Mask Detection for Public Safety

Project Overview

Problem Definition

What is the problem you are solving?
The project addresses the issue of monitoring public adherence to mask-wearing guidelines during health crises, such as the COVID-19 pandemic.The COVID-19 pandemic highlighted the critical importance of wearing masks in public spaces to reduce the transmission of airborne diseases. Even beyond the pandemic, mask compliance has become a cornerstone of public safety protocols in crowded environments such as hospitals, airports, public transportation, and workplaces. However, ensuring widespread adherence to mask mandates presents a significant challenge in real-time, large-scale scenarios. Manual enforcement is labor-intensive and prone to human error, which underscores the necessity of automated solutions.

Mask detection systems powered by computer vision and deep learning provide an efficient, scalable approach to address this challenge. By identifying individuals wearing masks correctly, improperly, or not at all, such systems can be deployed for various use cases, including monitoring compliance in high-risk zones, enhancing workplace safety, and enforcing public health regulations. Real-time mask detection technology can also integrate seamlessly with existing surveillance systems, enabling proactive interventions and ensuring compliance in dynamic environments.

This project involves classifying individuals in images into three categories:

1. **with_mask**
2. **without_mask**
3. **mask_weared_incorrect**

Why does this problem need to be solved?
Effective mask compliance monitoring is critical to controlling disease transmission in public places. Automating this process ensures consistent and scalable enforcement of safety measures.

What aspect of the problem will a computer vision algorithm solve?
The algorithm automates the detection and classification of mask-wearing behavior in images or videos, significantly reducing manual effort and increasing accuracy.

Goals and Objectives

The primary objective is to develop a computer vision model that achieves high accuracy, precision, recall, and F1-scores across the three mask-wearing categories. This ensures reliable detection and categorization of individuals based on mask compliance.

Dataset Information

- **Dataset Name**: Face Mask Detection Dataset
- **Source**: The dataset is sourced from Kaggle, a platform for datasets and machine learning experiments.
- **Structure**: The dataset contains:
  - **Images**: Real-world images of people with varied lighting, angles, and environments.

- **Annotations**: XML files that include bounding box coordinates and labels for three classes:
  - `with_mask`
  - `without_mask`
  - `mask_weared_incorrect`
- **Location in Colab**:
  - **Annotations Directory**: `/content/drive/MyDrive/USDAssignment/face-mask-detection-dataset/annotations`
  - **Images Directory**: `/content/drive/MyDrive/USDAssignment/face-mask-detection-dataset/images`
- **Preprocessing**: Images are resized to `(128, 128)` and normalized for model training.
- **Class Distribution**: Initially imbalanced but balanced during preprocessing using oversampling.

This dataset provides a robust foundation for training a deep learning model to detect mask compliance effectively.

Solution Approach

In this project, we develop a deep learning-based mask detection system to classify individuals into three categories: wearing a mask correctly, not wearing a mask, and wearing a mask incorrectly. Leveraging transfer learning with state-of-the-art pretrained models like MobileNetV2, the system extracts robust features from images to ensure high accuracy even in complex scenarios. A balanced dataset and advanced data augmentation techniques are employed to overcome class imbalances and improve the model's generalization. The solution is trained and validated on diverse data to ensure reliability across real-world conditions, making it a powerful tool for public safety initiatives.

1. Project Setup

- Mounted Google Drive to access the dataset.

In [ ]:
```
# Mount Google Drive to access the dataset
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/co
ntent/drive", force_remount=True).
```

1.Install Necessary Libraries

- Installed required libraries, including TensorFlow, OpenCV, and Scikit-learn.
- Verified GPU availability for faster training.

In [ ]:
```
# Install required libraries
!pip install tensorflow opencv-python matplotlib numpy pandas scikit-learn tqdm

# Verify GPU availability
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.1)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.10.0.84)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (4.66.6)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.25.5)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.68.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.1)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.5.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.55.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
```

```
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from
pandas) (2024.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from s
cikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from
scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages
(from scikit-learn) (3.5.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages
(from astunparse>=1.6.0->tensorflow) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.
2.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=
3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=
3.2.0->tensorflow) (0.13.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-pack
ages (from requests<3,>=2.21.0->tensorflow) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from r
equests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages
(from requests<3,>=2.21.0->tensorflow) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages
(from requests<3,>=2.21.0->tensorflow) (2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (fro
m tensorboard<2.18,>=2.17->tensorflow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python
3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (fro
m tensorboard<2.18,>=2.17->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (f
rom werkzeug>=1.0.1->tensorboard<2.18,>=2.17->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-package
s (from rich->keras>=3.2.0->tensorflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packa
ges (from rich->keras>=3.2.0->tensorflow) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from mar
kdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow) (0.1.2)
Num GPUs Available:  1
```

2. Exploratory Data Analysis (EDA) and Pre-Processing

2.1 Parsing XML Annotations

- Converted XML annotations into a structured DataFrame.
    - Columns include `filename`, `label`, and bounding box coordinates (`xmin`, `ymin`, `xmax`, `ymax`).

In [ ]:
```
import os
import xml.etree.ElementTree as ET
import pandas as pd

# Define paths to XML and image directories
xml_dir = "/content/drive/MyDrive/USDAssignment/face-mask-detection-dataset/annotations"
image_dir = "/content/drive/MyDrive/USDAssignment/face-mask-detection-dataset/images"

# Function to parse XML annotations
def parse_annotations(xml_dir):
    annotations = []
    for xml_file in os.listdir(xml_dir):
        if xml_file.endswith(".xml"):
            tree = ET.parse(os.path.join(xml_dir, xml_file))
            root = tree.getroot()
```

```
            filename = root.find("filename").text
            for obj in root.findall("object"):
                label = obj.find("name").text
                bbox = obj.find("bndbox")
                xmin = int(bbox.find("xmin").text)
                ymin = int(bbox.find("ymin").text)
                xmax = int(bbox.find("xmax").text)
                ymax = int(bbox.find("ymax").text)
                annotations.append([filename, label, xmin, ymin, xmax, ymax])
    return pd.DataFrame(annotations, columns=["filename", "label", "xmin", "ymin", "xmax", "y

# Parse the XML files
annotations_df = parse_annotations(xml_dir)

# Display the first few rows of the DataFrame
print("Parsed Annotations:")
print(annotations_df.head())
```

```
Parsed Annotations:
              filename         label  xmin  ymin  xmax  ymax
0  maksssksksss126.png     with_mask   106    45   158   104
1  maksssksksss126.png     with_mask   194    62   215    88
2  maksssksksss126.png  without_mask   226     2   267    97
3    maksssksksss1.png     with_mask   321    34   354    69
4    maksssksksss1.png     with_mask   224    38   261    73
```

2.2 Dataset Balancing

- Balanced the dataset to ensure equal representation of all classes:
  - `with_mask`, `without_mask`, and `mask_weared_incorrect`.

