```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os, glob, random, itertools
import pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as
sns
from tqdm.auto import tqdm
import tensorflow as tf
from tensorflow.keras.preprocessing import
image_dataset_from_directory


# ------------------------------------------------------------
# 1. Paths & constants
# ------------------------------------------------------------
DATASET_DIR = "/content/drive/MyDrive/ChestXRay2017/chest_xray"   #
<-- change if needed
IMG_SIZE    = (224, 224)
BATCH_SIZE  = 32
AUTOTUNE    = tf.data.AUTOTUNE

# ------------------------------------------------------------
# 2. Build a DataFrame of filepaths + clean labels
# ------------------------------------------------------------
def subclass_from_filename(fname: str) -> str:
    """Return 'bacterial', 'viral', or generic 'pneumonia'."""
    lower = fname.lower()
    if "_bacteria" in lower: return "bacterial"
    if "_virus"    in lower: return "viral"
    return "pneumonia"                # fallback

records = []
for split in ("train", "test"):
    for cls in ("NORMAL", "PNEUMONIA"):
        folder = os.path.join(DATASET_DIR, split, cls)
        for fp in glob.glob(os.path.join(folder, "*.jpeg")):
            label = "normal" if cls == "NORMAL" else
subclass_from_filename(fp)
            records.append((fp, label, split))

df = pd.DataFrame(records, columns=["filepath", "label", "split"])
print("Total images:", len(df))
df.head()
```

```
Total images: 5876
```

```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 5876,\n  \"fields\":
[\n    {\n      \"column\": \"filepath\",\n      \"properties\": {\n
```

```
\"dtype\": \"string\",\n        \"num_unique_values\": 5876,\n
\"samples\": [\n
\"/content/drive/MyDrive/ChestXRay2017/chest_xray/test/NORMAL/IM-0013-
0001.jpeg\",\n
\"/content/drive/MyDrive/ChestXRay2017/chest_xray/train/PNEUMONIA/
person1318_virus_2274.jpeg\",\n
\"/content/drive/MyDrive/ChestXRay2017/chest_xray/train/NORMAL/NORMAL2
-IM-0934-0001.jpeg\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"label\",\n      \"properties\": {\n        \"dtype\": \"category\",\
n        \"num_unique_values\": 3,\n        \"samples\": [\n
\"normal\",\n          \"bacterial\",\n          \"viral\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"split\",\n      \"properties\": {\n        \"dtype\": \"category\",\
n        \"num_unique_values\": 2,\n        \"samples\": [\n
\"test\",\n          \"train\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```

```
# ------------------------------------------------------------
# 3. Create a validation split
# ------------------------------------------------------------
from sklearn.model_selection import train_test_split

train_df = df[df.split=="train"]
train_idx, val_idx = train_test_split(
    train_df.index,
    test_size=0.15,
    stratify=train_df["label"],
    random_state=42)

df.loc[val_idx, "split"] = "val"
print(df["split"].value_counts())
```

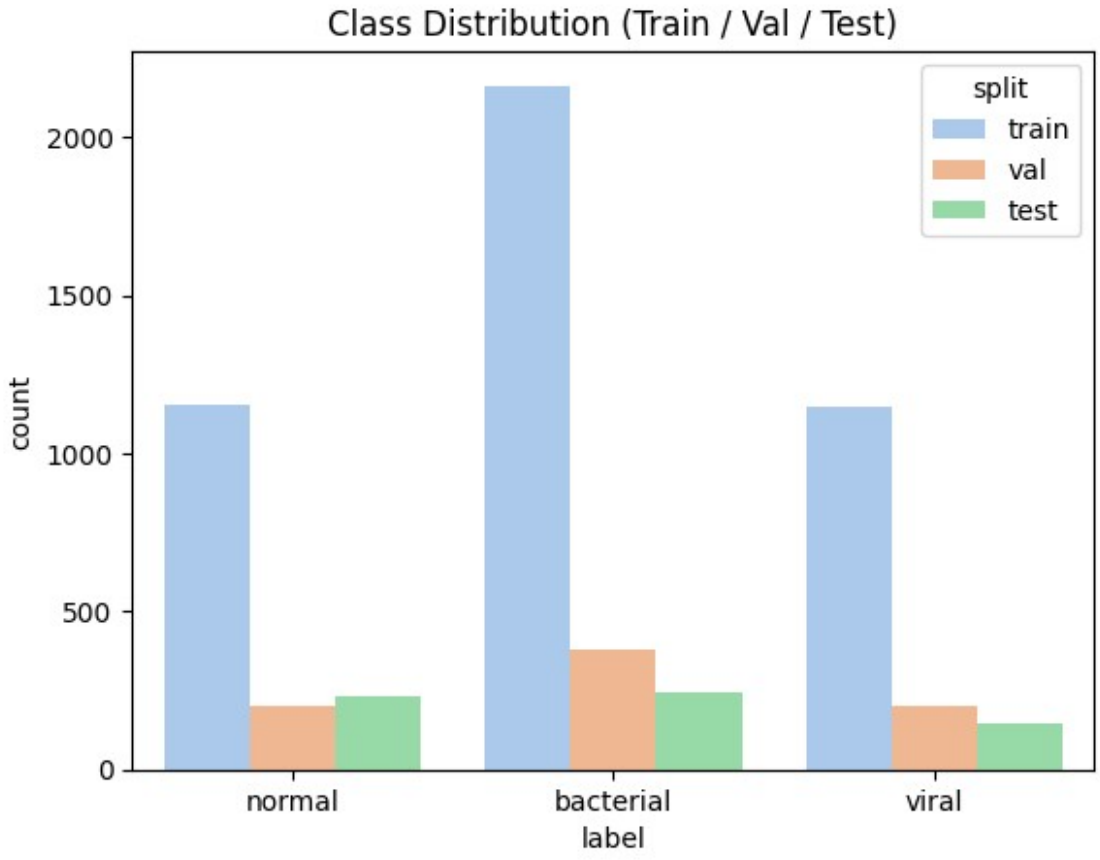
```
split
train    4464
val       788
test      624
Name: count, dtype: int64
```

Exploratory Data Analysis

Class distribution by split

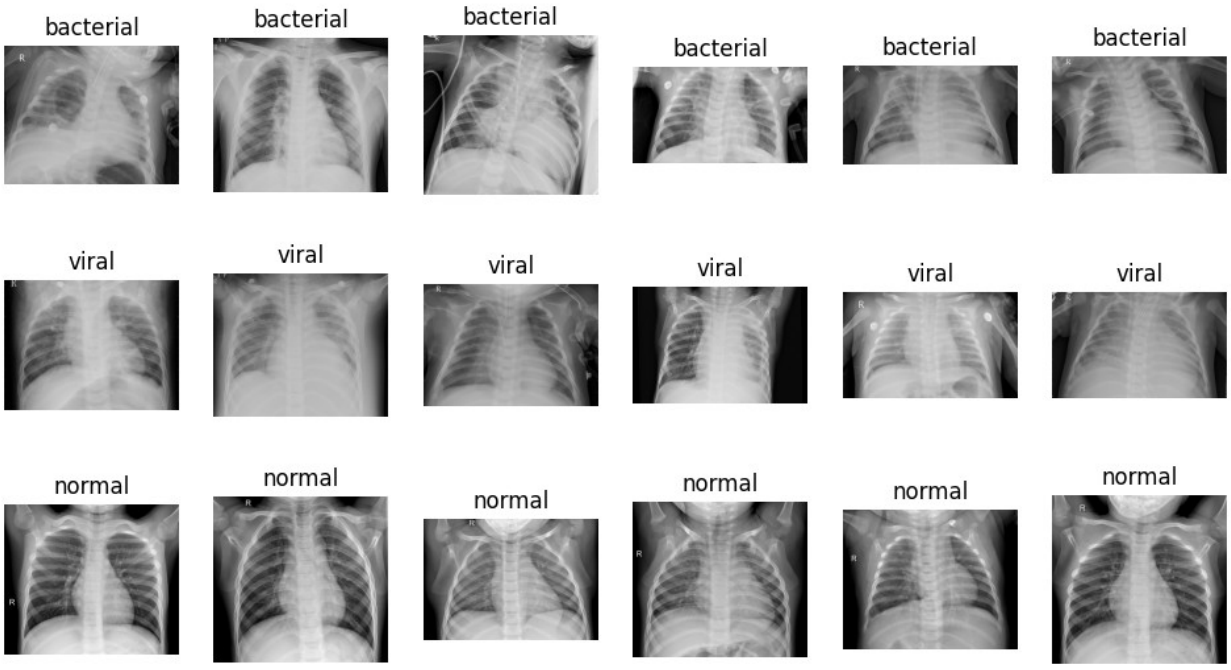
```
#   Class distribution by split
sns.countplot(x="label", hue="split", data=df, palette="pastel")
```

