**Multi Class Fish Image Classification**

**Problem Statement:**

This project focuses on classifying fish images into multiple categories using deep learning models. It involves training a Convolutional Neural Network (CNN) from scratch and applying transfer learning techniques by leveraging pre-trained models to improve performance. Additionally, the project includes saving the best-performing models for future use and deploying a Streamlit application to enable users to upload fish images and receive real-time predictions of fish categories. The goal is to achieve high classification accuracy and provide an easy-to-use web-based tool for fish image classification.

## **Objectives**

* Develop a CNN-based model for fish image classification.
* Experiment with transfer learning using pre-trained models (e.g., MobileNet, VGG16, ResNet50) to enhance performance.
* Select and save the best-performing model for future predictions.
* Compare model performance based on accuracy, precision, recall, and F1-score.
* Deploy the model using a Streamlit application to allow real-time fish image classification.
* Ensure a user-friendly web interface for easy interaction.

## **Dataset**

The dataset consists of fish images, categorized into multiple species, sourced from publicly available datasets or custom datasets. The images are pre-processed and augmented to improve the model's generalization capabilities. The dataset is split into:

* **Training Set**: Contains a majority of the images used to train the models.
* **Validation Set**: Used to monitor performance during training and prevent overfitting.
* **Test Set**: A separate dataset to evaluate the final model's performance.

**Approach:**

### **A.) Data Preprocessing and Augmentation**

In this project, the dataset was prepared using various preprocessing and augmentation techniques to ensure that the model generalizes well and can handle unseen data effectively.

#### **1. Rescaling:**

All images were rescaled to the [0, 1] range by dividing the pixel values by 255. This normalization step ensures that the input images are on a common scale, making it easier for the model to process the data. The rescaling was applied uniformly across the training, validation, and test sets.

#### **2. Data Augmentation (for training data):**

To enhance the robustness of the model and prevent overfitting, various augmentation techniques were applied to the training data. These transformations introduce variability into the dataset, allowing the model to learn from diverse image conditions and improve its generalization capabilities. The following augmentations were used:

* **Rotation:** Randomly rotated images by up to 40 degrees to make the model more invariant to image orientation.
* **Width Shift:** Randomly shifted images horizontally by 20% of the image width to simulate varying positions.
* **Height Shift:** Randomly shifted images vertically by 20% of the image height to account for varying vertical placements.
* **Shear:** Applied shear transformations to distort images and make the model more robust to geometric transformations.
* **Zoom:** Randomly zoomed into the images by up to 20% to simulate different zoom conditions.
* **Horizontal Flip:** Randomly flipped images horizontally to simulate various image orientations.
* **Fill Mode:** Pixels introduced as a result of these transformations were filled using the 'nearest' strategy, ensuring smooth transitions in the augmented images.

#### **3. Validation and Test Data:**

For the validation and test sets, no augmentation was applied to maintain the integrity of the data for evaluation purposes. Only rescaling was performed, where the pixel values were scaled to the [0, 1] range for consistency with the training data.

#### **4. Data Loading:**

Images were loaded from the specified directories (train, validation, and test) and resized to a uniform dimension of 224x224 pixels. A batch size of 32 was used to efficiently process the data in batches during both training and evaluation phases. The images were categorized into distinct classes using one-hot encoding, allowing the model to output predictions across multiple fish categories. The data was processed using the ImageDataGenerator, which loads and augments the images on the fly during training.

#### **5. Class Imbalance Handling:**

To handle the imbalance in the dataset (if some classes have significantly more images than others), class weights were calculated. These class weights ensure that the model does not become biased towards the overrepresented classes during training. The class labels were extracted from the training data, and the class weights were computed using compute\_class\_weight from the sklearn.utils.class\_weight module. This technique assigns a higher weight to underrepresented classes and a lower weight to overrepresented classes, effectively balancing the learning process.

The computed class weights were used during model training to penalize misclassifications of the underrepresented classes more heavily, helping the model learn to classify all categories with improved precision and recall.

**B.) Building a CNN Model from Scratch and Fine-tuning Pre-trained Models:**

In this step, we employed two approaches for fish image classification:

1. **Building a CNN Model from Scratch**: We developed and trained a custom Convolutional Neural Network (CNN) tailored to this specific task.
2. **Fine-tuning Pre-trained Models**: We fine-tuned five pre-trained models: VGG16, ResNet50, MobileNet, InceptionV3, and EfficientNetB0. These models, trained on ImageNet, were adapted to the fish dataset.

**1. Building a CNN Model from Scratch**

The custom CNN model was designed to extract meaningful features from the input fish images and classify them into different categories.

**Steps for CNN Model:**

**1. Input Layer:** The input images were resized to (224, 224, 3) dimensions to match the models input size.

**2. Convolutional Layers:**

* **Conv2D Layer 1:** A convolutional layer with 32 filters, a kernel size of (3, 3), and ReLU activation function was added. This layer detects low-level features such as edges.
* **Conv2D Layer 2:** A convolutional layer with 64 filters, a kernel size of (3, 3), and ReLU activation function was added. This layer detects more complex patterns like textures.
* **Conv2D Layer 3:** A convolutional layer with 128 filters, a kernel size of (3, 3), and ReLU activation function was added to capture higher-level patterns.

**3. Pooling Layers:** After each convolutional layer, a max-pooling layer with a pool size of (2, 2) was added to reduce the spatial dimensions and focus on key features.

**4. Global Average Pooling**: Instead of flattening, we used global average pooling to reduce the model size and overfitting.

**5. Dropout Layers:** Dropout layers were added after each pooling layer to reduce overfitting. A dropout rate of 0.5 was used.