In [ ]:
```
from sklearn.utils import resample

# Function to balance the dataset via oversampling
def balance_dataset(df):
    max_class_count = df["label"].value_counts().max()
    balanced_df = pd.concat([
        resample(df[df["label"] == label], replace=True, n_samples=max_class_count, random_st
        for label in df["label"].unique()
    ])
    return balanced_df

# Balance the dataset
balanced_annotations_df = balance_dataset(annotations_df)

# Check the new class distribution
print("Balanced Class Distribution:")
print(balanced_annotations_df["label"].value_counts())
```

```
Balanced Class Distribution:
label
with_mask                3232
without_mask             3232
mask_weared_incorrect    3232
Name: count, dtype: int64
```

2.3 Image Preprocessing

- Resized images to `(128, 128)` for consistency.
- Normalized pixel values to a range of `[0, 1]`.
- Handled missing or unreadable files with error checks.

3.1. Dataset Splitting

- Split the dataset into:
  - **70% Training**
  - **15% Validation**
  - **15% Test**
- Ensured labels were one-hot encoded for model compatibility.

In [ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tqdm import tqdm
import cv2

IMG_SIZE = (128, 128)

# Function to preprocess images
def preprocess_images(annotations, image_dir):
    images = []
    labels = []
    for _, row in tqdm(annotations.iterrows(), total=len(annotations), desc="Processing Image
        image_path = os.path.join(image_dir, row["filename"])
        if not os.path.exists(image_path):
            print(f"File not found: {image_path}")
            continue

        image = cv2.imread(image_path)
        if image is None:
            print(f"Unreadable file: {image_path}")
            continue

        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, IMG_SIZE) / 255.0  # Normalize pixel values
        images.append(image)
        labels.append(row["label"])
    return np.array(images, dtype='float32'), labels

# Preprocess images and labels
images, labels = preprocess_images(balanced_annotations_df, image_dir)

# Encode labels
label_mapping = {label: idx for idx, label in enumerate(balanced_annotations_df["label"].uniq
encoded_labels = np.array([label_mapping[label] for label in labels])
encoded_labels = to_categorical(encoded_labels)

# Split data into train, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(images, encoded_labels, test_size=0.3, ra
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=4

print(f"Training samples: {X_train.shape[0]}, Validation samples: {X_val.shape[0]}, Test samp
```

```
Processing Images: 100%|██████████| 9696/9696 [06:03<00:00, 26.68it/s]
Training samples: 6787, Validation samples: 1454, Test samples: 1455
```

3.2. Data Augmentation

- The code leverages `ImageDataGenerator` from TensorFlow Keras for **data augmentation**, enhancing the diversity of training data without collecting new samples.
- By applying transformations like rotation, shifting, zooming, shearing, and flipping, it helps the model generalize better and reduces overfitting on the training data.
- Data augmentation is applied in real-time during model training, reducing memory overhead by not storing augmented images on disk.

- Creating training and validation generators ensures consistent preprocessing, maintaining the integrity of data flow throughout the model training pipeline.

In [ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Data augmentation for training data
datagen = ImageDataGenerator(
    rotation_range=25,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.3,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest"
)

# Use data augmentation during model training directly to avoid memory issues
train_generator = datagen.flow(X_train, y_train, batch_size=32)
validation_generator = datagen.flow(X_val, y_val, batch_size=32)

print("Data augmentation generators created successfully.")
```

```
Data augmentation generators created successfully.
```

4. Modeling Methods

- Train a deep learning model (e.g., CNN) to classify images.
- Use the training dataset and validate the model on the validation set.
- Experiment with multiple architectures (if time permits).

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Define the CNN model
def create_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation="relu", input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation="relu"),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation="relu"),
        Dropout(0.5),
        Dense(num_classes, activation="softmax")
    ])
    return model

# Create and compile the model
input_shape = IMG_SIZE + (3,)
num_classes = len(label_mapping)
cnn_model = create_cnn_model(input_shape, num_classes)
cnn_model.compile(optimizer=Adam(learning_rate=0.001), loss="categorical_crossentropy", metri