```
plt.title("Class Distribution (Train / Val / Test)")
plt.show()
```



Random image mosaics for sanity-check

```
# Random image mosaics for sanity-check
def show_random_samples(lbl="bacterial", n=6):
    sample_paths = df[(df.label==lbl) & (df.split=="train")]
["filepath"].sample(n)
    plt.figure(figsize=(12,2))
    for i, fp in enumerate(sample_paths, 1):
        img = plt.imread(fp)
        plt.subplot(1, n, i)
        plt.imshow(img, cmap="gray")
        plt.axis("off")
        plt.title(lbl)
    plt.show()

for lbl in ["bacterial", "viral", "normal"]:
    show_random_samples(lbl)
```




```
#  Image-size statistics
shapes = [tf.image.decode_jpeg(tf.io.read_file(fp), channels=3).shape
          for fp in tqdm(df["filepath"])]
shape_df = pd.DataFrame(shapes, columns=["h","w","c"])
print(shape_df.describe()[["h","w"]])
```

{"model_id":"759a0409fe904146811c39ffb004b0e2","version_major":2,"version_minor":0}

```
                 h            w
count  5876.00000  5876.000000
mean    970.44418  1327.361811
std     383.36124   363.491567
min     127.00000   384.000000
25%     688.00000  1056.000000
50%     887.00000  1280.000000
75%    1187.00000  1560.000000
max    2713.00000  2916.000000
```

Preprocessing

```
#  list of filepaths & labels → tf.data.Dataset
label_to_index = {lbl:i for i,lbl in
enumerate(sorted(df["label"].unique()))}
```

```
def paths_to_dataset(filepaths, labels):
    ds = tf.data.Dataset.from_tensor_slices((filepaths, labels))
    # -- map strings → image tensors + one-hot labels
    def _load(path, lbl):
        img = tf.io.read_file(path)
        img = tf.image.decode_jpeg(img, channels=3)
        img = tf.image.resize(img, IMG_SIZE)
        img = tf.cast(img, tf.float32) / 255.0         # [0,1]
        lbl = tf.one_hot(lbl, depth=len(label_to_index))
        return img, lbl
    ds = ds.map(_load, num_parallel_calls=AUTOTUNE)
    return ds

# Gather splits
train_paths = df[df.split=="train"]["filepath"].tolist()
train_labels= df[df.split=="train"]
["label"].map(label_to_index).tolist()

val_paths   = df[df.split=="val"]["filepath"].tolist()
val_labels  = df[df.split=="val"]
["label"].map(label_to_index).tolist()

test_paths  = df[df.split=="test"]["filepath"].tolist()
test_labels = df[df.split=="test"]
["label"].map(label_to_index).tolist()

# Build datasets
train_ds = paths_to_dataset(train_paths, train_labels)\
            .shuffle(1024).batch(BATCH_SIZE).prefetch(AUTOTUNE)

val_ds   = paths_to_dataset(val_paths, val_labels)\
            .batch(BATCH_SIZE).prefetch(AUTOTUNE)

test_ds  = paths_to_dataset(test_paths, test_labels)\
            .batch(BATCH_SIZE).prefetch(AUTOTUNE)

print("Dataset shapes →", next(iter(train_ds))[0].shape,
next(iter(train_ds))[1].shape)
```

```
Dataset shapes → (32, 224, 224, 3) (32, 3)
```

Model Implementation

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
```

```
import seaborn as sns
import time

# Dataset-specific values
NUM_CLASSES = 3
EPOCHS = 15
INPUT_SHAPE = (224, 224, 3)
class_names = ['WithMask', 'WithoutMask', 'MaskWearedIncorrect']
```

We Build Four models The four models include:

- DenseNet121 (Transfer Learning)
- Custom CNN (lightweight from scratch)
- MobileNetV2 (Transfer Learning)
- VGG16 (Transfer Learning)

```
# ---------------------------------------------------------------
# 1. DenseNet121 Model
# ---------------------------------------------------------------
def build_densenet_model(num_classes):
    base = tf.keras.applications.DenseNet121(
        include_top=False, weights="imagenet",
        input_shape=INPUT_SHAPE, pooling="avg")
    base.trainable = False
    inputs = layers.Input(shape=INPUT_SHAPE)
    x = base(inputs, training=False)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    return models.Model(inputs, outputs)

# ---------------------------------------------------------------
# 2. Custom CNN Model
# ---------------------------------------------------------------
def build_custom_cnn(num_classes):
    inputs = layers.Input(shape=INPUT_SHAPE)
    x = layers.Conv2D(32, 3, activation='relu', padding='same')
(inputs)
    x = layers.MaxPooling2D()(x)
    x = layers.Conv2D(64, 3, activation='relu', padding='same')(x)
    x = layers.MaxPooling2D()(x)
    x = layers.Conv2D(128, 3, activation='relu', padding='same')(x)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    return models.Model(inputs, outputs)
```

```
# ---------------------------------------------------------------
# 3. MobileNetV2 Model
# ---------------------------------------------------------------
def build_mobilenet_model(num_classes):
    base = tf.keras.applications.MobileNetV2(
        include_top=False, weights="imagenet",
        input_shape=INPUT_SHAPE, pooling="avg")
    base.trainable = False
    inputs = layers.Input(shape=INPUT_SHAPE)
    x = base(inputs, training=False)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    return models.Model(inputs, outputs)

# ---------------------------------------------------------------
# 4. VGG16 Model
# ---------------------------------------------------------------
def build_vgg16_model(num_classes):
    base = tf.keras.applications.VGG16(
        include_top=False, weights="imagenet",
        input_shape=INPUT_SHAPE, pooling="avg")
    base.trainable = False
    inputs = layers.Input(shape=INPUT_SHAPE)
    x = base(inputs, training=False)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    return models.Model(inputs, outputs)
```