**6. Fully Connected Layers:**

* **Dense Layer 1:** A fully connected layer with 512 units and ReLU activation function was added to learn complex representations.
* **Dropout Layer:** A dropout layer with a rate of 0.5 was added to prevent overfitting.
* **Dense Layer 2:** A final dense layer with the number of units equal to the number of classes (i.e., num\_classes) was added, with a softmax activation function to output class probabilities.

**7. Model Compilation:** The model was compiled using the Adam optimizer, categorical cross-entropy loss, and accuracy as the evaluation metric.

**8. Training and Evaluation**:

* The model was trained for **20 epochs** with a batch size of 32.
* Early stopping was applied, monitoring validation accuracy, to stop training when there was no improvement for five consecutive epochs.

**Results of CNN Model from Scratch**:

* Training Accuracy: 81.9%
* Validation Accuracy: 83.7%
* Test Accuracy: 85.0%
* Training Loss: 0.4798
* Validation Loss: 0.4523
* Test Loss: 0.4335

#### **Observations:**

* The model demonstrated a steady improvement in accuracy and reduction in loss over epochs.
* Validation and test accuracies are slightly higher than training accuracy, indicating **no overfitting** and **good generalization**.
* Class weights helped balance performance across classes, improving model robustness on imbalanced data.

#### **Conclusion:**

The CNN model achieved high accuracy on unseen data (85%) and performs reliably on both validation and test sets. This indicates that the model is **generalizing well** and is **suitable for deployment or further fine-tuning**.

**2. Fine-tuning Pre-trained Models**

After building a CNN from scratch, we fine-tuned five pre-trained models: VGG16, ResNet50, MobileNet, InceptionV3, and EfficientNetB0. These models are pre-trained on the ImageNet dataset, and we utilized transfer learning to adapt them for the fish classification task.

**Steps for Fine-tuning Pre-trained Models**:

1. **Freezing Base Layers:**The base layers of each pre-trained model (VGG16, ResNet50, MobileNet, InceptionV3, and EfficientNetB0) were frozen to retain the learned weights from the ImageNet dataset and avoid overfitting on our smaller fish dataset.
2. **Adding Custom Classification Layers:**

* A global average pooling layer.
* A fully connected layer with 512 units and ReLU activation.
* A dropout layer with a dropout rate of 0.5 to reduce overfitting.
* A final dense layer with the number of classes (num\_classes) and softmax activation to predict probabilities for each class.

1. **Model Compilation:**The models were compiled using the Adam optimizer with a learning rate of 0.0001, categorical cross-entropy loss, and accuracy as the evaluation metric.
2. **Training with Early Stopping:**Each model was trained for 20 epochs with early stopping to avoid overfitting. The best model, in terms of validation accuracy, was saved for further evaluation.

**Training and Evaluation of Pre-trained Models**:

1. **VGG16**:

* Validation Accuracy: **84.62%** (Epoch 20)
* Loss decreased but accuracy was modest, reaching 84.62% by Epoch 20.

1. **ResNet50**:

* Validation Accuracy: **20.24%** (Epoch 7)
* Consistently poor performance, with no significant improvement in accuracy over epochs.

1. **MobileNet**:

* Best Validation Accuracy: **99.45%** (Epoch 12)
* Achieved almost perfect accuracy and very low validation loss.

1. **InceptionV3**:

* Validation Accuracy: **98.08%** (Epoch 20)
* Strong performance with high accuracy (98.08%) and low validation loss.

1. **EfficientNetB0:**

* Validation Accuracy: **12.82%** (Epoch 7)
* Low Accuracy with Low Validation Loss

**Conclusion**: MobileNet performed the best with the highest validation accuracy and fastest training time.

The custom CNN model and the fine-tuned pre-trained models, particularly MobileNet, demonstrated excellent performance on the fish classification task. The **MobileNet** model was the most effective, achieving **99.45% validation accuracy** and **99.62% test accuracy** with a low loss, making it the optimal model for deployment. The model was saved as best\_model\_MobileNet.h5 for future use.

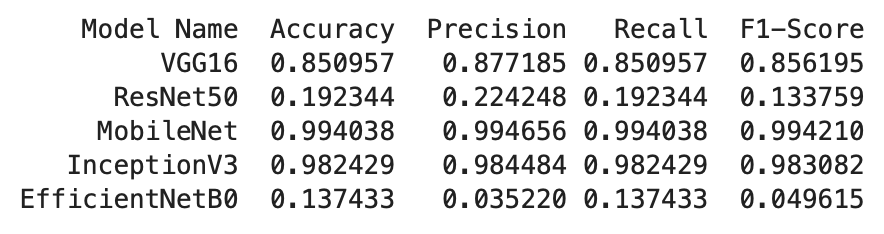
**C.)** **Model Evaluation**

To assess the performance of the trained models, several key metrics were used, including accuracy, precision, recall, F1-score, and the confusion matrix. These metrics provided insight into how well the models classified the fish images across multiple categories.

1. **Metrics Used:**

* **Accuracy:** The percentage of correctly predicted fish images.
* **Precision:** The ability of the model to not label a negative instance as positive.
* **Recall:** The ability of the model to find all the relevant cases (true positives).
* **F1-score:** The harmonic mean of precision and recall, balancing both metrics.
* **Confusion Matrix:** A visualization of true positives, true negatives, false positives, and false negatives for each class.