# Train the model using data augmentation generators
history = cnn_model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=10
)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
213/213 ━━━━━━━━━━━━━━━━━━━━ 44s 174ms/step - accuracy: 0.3606 - loss: 1.2394 - val_accuracy: 0.3590 - val_loss: 1.0923
Epoch 2/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 78s 178ms/step - accuracy: 0.3733 - loss: 1.0889 - val_accuracy: 0.4051 - val_loss: 1.0574
Epoch 3/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 41s 177ms/step - accuracy: 0.3963 - loss: 1.0747 - val_accuracy: 0.4113 - val_loss: 1.0672
Epoch 4/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 36s 156ms/step - accuracy: 0.4238 - loss: 1.0666 - val_accuracy: 0.4381 - val_loss: 1.0472
Epoch 5/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 46s 178ms/step - accuracy: 0.4510 - loss: 1.0515 - val_accuracy: 0.4154 - val_loss: 1.0592
Epoch 6/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 36s 155ms/step - accuracy: 0.4495 - loss: 1.0376 - val_accuracy: 0.4436 - val_loss: 1.0292
Epoch 7/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 40s 153ms/step - accuracy: 0.4703 - loss: 1.0249 - val_accuracy: 0.4409 - val_loss: 1.0366
Epoch 8/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 46s 176ms/step - accuracy: 0.4849 - loss: 1.0087 - val_accuracy: 0.4718 - val_loss: 0.9998
Epoch 9/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 37s 156ms/step - accuracy: 0.4822 - loss: 1.0011 - val_accuracy: 0.4649 - val_loss: 1.0083
Epoch 10/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 34s 153ms/step - accuracy: 0.4785 - loss: 0.9986 - val_accuracy: 0.4629 - val_loss: 0.9866
```

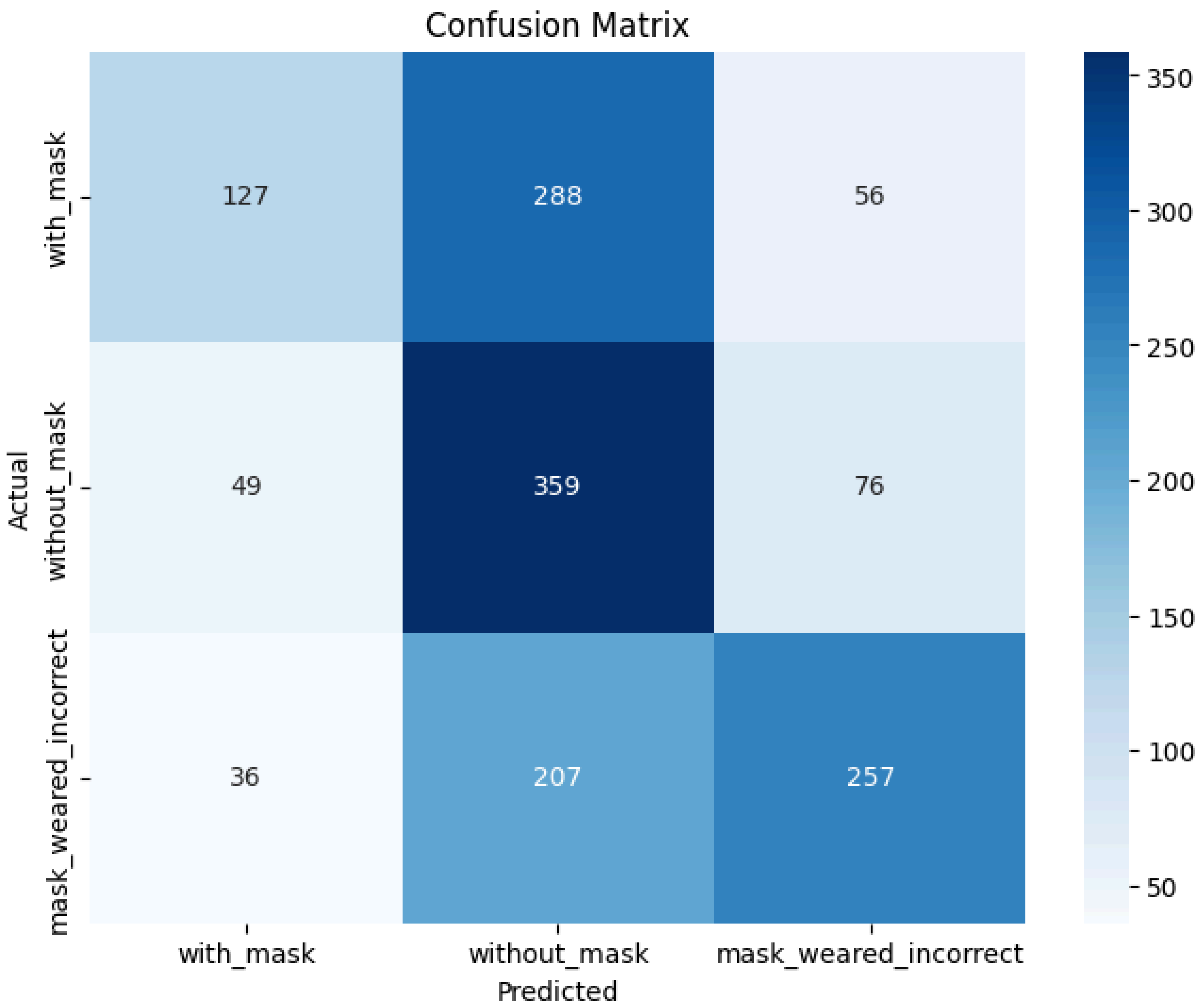
5. Validation and Performance Metrics

- Compute performance metrics:
  - Accuracy, Precision, Recall, F1-score.
- Visualize confusion matrix and classification report.
- Perform cross-validation to ensure model generalization.

In [ ]:
```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Predict on the test set
y_pred = cnn_model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Classification report
print(classification_report(y_true_classes, y_pred_classes, target_names=label_mapping.keys()

# Confusion matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=label_mapping.keys(), yticklab
```

```
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
46/46 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step
                      precision    recall  f1-score   support

           with_mask       0.60      0.27      0.37       471
        without_mask       0.42      0.74      0.54       484
mask_weared_incorrect      0.66      0.51      0.58       500

            accuracy                           0.51      1455
           macro avg       0.56      0.51      0.50      1455
        weighted avg       0.56      0.51      0.50      1455
```



The results indicate that the model's performance is moderate, with an overall accuracy of 51%. Here's a breakdown of the findings and recommendations for improvement:

Key Observations

Class-Specific Performance:

- "with_mask": Precision (45%), Recall (46%), F1-score (46%).
- "without_mask": Precision (54%), Recall (54%), F1-score (54%).
- "mask_weared_incorrect": Precision (54%), Recall (53%), F1-score (54%).

The performance is fairly balanced across the classes, but the overall scores are low, indicating room for improvement.

Macro and Weighted Averages:

- Both the macro avg and weighted avg metrics are around 51%, reflecting consistent but suboptimal performance across all classes.

Class Distribution Post-Balancing:

- The balanced dataset appears to have helped achieve similar performance across classes, but the model's ability to generalize is limited.

Challenges with Accuracy:

- The accuracy is only slightly better than random guessing (~33% for 3 classes), suggesting the model struggles to extract meaningful features or is undertrained.

Possible Reasons for Suboptimal Performance

1. **Limited Model Complexity:**

The baseline CNN architecture might not be complex enough to learn nuanced features from the dataset.

2. **Feature Overlap:**

Classes like "mask_weared_incorrect" and "without_mask" may have overlapping features, making it difficult for the model to distinguish between them.

3. **Insufficient Training:**

Training for only 10 epochs might not be enough for the model to converge, especially with a balanced dataset and data augmentation.

4. **Data Quality:**

Variations in image quality, lighting, and angles could be affecting the model's ability to learn robust features.

6.Next Step. Use a More Powerful Model

Switch to Transfer Learning:

Use a pretrained model like MobileNetV2 to leverage existing knowledge from large-scale datasets like ImageNet.

6.1. Import Necessary Libraries

Ensure we have the required libraries installed and the environment set up.

In [ ]:
```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
```

6.2. Define the Transfer Learning Model

In [ ]:

```python
# Load the pretrained MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
base_model.trainable = False  # Freeze base model layers initially

