

```
In [1]: # =====
# Step 1: Mount Google Drive
# This step mounts Google Drive to access the dataset,
# save trained models, and store experiment results.
# =====

from google.colab import drive

drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [2]: # =====
# Step 2: Install Required Libraries
# This step installs all required Python libraries used
# throughout the ASL recognition project.
# =====

!pip -q install split-folders
!pip -q install textblob
```

```
In [3]: # =====
# Step 3: Import Required Libraries
# This step imports all libraries required for data
# preprocessing, model training, evaluation, visualization,
# and result analysis.
# =====

import os
import random
import shutil
import time
import json
import warnings

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from PIL import Image

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input

from tensorflow.keras.models import Model

from tensorflow.keras.layers import (
    Dense,
    Dropout,
    GlobalAveragePooling2D
)

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import (
    EarlyStopping,
    ModelCheckpoint,
    ReduceLROnPlateau
)

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

from textblob import TextBlob

warnings.filterwarnings("ignore")

print("=" * 60)
print("TensorFlow Version :", tf.__version__)
print("=" * 60)

=====
TensorFlow Version : 2.20.0
=====
```

```
In [4]: # =====
# Step 4: Set Random Seed and Verify GPU Configuration
# This step sets a fixed random seed to ensure reproducible
# training results and verifies the TensorFlow version and
# GPU availability before model training.
# =====

SEED = 42

# Python random seed
random.seed(SEED)

# NumPy random seed
np.random.seed(SEED)

# TensorFlow random seed
tf.random.set_seed(SEED)

print("=" * 60)
print("Project Configuration")
print("=" * 60)
print(f"Random Seed      : {SEED}")
print(f"TensorFlow Version: {tf.__version__}")

gpus = tf.config.list_physical_devices('GPU')

if gpus:
    print(f"GPU Available      : Yes")
    print(f"GPU Device         : {gpus[0].name}")
else:
    print("GPU Available      : No")
    print("Please enable T4 GPU before training.")

print("=" * 60)

=====
Project Configuration
=====
Random Seed      : 42
TensorFlow Version: 2.20.0
GPU Available     : Yes
GPU Device       : /physical_device:GPU:0
=====
```

In [5]:

```
# =====
# Step 5: Define Dataset Path
# This step locates the dataset directory inside Google Drive,
# verifies that it exists, and displays the available class
# folders before data preparation.
# =====

DATASET_DIR = "/content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset"

print("=" * 60)
print("Dataset Configuration")
print("=" * 60)

if os.path.isdir(DATASET_DIR):
    print("Dataset Status : Found")
    print(f"Dataset Path   : {DATASET_DIR}")

    class_names = sorted(os.listdir(DATASET_DIR))

    print(f"Total Classes   : {len(class_names)}")
    print(f"Class Names      : {class_names}")

else:
    raise FileNotFoundError(
        f"\nDataset not found!\nPlease check the following path:\n{DATASET_DIR}"
    )

print("=" * 60)

=====
Dataset Configuration
=====
Dataset Status : Found
Dataset Path   : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset
Total Classes  : 36
Class Names    : ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
=====
```

In [6]:

```
# =====
# Step 6: Verify Dataset Classes
# This step verifies that all expected ASL class folders
# are available before model training.
# =====

EXPECTED_CLASSES = [
    '0','1','2','3','4','5','6','7','8','9',
    'a','b','c','d','e','f','g','h','i','j',
    'k','l','m','n','o','p','q','r','s','t',
    'u','v','w','x','y','z'
]

available_classes = sorted([
    folder_name
    for folder_name in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, folder_name))
])

missing_classes = sorted(
    set(EXPECTED_CLASSES) - set(available_classes)
)

unexpected_classes = sorted(
    set(available_classes) - set(EXPECTED_CLASSES)
)

dataset_ready = (
    len(missing_classes) == 0
    and len(unexpected_classes) == 0
)

print("=" * 60)
print("Dataset Class Verification")
print("=" * 60)
print(f"Dataset Status      : {'PASSED' if dataset_ready else 'FAILED'}")
print(f"Expected Classes     : {len(EXPECTED_CLASSES)}")
print(f"Available Classes     : {len(available_classes)}")
print(f"Missing Classes       : {missing_classes}")
print(f"Unexpected Classes    : {unexpected_classes}")
print("=" * 60)

if not dataset_ready:
    raise ValueError("Dataset class verification failed.")