1. **Model Performance Summary:** Each of the pre-trained models (VGG16, ResNet50, MobileNet, InceptionV3, and EfficientNetB0) was evaluated on the test dataset. Below is a comparison of the key performance metrics for each model:



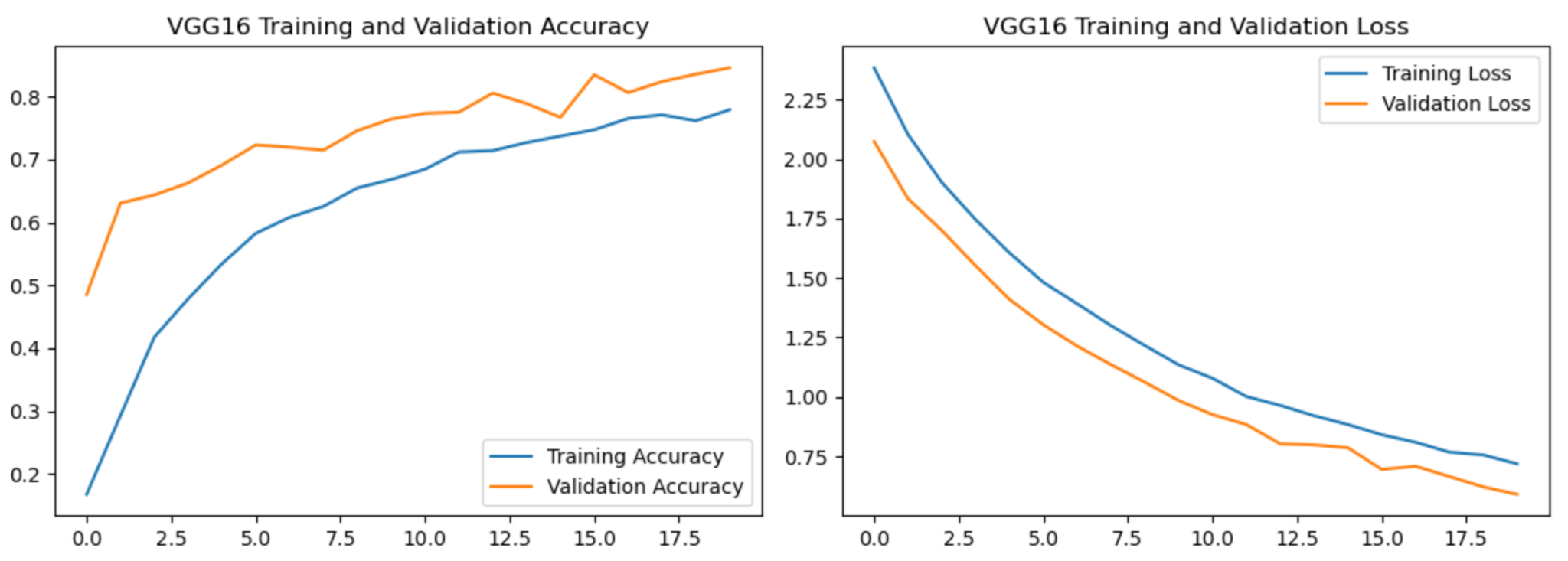
**MobileNet** achieved the best and most balanced performance across all evaluation metrics, making it the most suitable candidate for deployment. It recorded a remarkably high **accuracy of 99.40%**, with similarly high **precision (99.47%)**, **recall (99.40%)**, and **F1-score (99.42%)**, indicating consistent and reliable classification performance.

**InceptionV3** also performed strongly, with **accuracy of 98.24%**, **precision of 98.45%**, **recall of 98.24%**, and **F1-score of 98.31%**, making it a solid alternative, although slightly behind MobileNet in overall effectiveness.

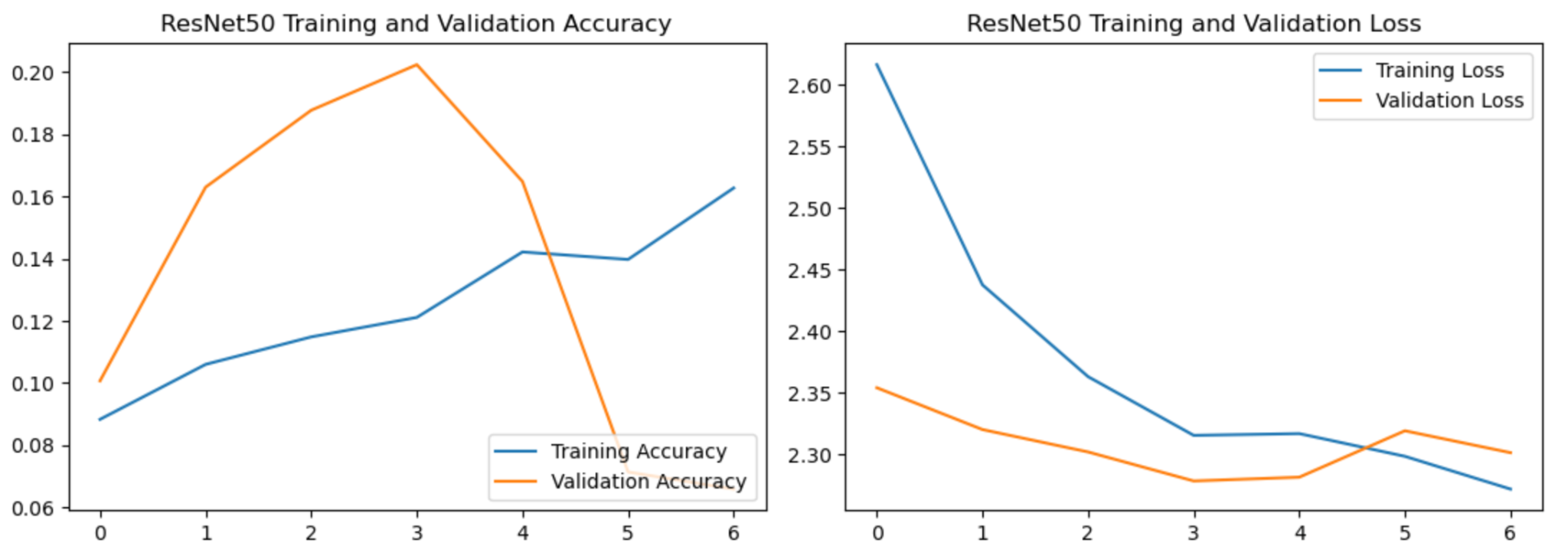
**VGG16** showed moderate performance, achieving **85.10% accuracy**, and relatively balanced precision and recall values, but it was still outperformed by both MobileNet and InceptionV3.