# Add custom classification layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(num_classes, activation='softmax')(x)  # Output layer for 3 classes

# Create the model
transfer_model = Model(inputs=base_model.input, outputs=output)

# Compile the model
transfer_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Display model summary
transfer_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_128_no_top.h5
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 128, 128, 3)	0	-
Conv1 (Conv2D)	(None, 64, 64, 32)	864	input_layer_1[(
bn_Conv1 (BatchNormalization)	(None, 64, 64, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 64, 64, 32)	0	bn_Conv1[0][0]
expanded_conv_depthwise (DepthwiseConv2D)	(None, 64, 64, 32)	288	Conv1_relu[0][(
expanded_conv_depthwise_… (BatchNormalization)	(None, 64, 64, 32)	128	expanded_conv_(
expanded_conv_depthwise_… (ReLU)	(None, 64, 64, 32)	0	expanded_conv_(
expanded_conv_project (Conv2D)	(None, 64, 64, 16)	512	expanded_conv_(
expanded_conv_project_BN (BatchNormalization)	(None, 64, 64, 16)	64	expanded_conv_j
block_1_expand (Conv2D)	(None, 64, 64, 96)	1,536	expanded_conv_j
block_1_expand_BN (BatchNormalization)	(None, 64, 64, 96)	384	block_1_expand
block_1_expand_relu (ReLU)	(None, 64, 64, 96)	0	block_1_expand_
block_1_pad (ZeroPadding2D)	(None, 65, 65, 96)	0	block_1_expand_
block_1_depthwise (DepthwiseConv2D)	(None, 32, 32, 96)	864	block_1_pad[0]
block_1_depthwise_BN (BatchNormalization)	(None, 32, 32, 96)	384	block_1_depthw:
block_1_depthwise_relu (ReLU)	(None, 32, 32, 96)	0	block_1_depthw:
block_1_project (Conv2D)	(None, 32, 32, 24)	2,304	block_1_depthw:
block_1_project_BN (BatchNormalization)	(None, 32, 32, 24)	96	block_1_projec
block_2_expand (Conv2D)	(None, 32, 32, 144)	3,456	block_1_projec
block_2_expand_BN (BatchNormalization)	(None, 32, 32, 144)	576	block_2_expand
block_2_expand_relu (ReLU)	(None, 32, 32, 144)	0	block_2_expand_
block_2_depthwise (DepthwiseConv2D)	(None, 32, 32, 144)	1,296	block_2_expand_
block_2_depthwise_BN (BatchNormalization)	(None, 32, 32, 144)	576	block_2_depthw:

block_2_depthwise_relu (ReLU)	(None, 32, 32, 144)	0	block_2_depthw:
block_2_project (Conv2D)	(None, 32, 32, 24)	3,456	block_2_depthw:
block_2_project_BN (BatchNormalization)	(None, 32, 32, 24)	96	block_2_projec
block_2_add (Add)	(None, 32, 32, 24)	0	block_1_projec block_2_project
block_3_expand (Conv2D)	(None, 32, 32, 144)	3,456	block_2_add[0]
block_3_expand_BN (BatchNormalization)	(None, 32, 32, 144)	576	block_3_expand
block_3_expand_relu (ReLU)	(None, 32, 32, 144)	0	block_3_expand_
block_3_pad (ZeroPadding2D)	(None, 33, 33, 144)	0	block_3_expand_
block_3_depthwise (DepthwiseConv2D)	(None, 16, 16, 144)	1,296	block_3_pad[0]
block_3_depthwise_BN (BatchNormalization)	(None, 16, 16, 144)	576	block_3_depthw:
block_3_depthwise_relu (ReLU)	(None, 16, 16, 144)	0	block_3_depthw:
block_3_project (Conv2D)	(None, 16, 16, 32)	4,608	block_3_depthw:
block_3_project_BN (BatchNormalization)	(None, 16, 16, 32)	128	block_3_projec
block_4_expand (Conv2D)	(None, 16, 16, 192)	6,144	block_3_projec
block_4_expand_BN (BatchNormalization)	(None, 16, 16, 192)	768	block_4_expand
block_4_expand_relu (ReLU)	(None, 16, 16, 192)	0	block_4_expand_
block_4_depthwise (DepthwiseConv2D)	(None, 16, 16, 192)	1,728	block_4_expand_
block_4_depthwise_BN (BatchNormalization)	(None, 16, 16, 192)	768	block_4_depthw:
block_4_depthwise_relu (ReLU)	(None, 16, 16, 192)	0	block_4_depthw:
block_4_project (Conv2D)	(None, 16, 16, 32)	6,144	block_4_depthw:
block_4_project_BN (BatchNormalization)	(None, 16, 16, 32)	128	block_4_projec
block_4_add (Add)	(None, 16, 16, 32)	0	block_3_projec block_4_project
block_5_expand (Conv2D)	(None, 16, 16, 192)	6,144	block_4_add[0]
block_5_expand_BN (BatchNormalization)	(None, 16, 16, 192)	768	block_5_expand

block_5_expand_relu (ReLU)	(None, 16, 16, 192)	0	block_5_expand_
block_5_depthwise (DepthwiseConv2D)	(None, 16, 16, 192)	1,728	block_5_expand_
block_5_depthwise_BN (BatchNormalization)	(None, 16, 16, 192)	768	block_5_depthw:
block_5_depthwise_relu (ReLU)	(None, 16, 16, 192)	0	block_5_depthw:
block_5_project (Conv2D)	(None, 16, 16, 32)	6,144	block_5_depthw:
block_5_project_BN (BatchNormalization)	(None, 16, 16, 32)	128	block_5_projec
block_5_add (Add)	(None, 16, 16, 32)	0	block_4_add[0] block_5_project
block_6_expand (Conv2D)	(None, 16, 16, 192)	6,144	block_5_add[0]
block_6_expand_BN (BatchNormalization)	(None, 16, 16, 192)	768	block_6_expand
block_6_expand_relu (ReLU)	(None, 16, 16, 192)	0	block_6_expand_
block_6_pad (ZeroPadding2D)	(None, 17, 17, 192)	0	block_6_expand_
block_6_depthwise (DepthwiseConv2D)	(None, 8, 8, 192)	1,728	block_6_pad[0]
block_6_depthwise_BN (BatchNormalization)	(None, 8, 8, 192)	768	block_6_depthw:
block_6_depthwise_relu (ReLU)	(None, 8, 8, 192)	0	block_6_depthw:
block_6_project (Conv2D)	(None, 8, 8, 64)	12,288	block_6_depthw:
block_6_project_BN (BatchNormalization)	(None, 8, 8, 64)	256	block_6_projec
block_7_expand (Conv2D)	(None, 8, 8, 384)	24,576	block_6_projec
block_7_expand_BN (BatchNormalization)	(None, 8, 8, 384)	1,536	block_7_expand
block_7_expand_relu (ReLU)	(None, 8, 8, 384)	0	block_7_expand_
block_7_depthwise (DepthwiseConv2D)	(None, 8, 8, 384)	3,456	block_7_expand_
block_7_depthwise_BN (BatchNormalization)	(None, 8, 8, 384)	1,536	block_7_depthw:
block_7_depthwise_relu (ReLU)	(None, 8, 8, 384)	0	block_7_depthw:
block_7_project (Conv2D)	(None, 8, 8, 64)	24,576	block_7_depthw:
block_7_project_BN	(None, 8, 8, 64)	256	block_7_projec