=====
Dataset Class Verification
=====
Dataset Status      : PASSED
Expected Classes     : 36
Available Classes     : 36
Missing Classes       : []
Unexpected Classes    : []
=====
```

In [7]:

```
# =====
# Step 7: Create Final Dataset Split
# This step creates one fixed Train, Validation, and Test split
# that will be used by all models for a fair comparison.
# =====

SPLIT_DATASET_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "01_Dataset/Dataset_Split_Final"
)

required_split_folders = ["train", "val", "test"]

split_is_complete = all(
    os.path.isdir(os.path.join(SPLIT_DATASET_DIR, folder_name))
    for folder_name in required_split_folders
)

if not split_is_complete:

    # Remove the empty or incomplete split folder.
    if os.path.exists(SPLIT_DATASET_DIR):
        shutil.rmtree(SPLIT_DATASET_DIR)

    splitfolders.ratio(
        DATASET_DIR,
        output=SPLIT_DATASET_DIR,
        seed=SEED,
        ratio=(0.70, 0.10, 0.20),
        move=False
    )

    print("=" * 60)
    print("Dataset Split Created Successfully")
    print("=" * 60)

else:
    print("=" * 60)
    print("Complete Dataset Split Already Exists")
    print("=" * 60)

print(f"Split Dataset Path : {SPLIT_DATASET_DIR}")
print("=" * 60)

=====
Complete Dataset Split Already Exists
=====
Split Dataset Path : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final
=====
```

In [8]:

```
# =====
# Step 8: Display Dataset Split Summary
# This step displays the number of images in the Training,
# Validation, and Testing datasets and calculates the total
# number of images.
# =====

VALID_EXTENSIONS = (".jpg", ".jpeg", ".png", ".bmp", ".webp")

split_name_options = {
    "Training": ["train", "training"],
    "Validation": ["val", "valid", "validation"],
    "Testing": ["test", "testing"]
}

resolved_splits = {}

for display_name, possible_names in split_name_options.items():
    for folder_name in possible_names:
        folder_path = os.path.join(SPLIT_DATASET_DIR, folder_name)

        if os.path.isdir(folder_path):
            resolved_splits[display_name] = folder_path
            break

print("=" * 60)
print("Dataset Split Summary")
print("=" * 60)

grand_total = 0

for display_name in ["Training", "Validation", "Testing"]:

    if display_name not in resolved_splits:
        raise FileNotFoundError(
            f"{display_name} folder was not found inside:\n"
            f"{SPLIT_DATASET_DIR}"
        )

    split_path = resolved_splits[display_name]
    total_images = 0

    for class_name in os.listdir(split_path):

        class_path = os.path.join(split_path, class_name)

        if not os.path.isdir(class_path):
            continue

        total_images += sum(
            1
            for file_name in os.listdir(class_path)
            if file_name.lower().endswith(VALID_EXTENSIONS)
        )

    grand_total += total_images
    print(f"{display_name:<12}: {total_images} images")

print("-" * 60)
print(f"Total Images : {grand_total}")
print("=" * 60)

TRAIN_DIR = resolved_splits["Training"]
VALIDATION_DIR = resolved_splits["Validation"]
TEST_DIR = resolved_splits["Testing"]

print("Dataset split paths configured successfully.")
print("=" * 60)
```

```
=====
Dataset Split Summary
=====
Training   : 1760 images
Validation : 251 images
Testing    : 504 images
-----
Total Images : 2515
=====
Dataset split paths configured successfully.