**ResNet50** and **EfficientNetB0**, however, performed poorly. ResNet50 had a low **accuracy of 19.23%** and a particularly low **F1-score of 13.38%**, suggesting that its predictions were often incorrect or imprecise. **EfficientNetB0** performed the worst, with an **accuracy of just 13.74%**, **precision of 3.52%**, and **F1-score of only 4.96%**, indicating severe misclassification issues.

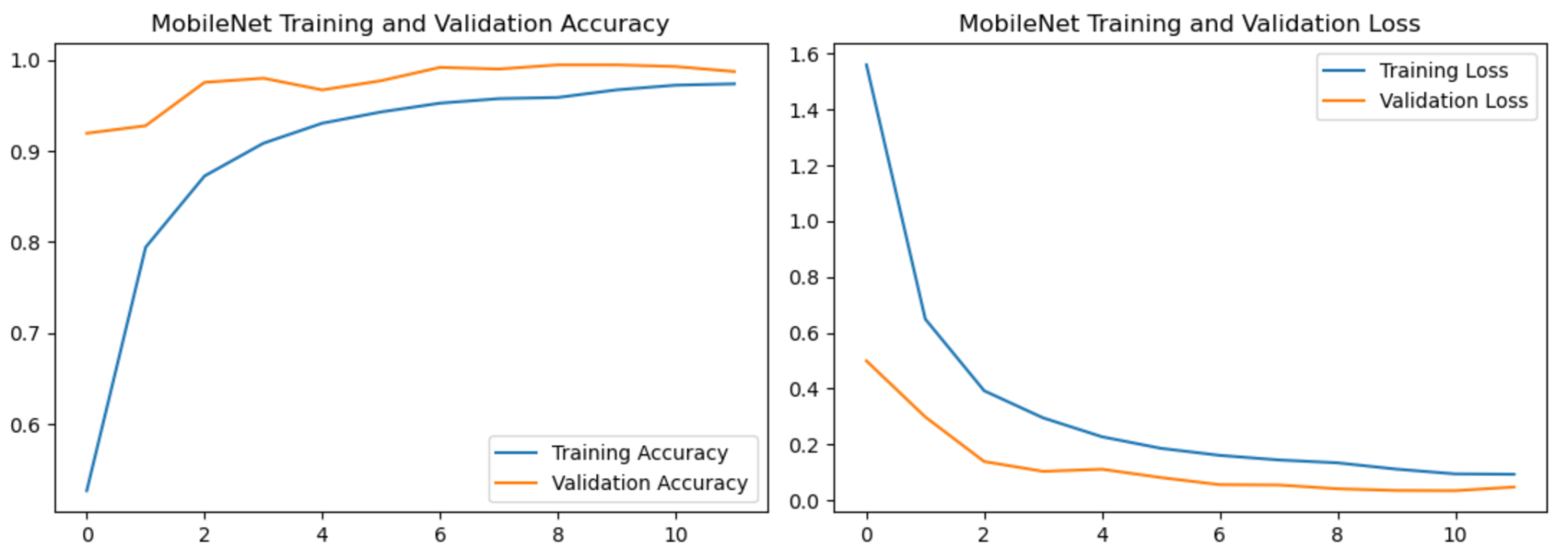
1. **Visualization of Training History:**The training history for each model, including accuracy and loss over epochs, was visualized to track model improvement during training. This helped in identifying any overfitting issues and guided the use of early stopping.