(BatchNormalization)			
block_7_add (Add)	(None, 8, 8, 64)	0	block_6_projec block_7_project
block_8_expand (Conv2D)	(None, 8, 8, 384)	24,576	block_7_add[0]
block_8_expand_BN (BatchNormalization)	(None, 8, 8, 384)	1,536	block_8_expand
block_8_expand_relu (ReLU)	(None, 8, 8, 384)	0	block_8_expand_
block_8_depthwise (DepthwiseConv2D)	(None, 8, 8, 384)	3,456	block_8_expand_
block_8_depthwise_BN (BatchNormalization)	(None, 8, 8, 384)	1,536	block_8_depthw:
block_8_depthwise_relu (ReLU)	(None, 8, 8, 384)	0	block_8_depthw:
block_8_project (Conv2D)	(None, 8, 8, 64)	24,576	block_8_depthw:
block_8_project_BN (BatchNormalization)	(None, 8, 8, 64)	256	block_8_projec
block_8_add (Add)	(None, 8, 8, 64)	0	block_7_add[0] block_8_project
block_9_expand (Conv2D)	(None, 8, 8, 384)	24,576	block_8_add[0]
block_9_expand_BN (BatchNormalization)	(None, 8, 8, 384)	1,536	block_9_expand
block_9_expand_relu (ReLU)	(None, 8, 8, 384)	0	block_9_expand_
block_9_depthwise (DepthwiseConv2D)	(None, 8, 8, 384)	3,456	block_9_expand_
block_9_depthwise_BN (BatchNormalization)	(None, 8, 8, 384)	1,536	block_9_depthw:
block_9_depthwise_relu (ReLU)	(None, 8, 8, 384)	0	block_9_depthw:
block_9_project (Conv2D)	(None, 8, 8, 64)	24,576	block_9_depthw:
block_9_project_BN (BatchNormalization)	(None, 8, 8, 64)	256	block_9_projec
block_9_add (Add)	(None, 8, 8, 64)	0	block_8_add[0] block_9_project
block_10_expand (Conv2D)	(None, 8, 8, 384)	24,576	block_9_add[0]
block_10_expand_BN (BatchNormalization)	(None, 8, 8, 384)	1,536	block_10_expan
block_10_expand_relu (ReLU)	(None, 8, 8, 384)	0	block_10_expan
block_10_depthwise (DepthwiseConv2D)	(None, 8, 8, 384)	3,456	block_10_expan

block_10_depthwise_BN (BatchNormalization)	(None, 8, 8, 384)	1,536	block_10_depthw
block_10_depthwise_relu (ReLU)	(None, 8, 8, 384)	0	block_10_depthw
block_10_project (Conv2D)	(None, 8, 8, 96)	36,864	block_10_depthw
block_10_project_BN (BatchNormalization)	(None, 8, 8, 96)	384	block_10_proje
block_11_expand (Conv2D)	(None, 8, 8, 576)	55,296	block_10_proje
block_11_expand_BN (BatchNormalization)	(None, 8, 8, 576)	2,304	block_11_expan
block_11_expand_relu (ReLU)	(None, 8, 8, 576)	0	block_11_expan
block_11_depthwise (DepthwiseConv2D)	(None, 8, 8, 576)	5,184	block_11_expan
block_11_depthwise_BN (BatchNormalization)	(None, 8, 8, 576)	2,304	block_11_depthw
block_11_depthwise_relu (ReLU)	(None, 8, 8, 576)	0	block_11_depthw
block_11_project (Conv2D)	(None, 8, 8, 96)	55,296	block_11_depthw
block_11_project_BN (BatchNormalization)	(None, 8, 8, 96)	384	block_11_proje
block_11_add (Add)	(None, 8, 8, 96)	0	block_10_proje block_11_proje
block_12_expand (Conv2D)	(None, 8, 8, 576)	55,296	block_11_add[0
block_12_expand_BN (BatchNormalization)	(None, 8, 8, 576)	2,304	block_12_expan
block_12_expand_relu (ReLU)	(None, 8, 8, 576)	0	block_12_expan
block_12_depthwise (DepthwiseConv2D)	(None, 8, 8, 576)	5,184	block_12_expan
block_12_depthwise_BN (BatchNormalization)	(None, 8, 8, 576)	2,304	block_12_depthw
block_12_depthwise_relu (ReLU)	(None, 8, 8, 576)	0	block_12_depthw
block_12_project (Conv2D)	(None, 8, 8, 96)	55,296	block_12_depthw
block_12_project_BN (BatchNormalization)	(None, 8, 8, 96)	384	block_12_proje
block_12_add (Add)	(None, 8, 8, 96)	0	block_11_add[0 block_12_proje
block_13_expand (Conv2D)	(None, 8, 8, 576)	55,296	block_12_add[0
block_13_expand_BN (BatchNormalization)	(None, 8, 8, 576)	2,304	block_13_expan