Train and Evalute the model

```
# Compile and train
def compile_and_train(model, name, train_ds, val_ds, epochs=EPOCHS):
    model.compile(
        optimizer=tf.keras.optimizers.Adam(1e-4),
        loss='categorical_crossentropy',
        metrics=['accuracy',
                 tf.keras.metrics.AUC(name='auc'),
                 tf.keras.metrics.Precision(name='precision'),
                 tf.keras.metrics.Recall(name='recall')]
    )
    print(f"\\n Training {name}...")
    start_time = time.time()
    history = model.fit(train_ds, validation_data=val_ds,
epochs=epochs, verbose=2)
    train_time = time.time() - start_time
    print(f" Training time for {name}: {train_time:.2f} seconds")
    return model, history, train_time
```

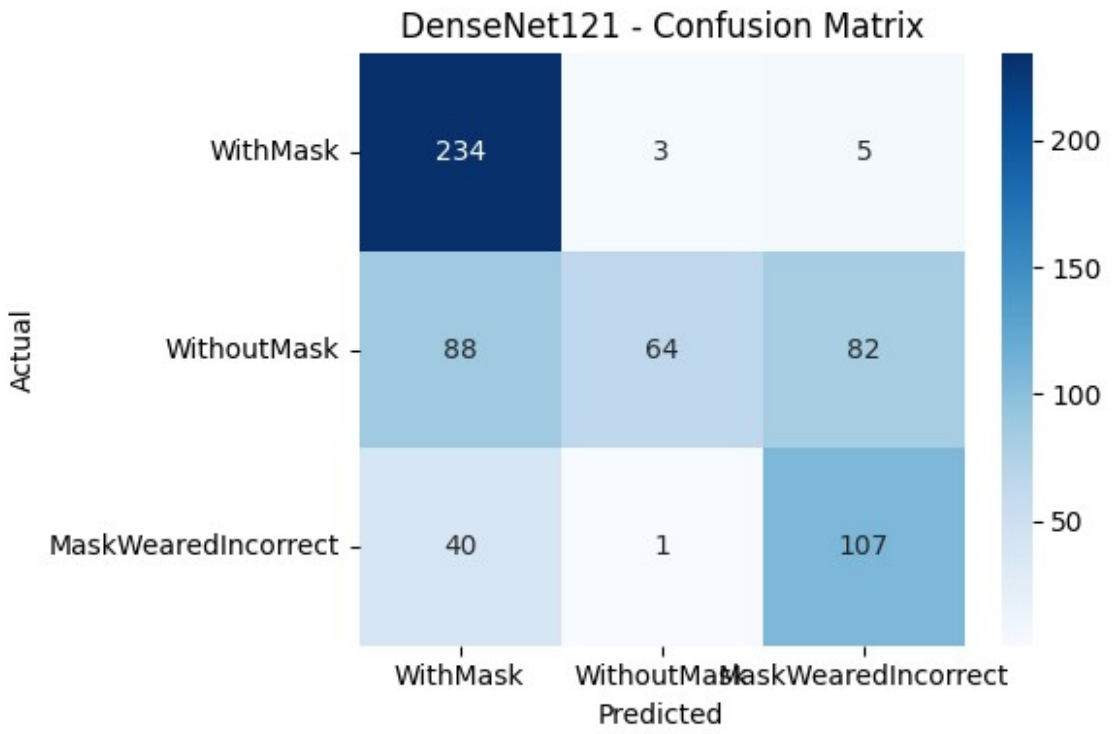
```
# Train and evaluate all models
models_to_train = {
    "DenseNet121": build_densenet_model(NUM_CLASSES),
    "Custom CNN": build_custom_cnn(NUM_CLASSES),
    "MobileNetV2": build_mobilenet_model(NUM_CLASSES),
    "VGG16": build_vgg16_model(NUM_CLASSES)
}

model_histories = {}

for model_name, model_instance in models_to_train.items():
    model, history, train_time = compile_and_train(model_instance,
model_name, train_ds, val_ds)

    # Evaluation
    print(f"\\n Evaluating {model_name}...")
    start_test = time.time()
    y_true, y_pred = [], []

    for x_batch, y_batch in test_ds:
        preds = model.predict(x_batch)
        y_pred.extend(np.argmax(preds, axis=1))
        y_true.extend(np.argmax(y_batch.numpy(), axis=1))

    test_time = time.time() - start_test
    print(f" Testing time: {test_time:.2f} seconds")

    print(" Classification Report:")
    print(classification_report(y_true, y_pred,
target_names=class_names))

    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title(f"{model_name} - Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.tight_layout()
    plt.show()

    # Training history
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.title(f'{model_name} - Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
```