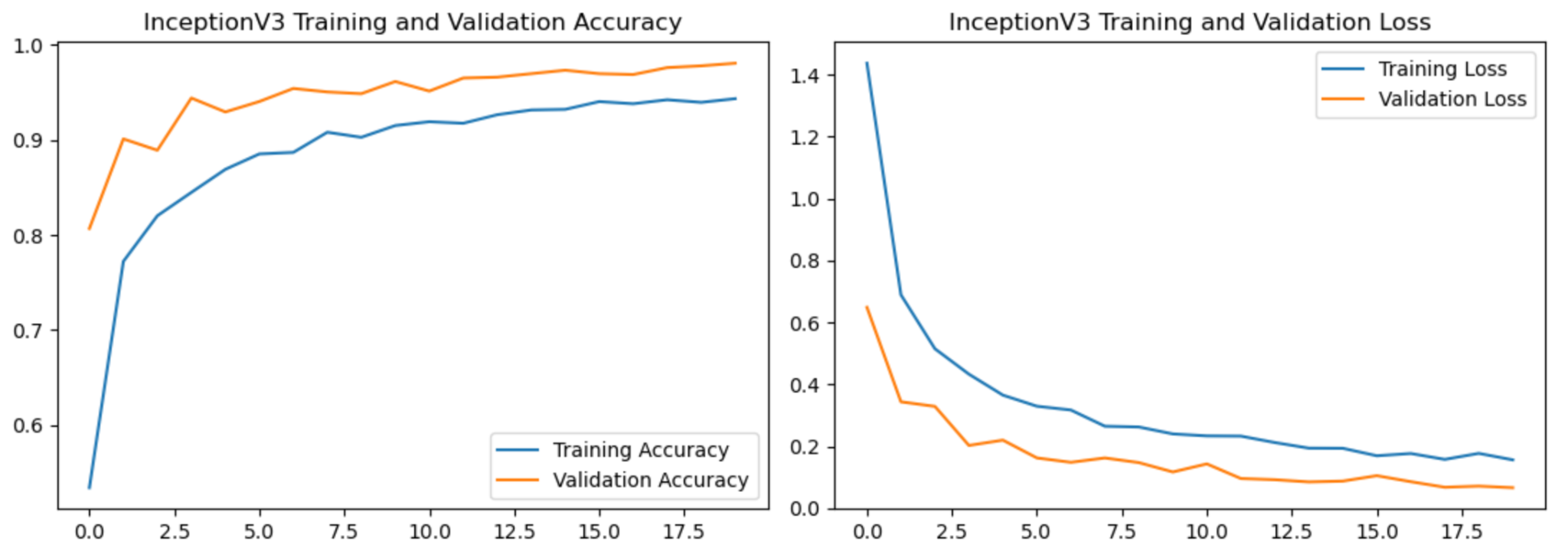
The graph shows that there is a potential overfitting issue, as indicated by the gap between training and validation metrics.



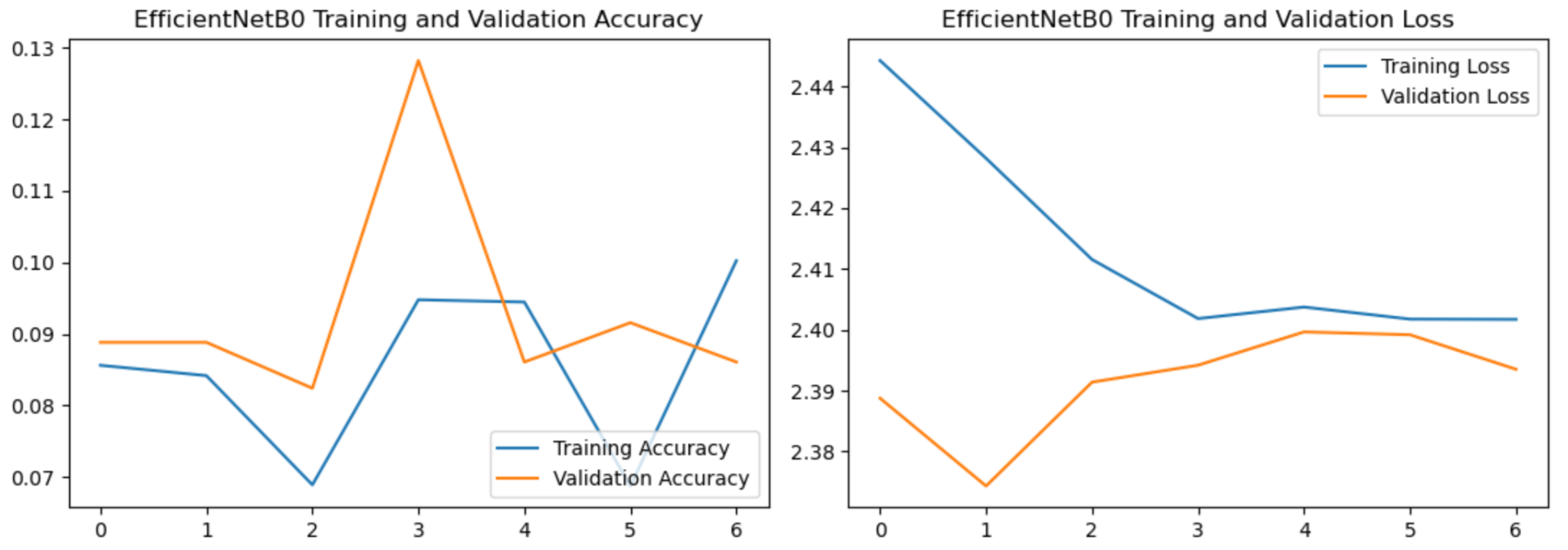
The graph indicates that the model shows signs of overfitting since the validation accuracy doesn't follow the improvement of the training accuracy.



The graph shows that the model is performing well due to very less gap between training and validation metrics.