=====
```

In [9]:

```
# =====
# Step 9: Create Image Data Generators
# This step applies MobileNetV2-specific preprocessing and
# data augmentation to prepare the training, validation,
# and testing datasets.
# =====

IMAGE_SIZE = (224, 224)
BATCH_SIZE = 16

train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=20,
    width_shift_range=0.15,
    height_shift_range=0.15,
    zoom_range=0.15,
    shear_range=0.10,
    brightness_range=[0.7, 1.3],
    horizontal_flip=False,
    fill_mode="nearest"
)

validation_test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input
)

train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=True,
    seed=SEED
)

validation_generator = validation_test_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

test_generator = validation_test_datagen.flow_from_directory(
    TEST_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

print("=" * 60)
print("Image Data Generators Created Successfully")
print("=" * 60)
print(f"Image Size      : {IMAGE_SIZE}")
print(f"Batch Size       : {BATCH_SIZE}")
print(f"Number of Classes : {train_generator.num_classes}")
print("=" * 60)
```

```
Found 1760 images belonging to 36 classes.
Found 251 images belonging to 36 classes.
Found 504 images belonging to 36 classes.
=====
Image Data Generators Created Successfully
=====
Image Size      : (224, 224)
Batch Size      : 16
Number of Classes : 36
=====
```

In [10]:

```
# =====
# Step 10: Display Dataset Information
# This step displays information about the dataset generators
# that will be used for model training and evaluation.
# =====

CLASS_NAMES = list(train_generator.class_indices.keys())
NUM_CLASSES = train_generator.num_classes

print("=" * 60)
print("Dataset Information")
print("=" * 60)
print(f"Training Samples : {train_generator.samples}")
print(f"Validation Samples : {validation_generator.samples}")
print(f"Testing Samples : {test_generator.samples}")
print(f"Number of Classes : {NUM_CLASSES}")
print(f"Batch Size : {BATCH_SIZE}")
print(f"Image Size : {IMAGE_SIZE}")
print("=" * 60)
print("Class Names")
print("=" * 60)

for class_name in CLASS_NAMES:
    print(class_name)

print("=" * 60)
```

```
=====
Dataset Information
=====
Training Samples : 1760
Validation Samples : 251
Testing Samples : 504
Number of Classes : 36
Batch Size : 16
Image Size : (224, 224)
=====
Class Names
=====
0
1
2
3
4
5
6
7
8
9
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
=====
```

In [11]:

```
# =====
# Step 11: Build the MobileNetV2 Model
# This step loads the pretrained MobileNetV2 model and replaces
# its original classification head with a new 36-class head.
# =====

base_model = MobileNetV2(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)

base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation="relu")(x)
x = Dropout(0.5)(x)

output = Dense(
    NUM_CLASSES,
    activation="softmax"
)(x)

model = Model(
    inputs=base_model.input,
    outputs=output,
    name="MobileNetV2_ASL_Model"
)

print("=" * 60)
print("MobileNetV2 Model Created Successfully")
print("=" * 60)
print(f"Base Model Trainable : {base_model.trainable}")
print(f"Number of Classes : {NUM_CLASSES}")
print("=" * 60)
```

```
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
[1m9486464/9406464][0m [32m-----[0m[37m[0m [1m0s[0m 0us/step
=====
MobileNetV2 Model Created Successfully
=====
Base Model Trainable : False
Number of Classes : 36
=====
```

In [12]:

```
# =====
# Step 12: Compile the MobileNetV2 Model
# This step configures the model for training by defining the
# optimizer, loss function, and evaluation metric.
# =====

from tensorflow.keras.optimizers import Adam

LEARNING_RATE = 0.0001

model.compile(
    optimizer=Adam(learning_rate=LEARNING_RATE),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

print("=" * 60)
print("Model Compilation Completed Successfully")
print("=" * 60)
print(f"Optimizer : Adam")
print(f"Learning Rate : {LEARNING_RATE}")
print(f"Loss Function : Categorical Crossentropy")
print(f"Evaluation : Accuracy")
print("=" * 60)
```

```
=====
Model Compilation Completed Successfully
=====
Optimizer : Adam
Learning Rate : 0.0001
Loss Function : Categorical Crossentropy
Evaluation : Accuracy
=====
```

In [13]:

```
# =====
# Step 13: Configure Training Callbacks
# This step configures Early Stopping, Model Checkpoint,
# and Learning Rate Reduction during model training.
# =====

MODEL_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/MobileNetV2"
)

os.makedirs(MODEL_DIR, exist_ok=True)

BEST_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "best_mobilenetv2_asl_model.keras"
)

callbacks = [
    EarlyStopping(
        monitor="val_loss",
        patience=7,
        restore_best_weights=True,
        verbose=1
    ),

    ReduceLROnPlateau(
        monitor="val_loss",
        factor=0.3,
        patience=4,
        min_lr=1e-6,
        verbose=1
    ),

    ModelCheckpoint(
        filepath=BEST_MODEL_PATH,
        monitor="val_accuracy",
        save_best_only=True,
        verbose=1
    )
]

print("=" * 60)
print("Training Callbacks Configured Successfully")
print("=" * 60)
print("Early Stopping Patience : 7")
print("LR Reduction Patience   : 4")
print(f"Best Model Path           : {BEST_MODEL_PATH}")
print("=" * 60)
```

```
=====
Training Callbacks Configured Successfully
=====
Early Stopping Patience : 7
LR Reduction Patience   : 4
Best Model Path         : /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
=====
```

In [14]:

```
# =====
# Step 14: Train the MobileNetV2 Model
# This step trains the ResNet50 Model using the training
# and validation datasets.
# =====

EPOCHS = 30

history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=EPOCHS,
    callbacks=callbacks,
    verbose=1
)

print("=" * 60)
print("Model Training Completed Successfully")
print("=" * 60)
```

```
Epoch 1/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 6s/step - accuracy: 0.0357 - loss: 3.8439
Epoch 1: val_accuracy improved from None to 0.21912, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 1: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m769s[0m 7s/step - accuracy: 0.0619 - loss: 3.5866 - val_accuracy: 0.2191 - val_loss: 3.0541 - learning_rate: 1.0000e-04
Epoch 2/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.1531 - loss: 3.1045
Epoch 2: val_accuracy improved from 0.21912 to 0.39841, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 2: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 245ms/step - accuracy: 0.1773 - loss: 3.0212 - val_accuracy: 0.3984 - val_loss: 2.5413 - learning_rate: 1.0000e-04
Epoch 3/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 230ms/step - accuracy: 0.2606 - loss: 2.6775
Epoch 3: val_accuracy improved from 0.39841 to 0.52191, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 3: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 242ms/step - accuracy: 0.2773 - loss: 2.5820 - val_accuracy: 0.5219 - val_loss: 2.1127 - learning_rate: 1.0000e-04
Epoch 4/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.3500 - loss: 2.3386
Epoch 4: val_accuracy improved from 0.52191 to 0.61753, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 4: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 247ms/step - accuracy: 0.3483 - loss: 2.2855 - val_accuracy: 0.6175 - val_loss: 1.8102 - learning_rate: 1.0000e-04
Epoch 5/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 230ms/step - accuracy: 0.3943 - loss: 2.0957
Epoch 5: val_accuracy improved from 0.61753 to 0.63745, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 5: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 242ms/step - accuracy: 0.4091 - loss: 2.0312 - val_accuracy: 0.6375 - val_loss: 1.5975 - learning_rate: 1.0000e-04
Epoch 6/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 228ms/step - accuracy: 0.4397 - loss: 1.9141
Epoch 6: val_accuracy improved from 0.63745 to 0.69721, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 6: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m26s[0m 240ms/step - accuracy: 0.4597 - loss: 1.8509 - val_accuracy: 0.6972 - val_loss: 1.4135 - learning_rate: 1.0000e-04
Epoch 7/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 229ms/step - accuracy: 0.4903 - loss: 1.7122
Epoch 7: val_accuracy improved from 0.69721 to 0.70916, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 7: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 241ms/step - accuracy: 0.4903 - loss: 1.7136 - val_accuracy: 0.7092 - val_loss: 1.3146 - learning_rate: 1.0000e-04