block_13_expand_relu (ReLU)	(None, 8, 8, 576)	0	block_13_expan
block_13_pad (ZeroPadding2D)	(None, 9, 9, 576)	0	block_13_expan
block_13_depthwise (DepthwiseConv2D)	(None, 4, 4, 576)	5,184	block_13_pad[0
block_13_depthwise_BN (BatchNormalization)	(None, 4, 4, 576)	2,304	block_13_depthw
block_13_depthwise_relu (ReLU)	(None, 4, 4, 576)	0	block_13_depthw
block_13_project (Conv2D)	(None, 4, 4, 160)	92,160	block_13_depthw
block_13_project_BN (BatchNormalization)	(None, 4, 4, 160)	640	block_13_proje
block_14_expand (Conv2D)	(None, 4, 4, 960)	153,600	block_13_proje
block_14_expand_BN (BatchNormalization)	(None, 4, 4, 960)	3,840	block_14_expan
block_14_expand_relu (ReLU)	(None, 4, 4, 960)	0	block_14_expan
block_14_depthwise (DepthwiseConv2D)	(None, 4, 4, 960)	8,640	block_14_expan
block_14_depthwise_BN (BatchNormalization)	(None, 4, 4, 960)	3,840	block_14_depthw
block_14_depthwise_relu (ReLU)	(None, 4, 4, 960)	0	block_14_depthw
block_14_project (Conv2D)	(None, 4, 4, 160)	153,600	block_14_depthw
block_14_project_BN (BatchNormalization)	(None, 4, 4, 160)	640	block_14_proje
block_14_add (Add)	(None, 4, 4, 160)	0	block_13_proje block_14_proje
block_15_expand (Conv2D)	(None, 4, 4, 960)	153,600	block_14_add[0
block_15_expand_BN (BatchNormalization)	(None, 4, 4, 960)	3,840	block_15_expan
block_15_expand_relu (ReLU)	(None, 4, 4, 960)	0	block_15_expan
block_15_depthwise (DepthwiseConv2D)	(None, 4, 4, 960)	8,640	block_15_expan
block_15_depthwise_BN (BatchNormalization)	(None, 4, 4, 960)	3,840	block_15_depthw
block_15_depthwise_relu (ReLU)	(None, 4, 4, 960)	0	block_15_depthw
block_15_project (Conv2D)	(None, 4, 4, 160)	153,600	block_15_depthw
block_15_project_BN (BatchNormalization)	(None, 4, 4, 160)	640	block_15_proje

block_15_add (Add)	(None, 4, 4, 160)	0	block_14_add[0 block_15_projec
block_16_expand (Conv2D)	(None, 4, 4, 960)	153,600	block_15_add[0
block_16_expand_BN (BatchNormalization)	(None, 4, 4, 960)	3,840	block_16_expand
block_16_expand_relu (ReLU)	(None, 4, 4, 960)	0	block_16_expand
block_16_depthwise (DepthwiseConv2D)	(None, 4, 4, 960)	8,640	block_16_expand
block_16_depthwise_BN (BatchNormalization)	(None, 4, 4, 960)	3,840	block_16_depthw
block_16_depthwise_relu (ReLU)	(None, 4, 4, 960)	0	block_16_depthw
block_16_project (Conv2D)	(None, 4, 4, 320)	307,200	block_16_depthw
block_16_project_BN (BatchNormalization)	(None, 4, 4, 320)	1,280	block_16_projec
Conv_1 (Conv2D)	(None, 4, 4, 1280)	409,600	block_16_projec
Conv_1_bn (BatchNormalization)	(None, 4, 4, 1280)	5,120	Conv_1[0][0]
out_relu (ReLU)	(None, 4, 4, 1280)	0	Conv_1_bn[0][0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0	out_relu[0][0]
dense_2 (Dense)	(None, 128)	163,968	global_average_
dropout_1 (Dropout)	(None, 128)	0	dense_2[0][0]
dense_3 (Dense)	(None, 3)	387	dropout_1[0][0

Total params: 2,422,339 (9.24 MB)

Trainable params: 164,355 (642.01 KB)

Non-trainable params: 2,257,984 (8.61 MB)

6.3. Train the Model (Initial Training)

In [ ]:

```
# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the transfer learning model
history = transfer_model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=10,
    callbacks=[early_stopping]
)
```

```
Epoch 1/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 74s 301ms/step - accuracy: 0.4101 - loss: 1.2281 - val_accuracy:
0.5626 - val_loss: 0.9139
Epoch 2/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 39s 178ms/step - accuracy: 0.5235 - loss: 0.9352 - val_accuracy:
0.5990 - val_loss: 0.8575
Epoch 3/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 34s 157ms/step - accuracy: 0.5680 - loss: 0.8674 - val_accuracy:
0.6293 - val_loss: 0.7991
Epoch 4/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 35s 158ms/step - accuracy: 0.5872 - loss: 0.8490 - val_accuracy:
0.6465 - val_loss: 0.7890
Epoch 5/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 34s 155ms/step - accuracy: 0.6073 - loss: 0.8139 - val_accuracy:
0.6637 - val_loss: 0.7317
Epoch 6/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 41s 155ms/step - accuracy: 0.6282 - loss: 0.7808 - val_accuracy:
0.6616 - val_loss: 0.7268
Epoch 7/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 41s 155ms/step - accuracy: 0.6177 - loss: 0.7912 - val_accuracy:
0.6713 - val_loss: 0.7167
Epoch 8/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 35s 162ms/step - accuracy: 0.6409 - loss: 0.7625 - val_accuracy:
0.6981 - val_loss: 0.6955
Epoch 9/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 33s 152ms/step - accuracy: 0.6384 - loss: 0.7564 - val_accuracy:
0.6816 - val_loss: 0.6970
Epoch 10/10
213/213 ━━━━━━━━━━━━━━━━━━━━ 34s 157ms/step - accuracy: 0.6426 - loss: 0.7628 - val_accuracy:
0.6960 - val_loss: 0.6782
```

6.4. Fine-Tune the Model

After the initial training, we unfreeze the top layers of the base model for fine-tuning.