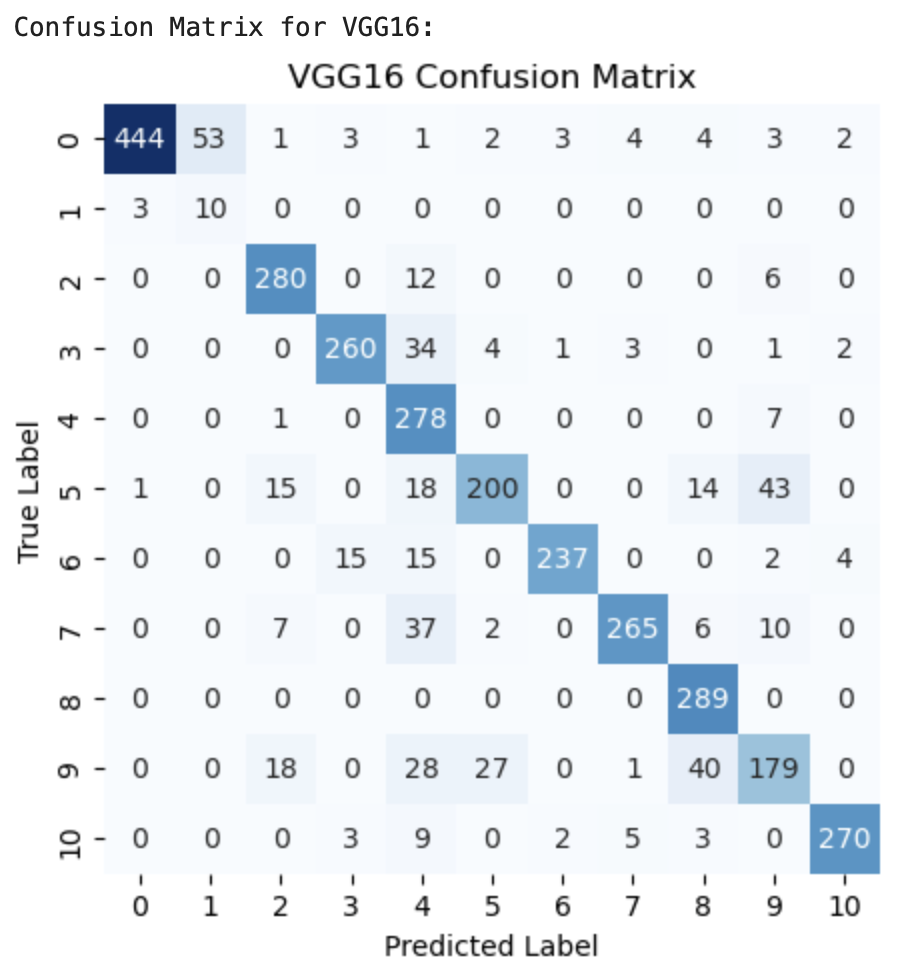


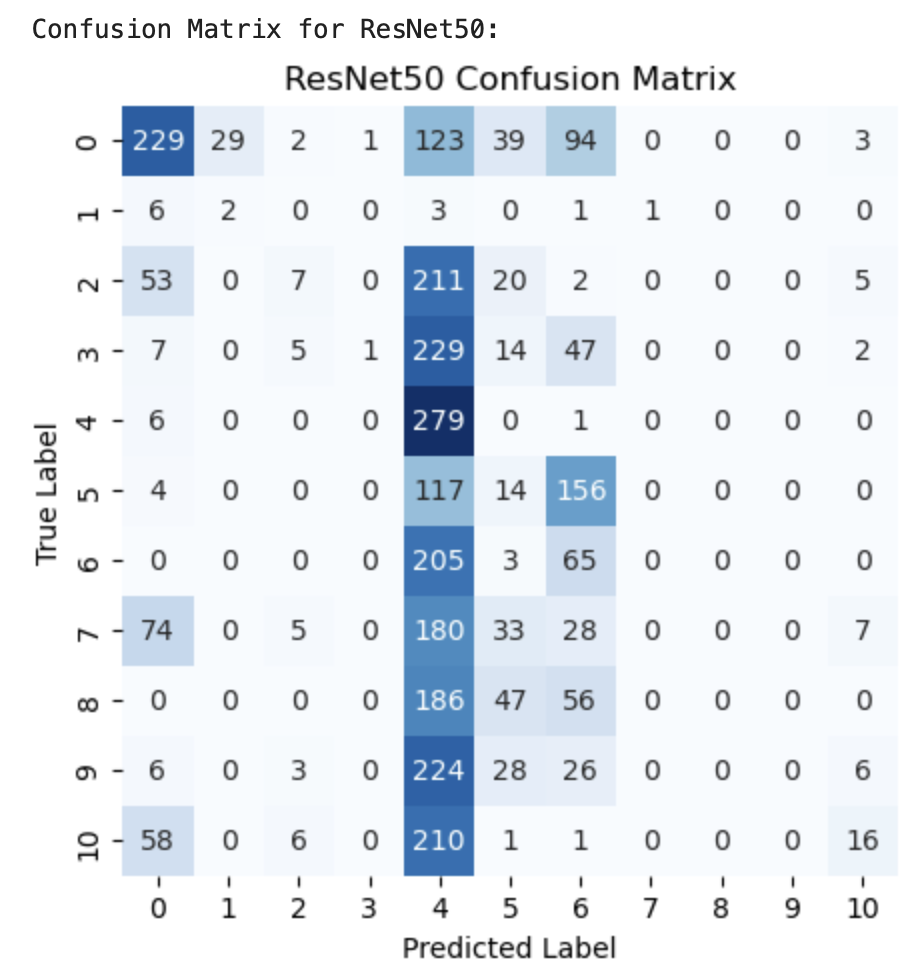
The graph indicates that the model demonstrates good training performance and reasonable validation capability, with a slight overfitting tendency.