Epoch 8/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 229ms/step - accuracy: 0.5340 - loss: 1.5747
Epoch 8: val_accuracy improved from 0.70916 to 0.72908, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 8: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 242ms/step - accuracy: 0.5409 - loss: 1.5523 - val_accuracy: 0.7291 - val_loss: 1.1670 - learning_rate: 1.0000e-04
Epoch 9/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 234ms/step - accuracy: 0.5818 - loss: 1.4533
Epoch 9: val_accuracy did not improve from 0.72908
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 244ms/step - accuracy: 0.5699 - loss: 1.4568 - val_accuracy: 0.7291 - val_loss: 1.0751 - learning_rate: 1.0000e-04
Epoch 10/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 235ms/step - accuracy: 0.6049 - loss: 1.3708
Epoch 10: val_accuracy improved from 0.72908 to 0.73307, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 10: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 248ms/step - accuracy: 0.6006 - loss: 1.3511 - val_accuracy: 0.7331 - val_loss: 1.0070 - learning_rate: 1.0000e-04
Epoch 11/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 242ms/step - accuracy: 0.6012 - loss: 1.3355
Epoch 11: val_accuracy improved from 0.73307 to 0.75697, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 11: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m28s[0m 255ms/step - accuracy: 0.6011 - loss: 1.3013 - val_accuracy: 0.7570 - val_loss: 0.9374 - learning_rate: 1.0000e-04
Epoch 12/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 237ms/step - accuracy: 0.6349 - loss: 1.2057
Epoch 12: val_accuracy did not improve from 0.75697
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 246ms/step - accuracy: 0.6210 - loss: 1.2271 - val_accuracy: 0.7410 - val_loss: 0.9058 - learning_rate: 1.0000e-04
Epoch 13/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 235ms/step - accuracy: 0.6697 - loss: 1.1262
Epoch 13: val_accuracy improved from 0.75697 to 0.78486, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 13: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 249ms/step - accuracy: 0.6517 - loss: 1.1417 - val_accuracy: 0.7849 - val_loss: 0.8455 - learning_rate: 1.0000e-04
Epoch 14/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 234ms/step - accuracy: 0.6944 - loss: 1.0575
Epoch 14: val_accuracy did not improve from 0.78486
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 243ms/step - accuracy: 0.6790 - loss: 1.0803 - val_accuracy: 0.7729 - val_loss: 0.7858 - learning_rate: 1.0000e-04
Epoch 15/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.6706 - loss: 1.0688
Epoch 15: val_accuracy did not improve from 0.78486
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 242ms/step - accuracy: 0.6858 - loss: 1.0493 - val_accuracy: 0.7809 - val_loss: 0.7723 - learning_rate: 1.0000e-04
Epoch 16/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 237ms/step - accuracy: 0.7221 - loss: 0.9580
Epoch 16: val_accuracy improved from 0.78486 to 0.81275, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 16: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 250ms/step - accuracy: 0.6989 - loss: 0.9874 - val_accuracy: 0.8127 - val_loss: 0.7275 - learning_rate: 1.0000e-04
Epoch 17/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 238ms/step - accuracy: 0.7069 - loss: 0.9527
Epoch 17: val_accuracy did not improve from 0.81275
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 246ms/step - accuracy: 0.6960 - loss: 0.9626 - val_accuracy: 0.8048 - val_loss: 0.6925 - learning_rate: 1.0000e-04
Epoch 18/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 231ms/step - accuracy: 0.7022 - loss: 0.9788
Epoch 18: val_accuracy did not improve from 0.81275
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 241ms/step - accuracy: 0.7188 - loss: 0.9180 - val_accuracy: 0.8048 - val_loss: 0.6782 - learning_rate: 1.0000e-04
Epoch 19/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 236ms/step - accuracy: 0.7242 - loss: 0.9180
Epoch 19: val_accuracy did not improve from 0.81275