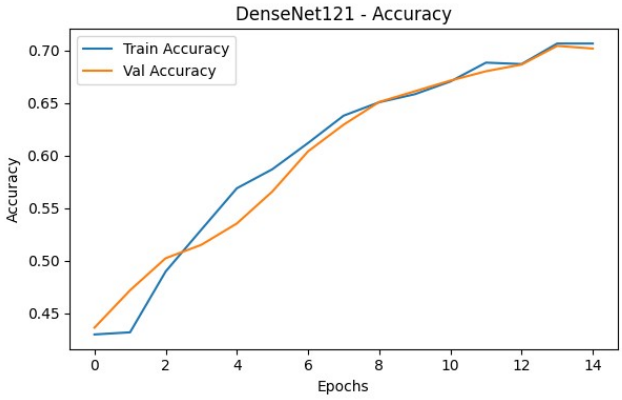
```
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title(f'{model_name} - Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

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

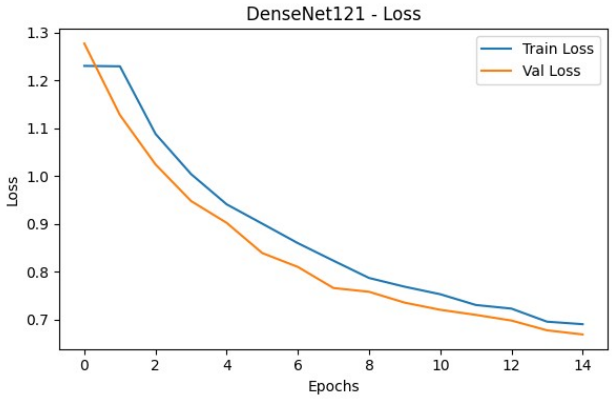
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/densenet/
densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/mobilenet_v2/
mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
\n Training DenseNet121...
Epoch 1/15
140/140 - 1547s - 11s/step - accuracy: 0.4301 - auc: 0.6179 - loss:
1.2300 - precision: 0.4524 - recall: 0.3564 - val_accuracy: 0.4365 -
val_auc: 0.6027 - val_loss: 1.2767 - val_precision: 0.4407 -
val_recall: 0.3769
Epoch 2/15
140/140 - 50s - 357ms/step - accuracy: 0.4321 - auc: 0.6144 - loss:
1.2291 - precision: 0.4645 - recall: 0.3560 - val_accuracy: 0.4721 -
val_auc: 0.6587 - val_loss: 1.1274 - val_precision: 0.5030 -
val_recall: 0.4213
Epoch 3/15
140/140 - 45s - 321ms/step - accuracy: 0.4899 - auc: 0.6782 - loss:
1.0881 - precision: 0.5286 - recall: 0.4097 - val_accuracy: 0.5025 -
val_auc: 0.7006 - val_loss: 1.0244 - val_precision: 0.5374 -
val_recall: 0.4467
Epoch 4/15
140/140 - 50s - 360ms/step - accuracy: 0.5296 - auc: 0.7209 - loss:
1.0041 - precision: 0.5793 - recall: 0.4568 - val_accuracy: 0.5152 -
val_auc: 0.7318 - val_loss: 0.9478 - val_precision: 0.5784 -
val_recall: 0.4683
Epoch 5/15
140/140 - 50s - 360ms/step - accuracy: 0.5690 - auc: 0.7535 - loss:
0.9410 - precision: 0.6089 - recall: 0.4908 - val_accuracy: 0.5355 -
val_auc: 0.7553 - val_loss: 0.9023 - val_precision: 0.5959 -
val_recall: 0.4848
```

```
Epoch 6/15
140/140 - 45s - 325ms/step - accuracy: 0.5871 - auc: 0.7734 - loss:
0.9007 - precision: 0.6313 - recall: 0.5085 - val_accuracy: 0.5660 -
val_auc: 0.7864 - val_loss: 0.8393 - val_precision: 0.6287 -
val_recall: 0.4898
Epoch 7/15
140/140 - 86s - 616ms/step - accuracy: 0.6120 - auc: 0.7947 - loss:
0.8600 - precision: 0.6507 - recall: 0.5341 - val_accuracy: 0.6041 -
val_auc: 0.8020 - val_loss: 0.8104 - val_precision: 0.6557 -
val_recall: 0.5051
Epoch 8/15
140/140 - 50s - 358ms/step - accuracy: 0.6380 - auc: 0.8123 - loss:
0.8234 - precision: 0.6827 - recall: 0.5668 - val_accuracy: 0.6294 -
val_auc: 0.8268 - val_loss: 0.7664 - val_precision: 0.7037 -
val_recall: 0.5305
Epoch 9/15
140/140 - 81s - 581ms/step - accuracy: 0.6508 - auc: 0.8277 - loss:
0.7873 - precision: 0.6898 - recall: 0.5883 - val_accuracy: 0.6510 -
val_auc: 0.8308 - val_loss: 0.7585 - val_precision: 0.7202 -
val_recall: 0.5520
Epoch 10/15
140/140 - 50s - 356ms/step - accuracy: 0.6584 - auc: 0.8355 - loss:
0.7691 - precision: 0.7012 - recall: 0.5992 - val_accuracy: 0.6612 -
val_auc: 0.8425 - val_loss: 0.7358 - val_precision: 0.7248 -
val_recall: 0.5749
Epoch 11/15
140/140 - 49s - 351ms/step - accuracy: 0.6705 - auc: 0.8421 - loss:
0.7534 - precision: 0.7090 - recall: 0.6037 - val_accuracy: 0.6713 -
val_auc: 0.8506 - val_loss: 0.7209 - val_precision: 0.7400 -
val_recall: 0.5850
Epoch 12/15
140/140 - 83s - 592ms/step - accuracy: 0.6884 - auc: 0.8522 - loss:
0.7309 - precision: 0.7240 - recall: 0.6263 - val_accuracy: 0.6802 -
val_auc: 0.8551 - val_loss: 0.7103 - val_precision: 0.7425 -
val_recall: 0.5964
Epoch 13/15
140/140 - 50s - 360ms/step - accuracy: 0.6871 - auc: 0.8546 - loss:
0.7234 - precision: 0.7205 - recall: 0.6248 - val_accuracy: 0.6865 -
val_auc: 0.8608 - val_loss: 0.6985 - val_precision: 0.7469 -
val_recall: 0.6104
Epoch 14/15
140/140 - 46s - 325ms/step - accuracy: 0.7065 - auc: 0.8656 - loss:
0.6961 - precision: 0.7366 - recall: 0.6514 - val_accuracy: 0.7043 -
val_auc: 0.8699 - val_loss: 0.6781 - val_precision: 0.7577 -
val_recall: 0.6269
Epoch 15/15
140/140 - 86s - 617ms/step - accuracy: 0.7065 - auc: 0.8670 - loss:
0.6909 - precision: 0.7408 - recall: 0.6503 - val_accuracy: 0.7018 -
val_auc: 0.8741 - val_loss: 0.6695 - val_precision: 0.7602 -
```

```
val_recall: 0.6358
 Training time for DenseNet121: 2402.36 seconds
\n Evaluating DenseNet121...
1/1 ━━━━━━━━━━━━━━━━━━━━ 12s 12s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 139ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 131ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 12s 12s/step
 Testing time: 269.43 seconds
 Classification Report:
                     precision    recall  f1-score   support

           WithMask       0.65      0.97      0.77       242
        WithoutMask       0.94      0.27      0.42       234
MaskWearedIncorrect       0.55      0.72      0.63       148

           accuracy                           0.65       624
          macro avg       0.71      0.65      0.61       624
       weighted avg       0.73      0.65      0.61       624
```