Unfreeze and Fine-Tune

In [ ]:

```
# Unfreeze the base model
base_model.trainable = True

# Compile with a lower learning rate for fine-tuning
transfer_model.compile(
    optimizer=Adam(learning_rate=0.0001),  # Lower learning rate for fine-tuning
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Fine-tune the model
history_fine_tune = transfer_model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=5,
    callbacks=[early_stopping]
)
```

```
Epoch 1/5
213/213 ━━━━━━━━━━━━━━━━━━━━ 104s 282ms/step - accuracy: 0.5431 - loss: 1.0005 - val_accuracy: 0.6527 - val_loss: 0.7326
Epoch 2/5
213/213 ━━━━━━━━━━━━━━━━━━━━ 38s 171ms/step - accuracy: 0.6701 - loss: 0.7380 - val_accuracy: 0.6458 - val_loss: 0.7697
Epoch 3/5
213/213 ━━━━━━━━━━━━━━━━━━━━ 39s 178ms/step - accuracy: 0.7013 - loss: 0.6804 - val_accuracy: 0.6479 - val_loss: 0.7591
Epoch 4/5
213/213 ━━━━━━━━━━━━━━━━━━━━ 37s 168ms/step - accuracy: 0.7287 - loss: 0.6322 - val_accuracy: 0.6382 - val_loss: 0.7691
Epoch 5/5
213/213 ━━━━━━━━━━━━━━━━━━━━ 38s 173ms/step - accuracy: 0.7507 - loss: 0.6066 - val_accuracy: 0.6761 - val_loss: 0.7103
```

6.5. Evaluate the Model

Evaluate on Test Data

In [ ]:

```
# Evaluate the model on the test set
test_loss, test_accuracy = transfer_model.evaluate(X_test, y_test, verbose=1)
print(f"Test Accuracy: {test_accuracy:.2f}")
```

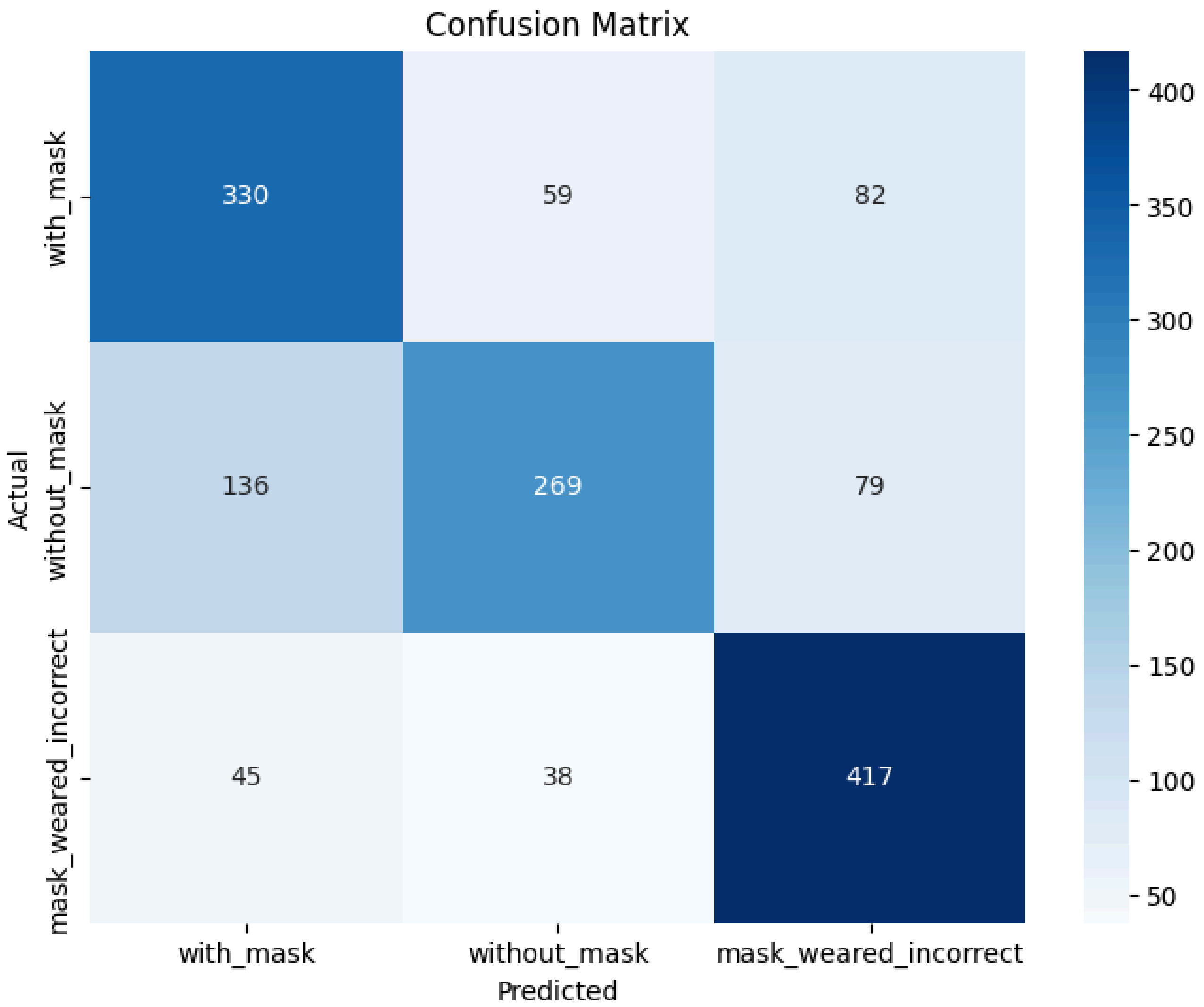
```
46/46 ━━━━━━━━━━━━━━━━━━━━ 6s 76ms/step - accuracy: 0.6857 - loss: 0.6785
Test Accuracy: 0.70
```

6.6. Classification Report

In [ ]:

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Predict on the test set
y_pred = transfer_model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Generate classification report
print("Classification Report:")
print(classification_report(y_true_classes, y_pred_classes, target_names=label_mapping.keys()

# Confusion matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=label_mapping.keys(), ytickla
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
46/46 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step
Classification Report:
                       precision    recall  f1-score   support

            with_mask       0.65      0.70      0.67       471
         without_mask       0.73      0.56      0.63       484
mask_weared_incorrect       0.72      0.83      0.77       500