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 245ms/step - accuracy: 0.7188 - loss: 0.9077 - val_accuracy: 0.8088 - val_loss: 0.6730 - learning_rate: 1.0000e-04
Epoch 20/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 235ms/step - accuracy: 0.7267 - loss: 0.8995
Epoch 20: val_accuracy did not improve from 0.81275
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 244ms/step - accuracy: 0.7239 - loss: 0.8758 - val_accuracy: 0.7849 - val_loss: 0.6663 - learning_rate: 1.0000e-04
Epoch 21/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 235ms/step - accuracy: 0.7540 - loss: 0.8390
Epoch 21: val_accuracy did not improve from 0.81275
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 244ms/step - accuracy: 0.7648 - loss: 0.8139 - val_accuracy: 0.7888 - val_loss: 0.6220 - learning_rate: 1.0000e-04
Epoch 22/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.7754 - loss: 0.7755
Epoch 22: val_accuracy did not improve from 0.81275
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 241ms/step - accuracy: 0.7636 - loss: 0.8072 - val_accuracy: 0.8127 - val_loss: 0.6221 - learning_rate: 1.0000e-04
Epoch 23/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.7485 - loss: 0.8060
Epoch 23: val_accuracy improved from 0.81275 to 0.82072, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 23: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 246ms/step - accuracy: 0.7778 - loss: 0.7585 - val_accuracy: 0.8207 - val_loss: 0.6088 - learning_rate: 1.0000e-04
Epoch 24/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 235ms/step - accuracy: 0.7772 - loss: 0.7410
Epoch 24: val_accuracy improved from 0.82072 to 0.82470, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 24: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 248ms/step - accuracy: 0.7778 - loss: 0.7585 - val_accuracy: 0.8247 - val_loss: 0.5682 - learning_rate: 1.0000e-04
Epoch 25/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 232ms/step - accuracy: 0.7762 - loss: 0.7307
Epoch 25: val_accuracy improved from 0.82470 to 0.82869, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 25: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 245ms/step - accuracy: 0.7750 - loss: 0.7313 - val_accuracy: 0.8287 - val_loss: 0.5539 - learning_rate: 1.0000e-04
Epoch 26/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 232ms/step - accuracy: 0.7954 - loss: 0.6879
Epoch 26: val_accuracy did not improve from 0.82869
[1m10/110][0m @32m-----[0m[37m[0m @1m26s[0m 241ms/step - accuracy: 0.7909 - loss: 0.6929 - val_accuracy: 0.8247 - val_loss: 0.5463 - learning_rate: 1.0000e-04
Epoch 27/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 231ms/step - accuracy: 0.7744 - loss: 0.7057
Epoch 27: val_accuracy improved from 0.82869 to 0.85657, saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras

Epoch 27: finished saving model to /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/best_mobilenetv2_asl_model.keras
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 245ms/step - accuracy: 0.7744 - loss: 0.7128 - val_accuracy: 0.8566 - val_loss: 0.5239 - learning_rate: 1.0000e-04
Epoch 28/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 234ms/step - accuracy: 0.7922 - loss: 0.6692
Epoch 28: val_accuracy did not improve from 0.85657
[1m10/110][0m @32m-----[0m[37m[0m @1m27s[0m 242ms/step - accuracy: 0.7875 - loss: 0.6857 - val_accuracy: 0.8406 - val_loss: 0.5456 - learning_rate: 1.0000e-04
Epoch 29/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 230ms/step - accuracy: 0.7844 - loss: 0.7204
Epoch 29: val_accuracy did not improve from 0.85657
[1m10/110][0m @32m-----[0m[37m[0m @1m26s[0m 239ms/step - accuracy: 0.7989 - loss: 0.6645 - val_accuracy: 0.8526 - val_loss: 0.5107 - learning_rate: 1.0000e-04
Epoch 30/30
[1m10/110][0m @32m-----[0m[37m[0m @1m0s[0m 233ms/step - accuracy: 0.8139 - loss: 0.6237
```

```
Epoch 30: val_accuracy did not improve from 0.85657
[1m10/110][0m  [32m-----[0m[37m[0m  [1m27s[0m  242ms/step - accuracy: 0.8080 - loss: 0.6377 - val_accuracy: 0.8446 - val_loss: 0.4992 - learning_rate: 1.0000e-04
Restoring model weights from the end of the best epoch: 30.