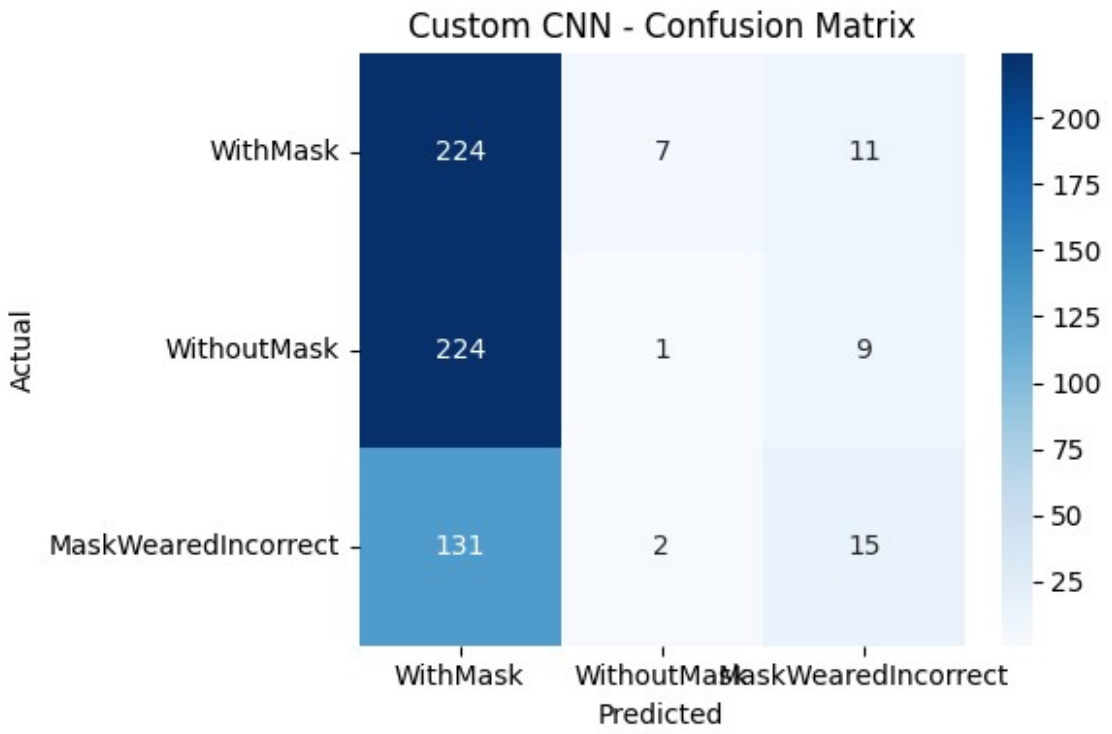


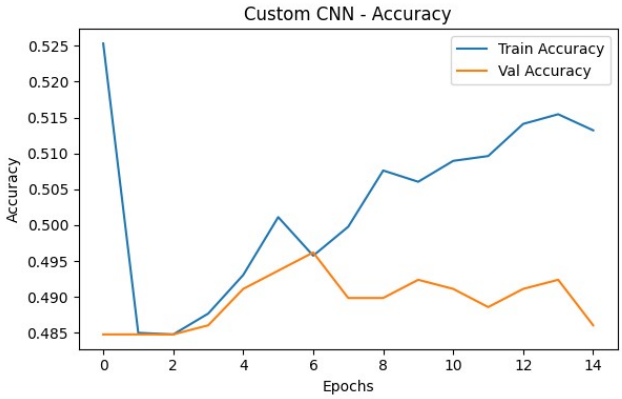


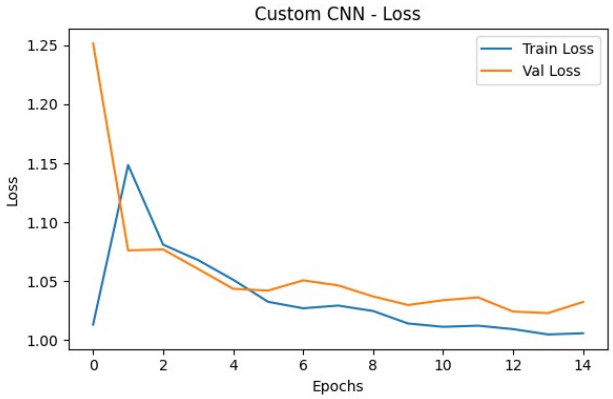

```
\n Training Custom CNN...
Epoch 1/15
140/140 - 58s - 416ms/step - accuracy: 0.5253 - auc: 0.6735 - loss:
1.0133 - precision: 0.5494 - recall: 0.2218 - val_accuracy: 0.4848 -
val_auc: 0.5928 - val_loss: 1.2514 - val_precision: 0.4848 -
val_recall: 0.4848
Epoch 2/15
140/140 - 70s - 499ms/step - accuracy: 0.4850 - auc: 0.5666 - loss:
1.1485 - precision: 0.4055 - recall: 0.0961 - val_accuracy: 0.4848 -
val_auc: 0.5951 - val_loss: 1.0761 - val_precision: 0.4784 -
val_recall: 0.4505
Epoch 3/15
140/140 - 82s - 586ms/step - accuracy: 0.4848 - auc: 0.5821 - loss:
1.0810 - precision: 0.4468 - recall: 0.0903 - val_accuracy: 0.4848 -
```

```
val_auc: 0.6030 - val_loss: 1.0770 - val_precision: 0.4837 -
val_recall: 0.4695
Epoch 4/15
140/140 - 48s - 345ms/step - accuracy: 0.4877 - auc: 0.6106 - loss:
1.0679 - precision: 0.4910 - recall: 0.1102 - val_accuracy: 0.4860 -
val_auc: 0.6236 - val_loss: 1.0604 - val_precision: 0.4909 -
val_recall: 0.4467
Epoch 5/15
140/140 - 48s - 339ms/step - accuracy: 0.4931 - auc: 0.6288 - loss:
1.0513 - precision: 0.5548 - recall: 0.1485 - val_accuracy: 0.4911 -
val_auc: 0.6461 - val_loss: 1.0436 - val_precision: 0.5251 -
val_recall: 0.4112
Epoch 6/15
140/140 - 49s - 349ms/step - accuracy: 0.5011 - auc: 0.6551 - loss:
1.0326 - precision: 0.6097 - recall: 0.1980 - val_accuracy: 0.4937 -
val_auc: 0.6582 - val_loss: 1.0421 - val_precision: 0.5295 -
val_recall: 0.4213
Epoch 7/15
140/140 - 43s - 307ms/step - accuracy: 0.4957 - auc: 0.6617 - loss:
1.0271 - precision: 0.6186 - recall: 0.2343 - val_accuracy: 0.4962 -
val_auc: 0.6593 - val_loss: 1.0507 - val_precision: 0.5263 -
val_recall: 0.4315
Epoch 8/15
140/140 - 88s - 627ms/step - accuracy: 0.4998 - auc: 0.6619 - loss:
1.0294 - precision: 0.6153 - recall: 0.2361 - val_accuracy: 0.4898 -
val_auc: 0.6619 - val_loss: 1.0465 - val_precision: 0.5398 -
val_recall: 0.4213
Epoch 9/15
140/140 - 77s - 548ms/step - accuracy: 0.5076 - auc: 0.6686 - loss:
1.0248 - precision: 0.6228 - recall: 0.2534 - val_accuracy: 0.4898 -
val_auc: 0.6701 - val_loss: 1.0371 - val_precision: 0.5453 -
val_recall: 0.4201
Epoch 10/15
140/140 - 44s - 315ms/step - accuracy: 0.5060 - auc: 0.6785 - loss:
1.0142 - precision: 0.6296 - recall: 0.2688 - val_accuracy: 0.4924 -
val_auc: 0.6742 - val_loss: 1.0299 - val_precision: 0.5561 -
val_recall: 0.4150
Epoch 11/15
140/140 - 49s - 348ms/step - accuracy: 0.5090 - auc: 0.6805 - loss:
1.0114 - precision: 0.6325 - recall: 0.2968 - val_accuracy: 0.4911 -
val_auc: 0.6753 - val_loss: 1.0339 - val_precision: 0.5517 -
val_recall: 0.4201
Epoch 12/15
140/140 - 80s - 570ms/step - accuracy: 0.5096 - auc: 0.6803 - loss:
1.0124 - precision: 0.6239 - recall: 0.3047 - val_accuracy: 0.4886 -
val_auc: 0.6767 - val_loss: 1.0363 - val_precision: 0.5415 -
val_recall: 0.4226
Epoch 13/15
140/140 - 44s - 311ms/step - accuracy: 0.5141 - auc: 0.6856 - loss:
```

```
1.0094 - precision: 0.6301 - recall: 0.3060 - val_accuracy: 0.4911 -
val_auc: 0.6802 - val_loss: 1.0243 - val_precision: 0.5621 -
val_recall: 0.4137
Epoch 14/15
140/140 - 48s - 344ms/step - accuracy: 0.5155 - auc: 0.6869 - loss:
1.0049 - precision: 0.6306 - recall: 0.3163 - val_accuracy: 0.4924 -
val_auc: 0.6806 - val_loss: 1.0230 - val_precision: 0.5613 -
val_recall: 0.4124
Epoch 15/15
140/140 - 80s - 573ms/step - accuracy: 0.5132 - auc: 0.6868 - loss:
1.0059 - precision: 0.6293 - recall: 0.3194 - val_accuracy: 0.4860 -
val_auc: 0.6818 - val_loss: 1.0324 - val_precision: 0.5477 -
val_recall: 0.4226
 Training time for Custom CNN: 907.24 seconds
\n Evaluating Custom CNN...
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 701ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 198ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 202ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 161ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 202ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 126ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 124ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 114ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 759ms/step
 Testing time: 8.33 seconds
 Classification Report:
                     precision    recall  f1-score   support

           WithMask       0.39      0.93      0.55       242
        WithoutMask       0.10      0.00      0.01       234
MaskWearedIncorrect       0.43      0.10      0.16       148

           accuracy                           0.38       624
          macro avg       0.31      0.34      0.24       624
       weighted avg       0.29      0.38      0.25       624
```