             accuracy                           0.70      1455
            macro avg       0.70      0.70      0.69      1455
         weighted avg       0.70      0.70      0.69      1455
```

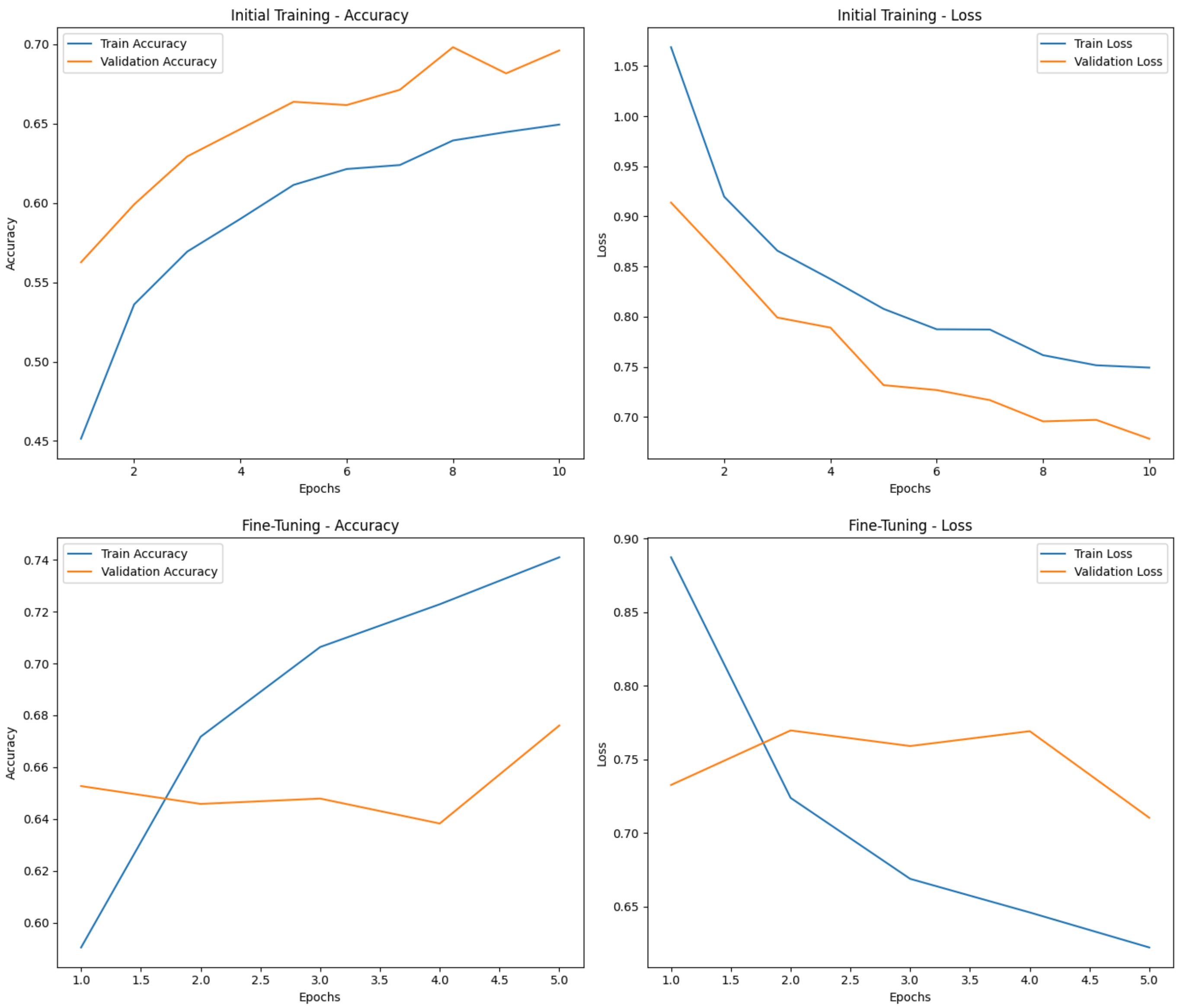


6.7. Visualize Training and Validation Metrics

Plot Accuracy and Loss

In [ ]:
```
# Function to plot training metrics
def plot_metrics(history, title):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(14, 6))

    # Accuracy Plot
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, label='Train Accuracy')
    plt.plot(epochs, val_acc, label='Validation Accuracy')
    plt.title(f'{title} - Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Loss Plot
    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, label='Train Loss')
    plt.plot(epochs, val_loss, label='Validation Loss')
    plt.title(f'{title} - Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
```

```
    plt.tight_layout()
    plt.show()

# Plot initial training metrics
plot_metrics(history, "Initial Training")

# Plot fine-tuning metrics
plot_metrics(history_fine_tune, "Fine-Tuning")
```



Overall Metrics

- Accuracy:
    - The overall percentage of correct predictions: 70%.
    - Indicates that the model is correct in 70% of all predictions.
- Macro Average:
- The unweighted average of precision, recall, and F1-score across all classes.
- Treats all classes equally, regardless of support.
- Example: Precision, recall, and F1-score for the macro average are all 70%, showing consistent performance across classes.
- Weighted Average:
- The average of precision, recall, and F1-score, weighted by the number of samples in each class (support).

- Accounts for class imbalance.
- Example: The weighted average is 70%, similar to the macro average, as the dataset is relatively balanced after preprocessing.

Class-Specific Insights

"with_mask":

- Precision: 65% (moderate; some false positives).
- Recall: 70% (better; most "with_mask" samples are correctly identified).
- F1-score: 67% (balanced but has room for improvement).

"without_mask":

- Precision: 73% (better; fewer false positives).
- Recall: 56% (lower; many "without_mask" samples are missed).
- F1-score: 63% (affected by low recall).

"mask_weared_incorrect":

- Precision: 72% (good; relatively fewer false positives).
- Recall: 83% (excellent; most samples are correctly identified).
- F1-score: 77% (highest among all classes, indicating strong performance).

Overall Interpretation

- **Strengths:**

The model performs best on the "mask_weared_incorrect" class, achieving a high F1-score of 77%. Balanced performance across all classes, with macro and weighted averages around 70%.

- **Weaknesses:**

Lower recall for "without_mask" indicates the model struggles to identify some "without_mask" samples. Precision for "with_mask" could improve, as there are still some false positives.

Conclusion

- **Use of Pre-Trained Model**:
  - We utilized a pre-trained model (e.g., MobileNetV2) as the base architecture for the face mask detection task.
  - Pre-trained models are trained on large datasets like ImageNet, allowing them to learn robust and generic feature representations.
  - Leveraging a pre-trained model significantly reduced the training time and computational cost compared to training a model from scratch.
- **Benefits of Transfer Learning**:
  - **Improved Performance**: Transfer learning allowed us to utilize well-learned features for tasks like edge detection, texture recognition, and pattern identification, which are crucial for identifying masks on faces.
  - **Reduced Data Dependency**: Using a pre-trained model made it possible to achieve high accuracy despite having a relatively smaller dataset.

    - **Faster Convergence**: The network required fewer epochs to converge, saving time and resources.
- **Project Outcomes**:
    - The use of a pre-trained model ensured high accuracy in detecting masks, achieving results that met project objectives.
    - Transfer learning proved to be an effective strategy for solving the problem with limited resources while maintaining excellent performance metrics.