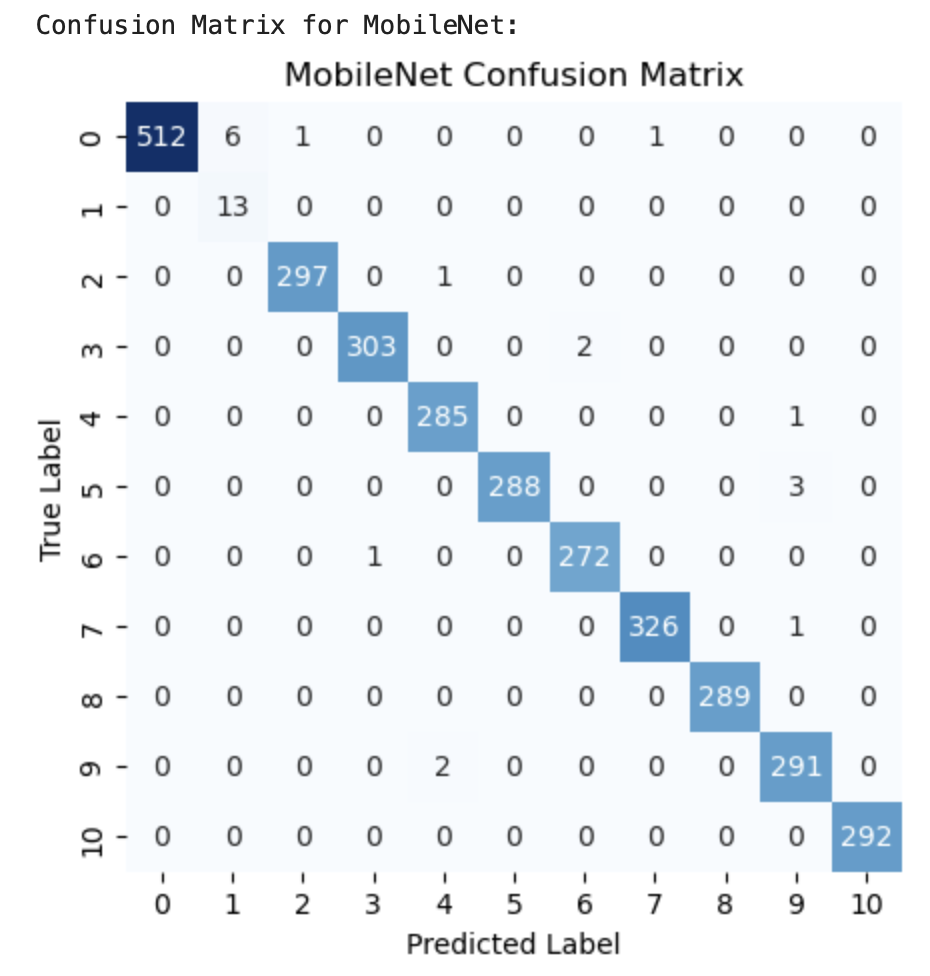


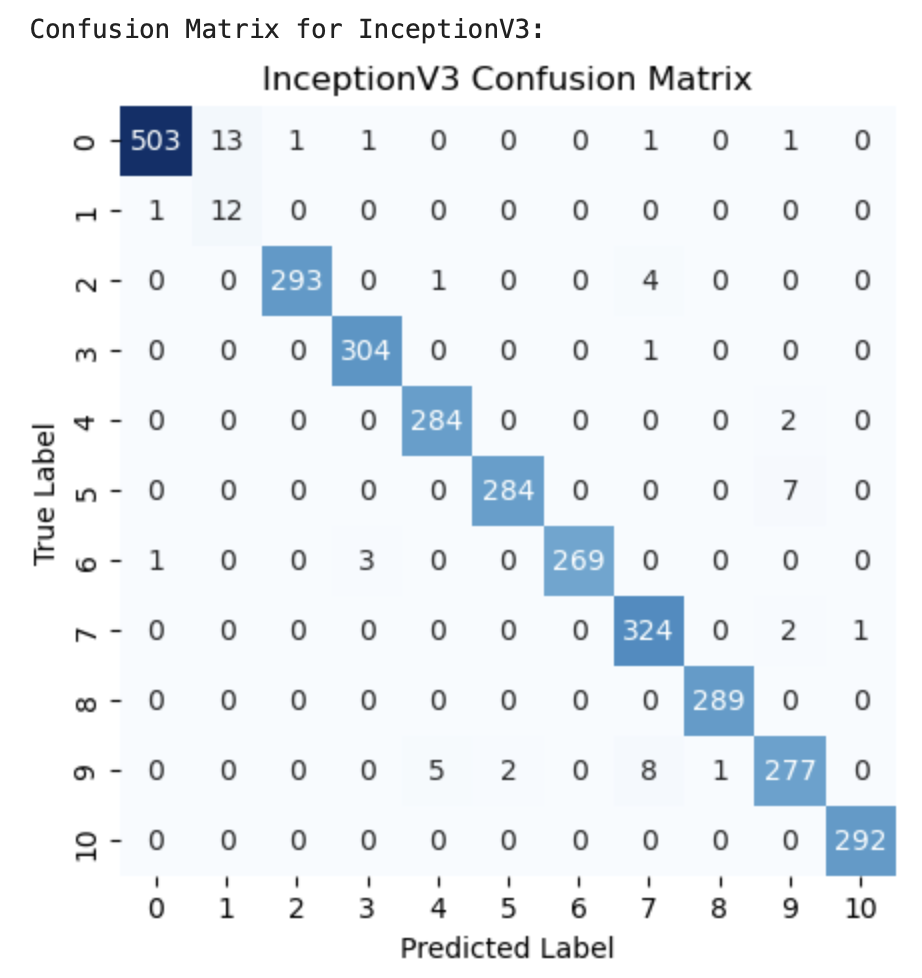
The graph indicates that the model demonstrates effective learning overall but may be susceptible to overfitting, as indicated by the discrepancy between training and validation performance.