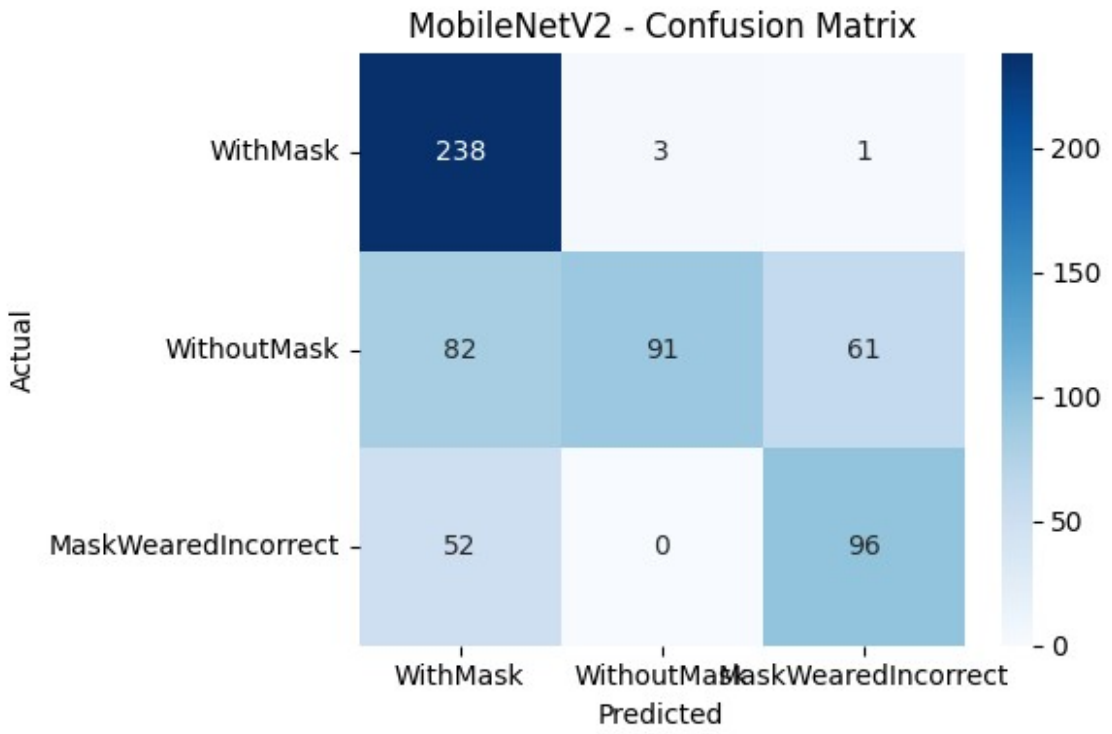


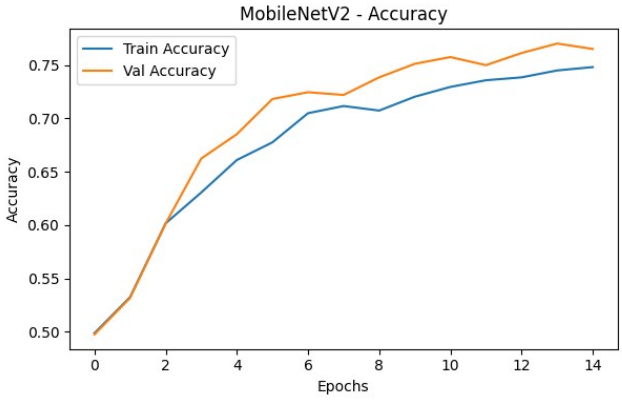


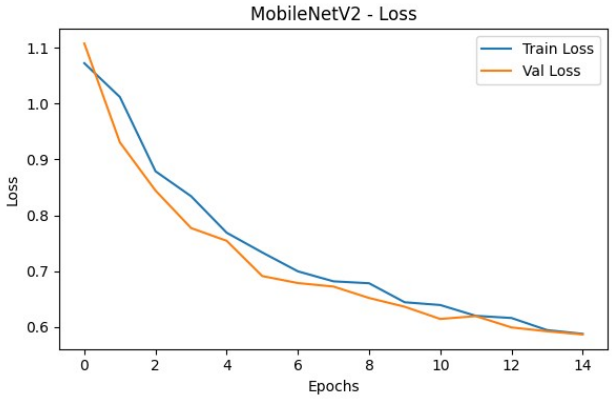

```
\n Training MobileNetV2...
Epoch 1/15
140/140 - 74s - 526ms/step - accuracy: 0.4987 - auc: 0.6950 - loss:
1.0723 - precision: 0.5254 - recall: 0.4330 - val_accuracy: 0.4975 -
val_auc: 0.6939 - val_loss: 1.1075 - val_precision: 0.5392 -
val_recall: 0.4708
Epoch 2/15
140/140 - 60s - 428ms/step - accuracy: 0.5325 - auc: 0.7203 - loss:
1.0117 - precision: 0.5793 - recall: 0.4581 - val_accuracy: 0.5317 -
val_auc: 0.7514 - val_loss: 0.9305 - val_precision: 0.6164 -
val_recall: 0.4772
Epoch 3/15
140/140 - 48s - 342ms/step - accuracy: 0.6017 - auc: 0.7855 - loss:
0.8788 - precision: 0.6504 - recall: 0.5293 - val_accuracy: 0.6015 -
```

```
val_auc: 0.7955 - val_loss: 0.8443 - val_precision: 0.6606 -
val_recall: 0.5089
Epoch 4/15
140/140 - 44s - 313ms/step - accuracy: 0.6306 - auc: 0.8081 - loss:
0.8342 - precision: 0.6667 - recall: 0.5627 - val_accuracy: 0.6624 -
val_auc: 0.8299 - val_loss: 0.7772 - val_precision: 0.7129 -
val_recall: 0.5546
Epoch 5/15
140/140 - 44s - 315ms/step - accuracy: 0.6611 - auc: 0.8367 - loss:
0.7688 - precision: 0.6967 - recall: 0.5995 - val_accuracy: 0.6853 -
val_auc: 0.8417 - val_loss: 0.7543 - val_precision: 0.7237 -
val_recall: 0.5850
Epoch 6/15
140/140 - 48s - 345ms/step - accuracy: 0.6776 - auc: 0.8502 - loss:
0.7335 - precision: 0.7159 - recall: 0.6243 - val_accuracy: 0.7183 -
val_auc: 0.8687 - val_loss: 0.6912 - val_precision: 0.7640 -
val_recall: 0.6409
Epoch 7/15
140/140 - 52s - 375ms/step - accuracy: 0.7050 - auc: 0.8643 - loss:
0.6997 - precision: 0.7382 - recall: 0.6508 - val_accuracy: 0.7246 -
val_auc: 0.8733 - val_loss: 0.6787 - val_precision: 0.7595 -
val_recall: 0.6574
Epoch 8/15
140/140 - 78s - 554ms/step - accuracy: 0.7117 - auc: 0.8710 - loss:
0.6818 - precision: 0.7420 - recall: 0.6662 - val_accuracy: 0.7221 -
val_auc: 0.8777 - val_loss: 0.6725 - val_precision: 0.7674 -
val_recall: 0.6701
Epoch 9/15
140/140 - 81s - 576ms/step - accuracy: 0.7074 - auc: 0.8718 - loss:
0.6784 - precision: 0.7388 - recall: 0.6658 - val_accuracy: 0.7386 -
val_auc: 0.8850 - val_loss: 0.6520 - val_precision: 0.7763 -
val_recall: 0.6827
Epoch 10/15
140/140 - 84s - 599ms/step - accuracy: 0.7204 - auc: 0.8837 - loss:
0.6444 - precision: 0.7476 - recall: 0.6756 - val_accuracy: 0.7513 -
val_auc: 0.8897 - val_loss: 0.6364 - val_precision: 0.7779 -
val_recall: 0.6980
Epoch 11/15
140/140 - 77s - 553ms/step - accuracy: 0.7296 - auc: 0.8860 - loss:
0.6395 - precision: 0.7561 - recall: 0.6891 - val_accuracy: 0.7576 -
val_auc: 0.8977 - val_loss: 0.6145 - val_precision: 0.7938 -
val_recall: 0.7132
Epoch 12/15
140/140 - 48s - 346ms/step - accuracy: 0.7359 - auc: 0.8920 - loss:
0.6200 - precision: 0.7605 - recall: 0.6978 - val_accuracy: 0.7500 -
val_auc: 0.8953 - val_loss: 0.6196 - val_precision: 0.7793 -
val_recall: 0.7081
Epoch 13/15
140/140 - 82s - 583ms/step - accuracy: 0.7386 - auc: 0.8934 - loss:
```

```
0.6162 - precision: 0.7596 - recall: 0.7036 - val_accuracy: 0.7614 -
val_auc: 0.9022 - val_loss: 0.5993 - val_precision: 0.7852 -
val_recall: 0.7145
Epoch 14/15
140/140 - 45s - 321ms/step - accuracy: 0.7451 - auc: 0.9004 - loss:
0.5946 - precision: 0.7670 - recall: 0.7115 - val_accuracy: 0.7703 -
val_auc: 0.9051 - val_loss: 0.5924 - val_precision: 0.8020 -
val_recall: 0.7297
Epoch 15/15
140/140 - 49s - 350ms/step - accuracy: 0.7482 - auc: 0.9022 - loss:
0.5878 - precision: 0.7721 - recall: 0.7126 - val_accuracy: 0.7652 -
val_auc: 0.9059 - val_loss: 0.5868 - val_precision: 0.7862 -
val_recall: 0.7234
 Training time for MobileNetV2: 913.83 seconds
\n Evaluating MobileNetV2...
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 209ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 125ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 168ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 210ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 214ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 161ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
 Testing time: 16.46 seconds
 Classification Report:
                     precision    recall  f1-score   support

           WithMask       0.64      0.98      0.78       242
        WithoutMask       0.97      0.39      0.55       234
MaskWearedIncorrect       0.61      0.65      0.63       148

           accuracy                           0.68       624
          macro avg       0.74      0.67      0.65       624
       weighted avg       0.76      0.68      0.66       624
```





