	33	33
09_c	38	10	27	31	21	48	32	28	22	43
10_down	34	5	33	26	21	50	36	26	25	44



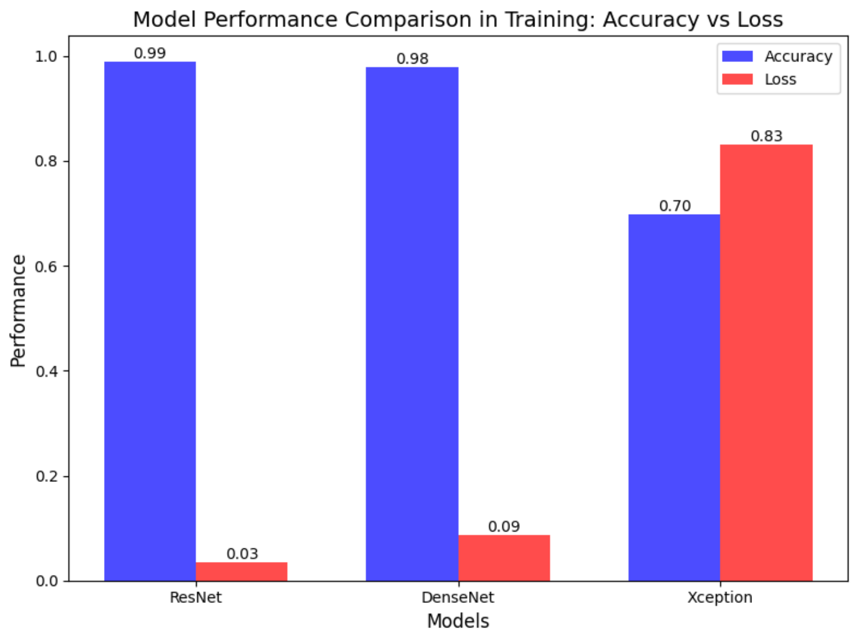

Combined ROC Curve for All Classes
Class 01_palm (AUC = 0.47)
Class 02_l (AUC = 0.51)
Class 03_fist (AUC = 0.54)
Class 04_fist_moved (AUC = 0.51)
Class 05_thumb (AUC = 0.47)
Class 06_index (AUC = 0.51)
Class 07_ok (AUC = 0.47)
Class 08_palm_moved (AUC = 0.50)
Class 09_c (AUC = 0.50)
Class 10_down (AUC = 0.51)
Random Chance
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

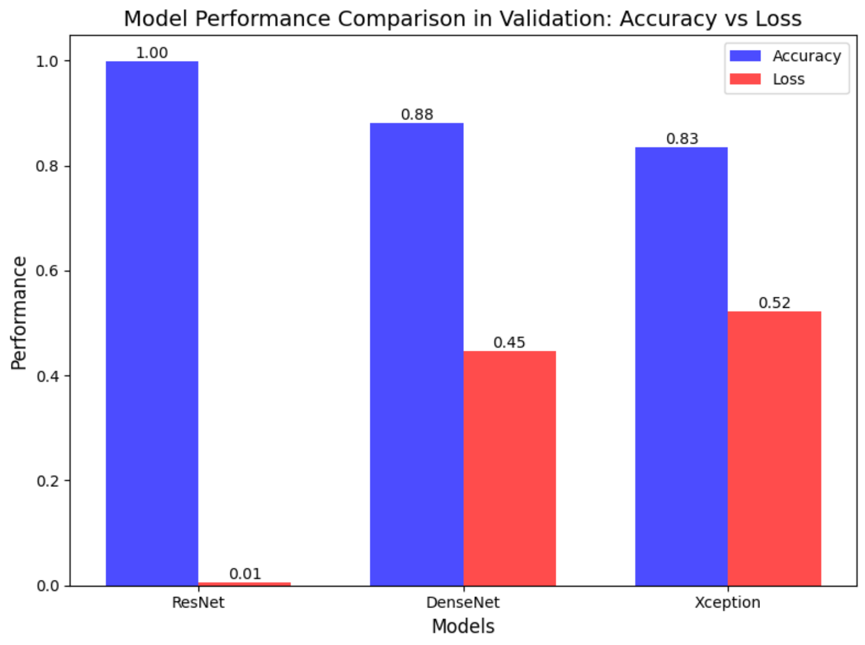
Densenet VS Resnet Vs Xception

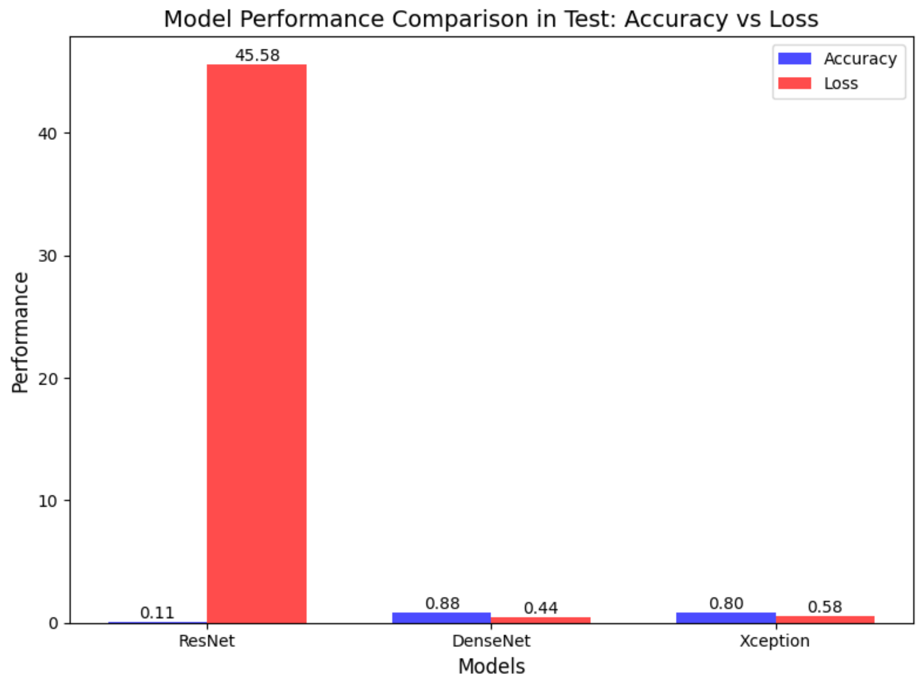
	Resnet	Densenet	Xception
Train accuracy	98%	97%	69%
Val accuracy	99% not stable	88%	83%
Test accuracy	10%	88%	79%
Advantage on the data	High capacity to learn patterns	Faster convergence and higher training accuracy.	More stable generalization, less prone to overfitting, and validation accuracy steadily improved.
Disadvantage on the data	Severe overfitting and instability	Overfitting is evident, and validation accuracy fluctuates significantly	Slower convergence and lower final training accuracy.

The best for our data: Xception
And why: Xception consistently improves over the epochs, achieving a final validation accuracy of 83.5% with a validation loss of 0.5229, which is the best balance between accuracy and loss among the models. It demonstrates steady learning without overfitting, as seen from the alignment of training and validation accuracy trends. This indicates that Xception generalizes well to our data.

The worst for our data : Resnet
And why: ResNet performs poorly, with a validation accuracy that stagnates around 55-60% and high validation loss throughout training. It struggles to capture the patterns in the data, failing to generalize or improve significantly.







References

- Deep Residual Learning for Image Recognition
- Densely Connected Convolutional Networks
- Xception: Deep Learning with Depthwise Separable Convolutions