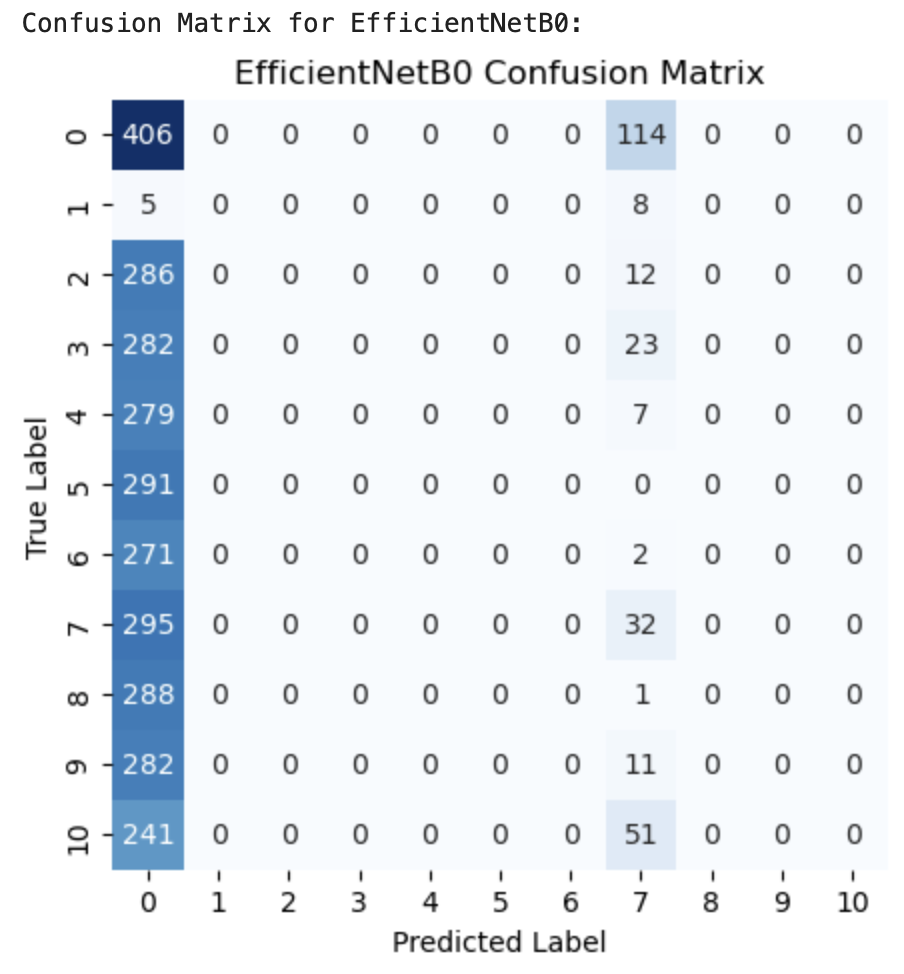
1. **Confusion Matrices:**Confusion matrices were plotted for each model to provide a detailed view of the misclassifications and help identify which fish species were confused with others. This visualization highlighted areas where model performance could be improved.











**D.) Deployment: Streamlit Application**

The fish image classifier model is deployed using **Streamlit**, which provides a user-friendly interface for interacting with the model. Below is an overview of the key components of the deployment:

#### **1. Model Loading**

* The pre-trained model, saved as best\_model\_MobileNet.h5, is loaded into the app using tensorflow.keras.models.load\_model.
* The model was previously trained to classify fish images into multiple categories.

#### **2. Fish Categories**

* The app defines the possible fish categories the model can predict, including types like 'animal fish', 'bass', 'black sea sprat', 'shrimp', etc.

#### **3. Image Upload**

* Users can upload an image of a fish through the file\_uploader component in Streamlit, which supports image formats like .jpg and .png.

#### **4. Image Preprocessing**

* The uploaded image is resized to **150x150 pixels**, normalized by dividing pixel values by 255, and expanded to add a batch dimension. This step ensures that the image is compatible with the model's input expectations.

#### **5. Model Prediction**

* After preprocessing, the image is passed through the model to predict the fish category.
* The predicted class is extracted using np.argmax() to find the class with the highest probability.
* The confidence score (i.e., the probability of the predicted class) is also displayed using np.max().

#### **6. Application Interface**

* The interface is divided into two columns:  
  + **Left Column**: Displays the uploaded fish image.
  + **Right Column**: Shows the predicted fish category and the confidence score in an easy-to-read format with color-coded text.

#### **7. User Interaction**

* The app is designed for users to easily upload fish images, receive real-time predictions, and gain insight into how confident the model is about its classification.

### **Conclusion**

In this project, we developed a robust deep learning model capable of classifying various fish species from images using Convolutional Neural Networks (CNNs) and transfer learning techniques. The process involved training multiple pre-trained models (MobileNet, VGG16, ResNet50, etc.), fine-tuning them for our dataset, and evaluating their performance through metrics like accuracy, precision, recall, F1-score, and confusion matrices. We identified the best-performing model (MobileNet) and deployed it using a user-friendly Streamlit application that allows users to upload images and receive real-time predictions with confidence scores.

The deployment of the model in a Streamlit app makes the fish classification system accessible and practical for a variety of use cases, such as fish species identification in research, aquaculture management, or educational tools for marine biology. The model's accuracy, combined with the intuitive interface, demonstrates the power of deep learning in image classification tasks and provides a real-world solution for identifying fish species from images.

This project showcases the full workflow of building, evaluating, and deploying a deep learning model, highlighting the practical applications of modern AI techniques in image classification. Through continual refinement and real-time feedback, the system can evolve further to support a broader range of categories and offer improved accuracy over time.