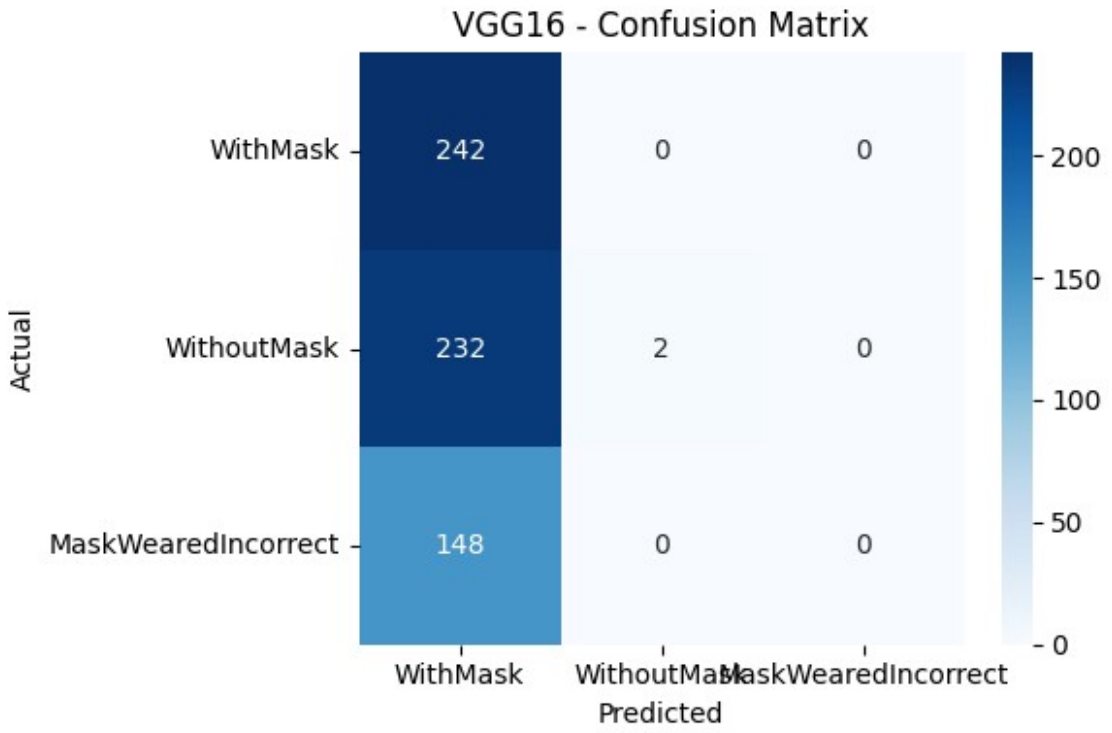


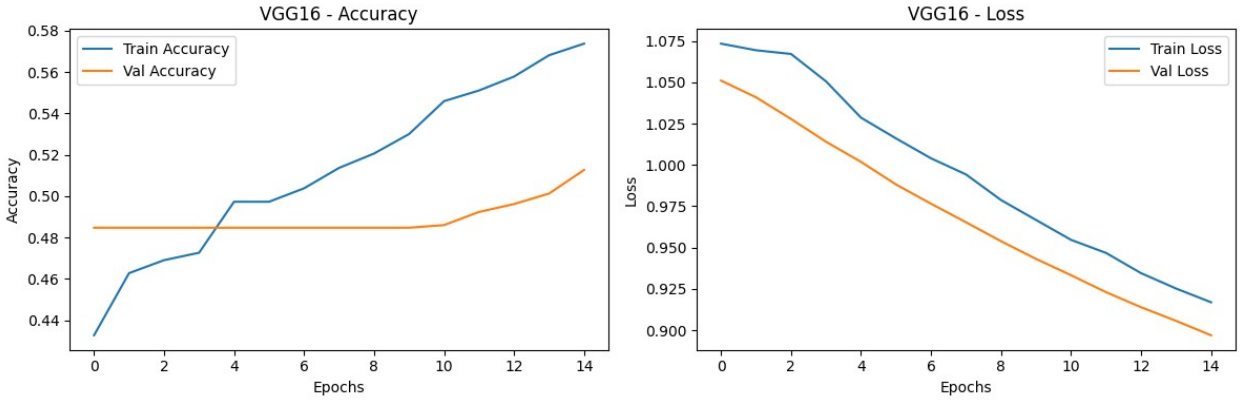

```
\n Training VGG16...
Epoch 1/15
140/140 - 86s - 613ms/step - accuracy: 0.4328 - auc: 0.6011 - loss:
1.0734 - precision: 0.5085 - recall: 0.1333 - val_accuracy: 0.4848 -
val_auc: 0.6287 - val_loss: 1.0510 - val_precision: 0.4841 -
val_recall: 0.3680
Epoch 2/15
140/140 - 53s - 381ms/step - accuracy: 0.4628 - auc: 0.6138 - loss:
1.0694 - precision: 0.4911 - recall: 0.2092 - val_accuracy: 0.4848 -
val_auc: 0.6588 - val_loss: 1.0409 - val_precision: 0.4895 -
val_recall: 0.4416
Epoch 3/15
140/140 - 53s - 381ms/step - accuracy: 0.4691 - auc: 0.6178 - loss:
1.0671 - precision: 0.4702 - recall: 0.1960 - val_accuracy: 0.4848 -
```

```
val_auc: 0.6837 - val_loss: 1.0278 - val_precision: 0.4966 -
val_recall: 0.4670
Epoch 4/15
140/140 - 53s - 376ms/step - accuracy: 0.4727 - auc: 0.6373 - loss:
1.0505 - precision: 0.4948 - recall: 0.2126 - val_accuracy: 0.4848 -
val_auc: 0.7018 - val_loss: 1.0140 - val_precision: 0.5155 -
val_recall: 0.4645
Epoch 5/15
140/140 - 48s - 343ms/step - accuracy: 0.4973 - auc: 0.6602 - loss:
1.0286 - precision: 0.5447 - recall: 0.2348 - val_accuracy: 0.4848 -
val_auc: 0.7136 - val_loss: 1.0018 - val_precision: 0.5396 -
val_recall: 0.4492
Epoch 6/15
140/140 - 85s - 610ms/step - accuracy: 0.4973 - auc: 0.6736 - loss:
1.0160 - precision: 0.5624 - recall: 0.2493 - val_accuracy: 0.4848 -
val_auc: 0.7279 - val_loss: 0.9881 - val_precision: 0.5800 -
val_recall: 0.4416
Epoch 7/15
140/140 - 53s - 382ms/step - accuracy: 0.5038 - auc: 0.6869 - loss:
1.0040 - precision: 0.5870 - recall: 0.2540 - val_accuracy: 0.4848 -
val_auc: 0.7385 - val_loss: 0.9765 - val_precision: 0.6049 -
val_recall: 0.4353
Epoch 8/15
140/140 - 53s - 381ms/step - accuracy: 0.5137 - auc: 0.6964 - loss:
0.9942 - precision: 0.5975 - recall: 0.2760 - val_accuracy: 0.4848 -
val_auc: 0.7496 - val_loss: 0.9653 - val_precision: 0.6225 -
val_recall: 0.4289
Epoch 9/15
140/140 - 51s - 361ms/step - accuracy: 0.5206 - auc: 0.7132 - loss:
0.9787 - precision: 0.6258 - recall: 0.2776 - val_accuracy: 0.4848 -
val_auc: 0.7601 - val_loss: 0.9538 - val_precision: 0.6523 -
val_recall: 0.4213
Epoch 10/15
140/140 - 81s - 582ms/step - accuracy: 0.5300 - auc: 0.7260 - loss:
0.9667 - precision: 0.6357 - recall: 0.2928 - val_accuracy: 0.4848 -
val_auc: 0.7729 - val_loss: 0.9431 - val_precision: 0.6543 -
val_recall: 0.4251
Epoch 11/15
140/140 - 53s - 377ms/step - accuracy: 0.5459 - auc: 0.7371 - loss:
0.9547 - precision: 0.6476 - recall: 0.2988 - val_accuracy: 0.4860 -
val_auc: 0.7743 - val_loss: 0.9333 - val_precision: 0.6800 -
val_recall: 0.4099
Epoch 12/15
140/140 - 49s - 351ms/step - accuracy: 0.5511 - auc: 0.7445 - loss:
0.9467 - precision: 0.6679 - recall: 0.3109 - val_accuracy: 0.4924 -
val_auc: 0.7853 - val_loss: 0.9230 - val_precision: 0.6792 -
val_recall: 0.4137
Epoch 13/15
140/140 - 84s - 598ms/step - accuracy: 0.5578 - auc: 0.7566 - loss:
```

```
0.9345 - precision: 0.6824 - recall: 0.3239 - val_accuracy: 0.4962 -
val_auc: 0.7884 - val_loss: 0.9139 - val_precision: 0.6799 -
val_recall: 0.4124
Epoch 14/15
140/140 - 54s - 383ms/step - accuracy: 0.5681 - auc: 0.7628 - loss:
0.9252 - precision: 0.6763 - recall: 0.3286 - val_accuracy: 0.5013 -
val_auc: 0.7934 - val_loss: 0.9057 - val_precision: 0.6728 -
val_recall: 0.4150
Epoch 15/15
140/140 - 54s - 383ms/step - accuracy: 0.5737 - auc: 0.7691 - loss:
0.9169 - precision: 0.6922 - recall: 0.3562 - val_accuracy: 0.5127 -
val_auc: 0.7949 - val_loss: 0.8969 - val_precision: 0.7007 -
val_recall: 0.4099
 Training time for VGG16: 910.18 seconds
\n Evaluating VGG16...
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 212ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 264ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 210ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 194ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 194ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 189ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 209ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 219ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 275ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 221ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 210ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 202ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
 Testing time: 9.80 seconds
 Classification Report:
                     precision    recall  f1-score   support

           WithMask       0.39      1.00      0.56       242
        WithoutMask       1.00      0.01      0.02       234
MaskWearedIncorrect       0.00      0.00      0.00       148

           accuracy                           0.39       624
          macro avg       0.46      0.34      0.19       624
       weighted avg       0.53      0.39      0.22       624


/usr/local/lib/python3.11/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-
```

```
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```




VGG16 - Accuracy
Train Accuracy
Val Accuracy
Accuracy
Epochs
VGG16 - Loss
Train Loss
Val Loss
Loss
Epochs