=====
Model Training Completed Successfully
=====
```

```
In [15]: # =====
# Step 15: Evaluate the MobileNetV2 Model
# This step evaluates the trained model using the testing
# dataset and reports the final loss and accuracy.
# =====

test_loss, test_accuracy = model.evaluate(
    test_generator,
    verbose=1
)

print("=" * 60)
print("Model Evaluation")
print("=" * 60)
print(f"Test Loss      : {test_loss:.4f}")
print(f"Test Accuracy : {test_accuracy:.4f}")
print("=" * 60)
```

```
[1m32/32][0m  [32m-----[0m[37m[0m  [1m180s[0m  6s/step - accuracy: 0.9187 - loss: 0.3565
=====
Model Evaluation
=====
Test Loss      : 0.3565
Test Accuracy : 0.9187
=====
```

```
In [16]: # =====
# Step 16: Generate Predictions and Classification Report
# This step generates predictions on the testing dataset and
# displays the classification report.
# =====

from sklearn.metrics import classification_report
import numpy as np

# Reset the test generator
test_generator.reset()

# Generate predictions
predictions = model.predict(test_generator, verbose=1)

# Predicted and true labels
y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes

# Classification Report
report = classification_report(
    y_true,
    y_pred,
    target_names=list(test_generator.class_indices.keys()),
    digits=4
)

print("=" * 60)
print("Classification Report")
print("=" * 60)
print(report)
print("=" * 60)
```

```
[1m32/32][0m  [32m-----[0m[37m[0m  [1m11s[0m  189ms/step
=====
Classification Report
=====
              precision    recall  f1-score   support

    0   0.6667    0.8571    0.7500        14
    1   0.8750    1.0000    0.9333        14
    2   1.0000    0.8571    0.9231        14
    3   1.0000    1.0000    1.0000        14
    4   0.8571    0.8571    0.8571        14
    5   0.8000    0.8571    0.8276        14
    6   0.7857    0.7857    0.7857        14
    7   0.8235    1.0000    0.9032        14
    8   1.0000    0.8571    0.9231        14
    9   1.0000    0.9286    0.9630        14
    a   0.8667    0.9286    0.8966        14
    b   1.0000    1.0000    1.0000        14
    c   1.0000    1.0000    1.0000        14
    d   1.0000    1.0000    1.0000        14
    e   1.0000    1.0000    1.0000        14
    f   1.0000    1.0000    1.0000        14
    g   1.0000    1.0000    1.0000        14
    h   1.0000    1.0000    1.0000        14
    i   1.0000    1.0000    1.0000        14
    j   1.0000    1.0000    1.0000        14
    k   0.9333    1.0000    0.9655        14
    l   1.0000    1.0000    1.0000        14
    m   0.8000    0.8571    0.8276        14
    n   1.0000    0.7857    0.8800        14
    o   0.8000    0.5714    0.6667        14
    p   1.0000    1.0000    1.0000        14
    q   1.0000    1.0000    1.0000        14
    r   1.0000    0.8571    0.9231        14
    s   1.0000    0.6429    0.7826        14
    t   0.6190    0.9286    0.7429        14
    u   1.0000    1.0000    1.0000        14
    v   0.9167    0.7857    0.8462        14
    w   0.6875    0.7857    0.7333        14
    x   1.0000    1.0000    1.0000        14
    y   1.0000    0.9286    0.9630        14
    z   1.0000    1.0000    1.0000        14

 accuracy          0.9187         504
 macro avg         0.9286    0.9187    0.9193         504
weighted avg         0.9286    0.9187    0.9193         504

=====
```

In [17]:

```
# =====
# Step 17: Create and Save Confusion Matrix
# This step creates a publication-quality confusion matrix
# using a blue color scale and saves it at 600 DPI.
# =====

CONFUSION_MATRIX_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Confusion_Matrices/"
    "mobilenetv2_confusion_matrix.png"
)

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(16, 14))
plt.imshow(cm, cmap="Blues")

plt.title(
    "MobileNetV2 Confusion Matrix",
    fontsize=18
)

plt.xlabel("Predicted Label", fontsize=14)
plt.ylabel("True Label", fontsize=14)

plt.xticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    rotation=90,
    fontsize=10
)

plt.yticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    fontsize=10
)

plt.colorbar()

threshold = cm.max() / 2

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):

        text_color = (
            "white"
            if cm[i, j] > threshold
            else "#0B3D91"
        )

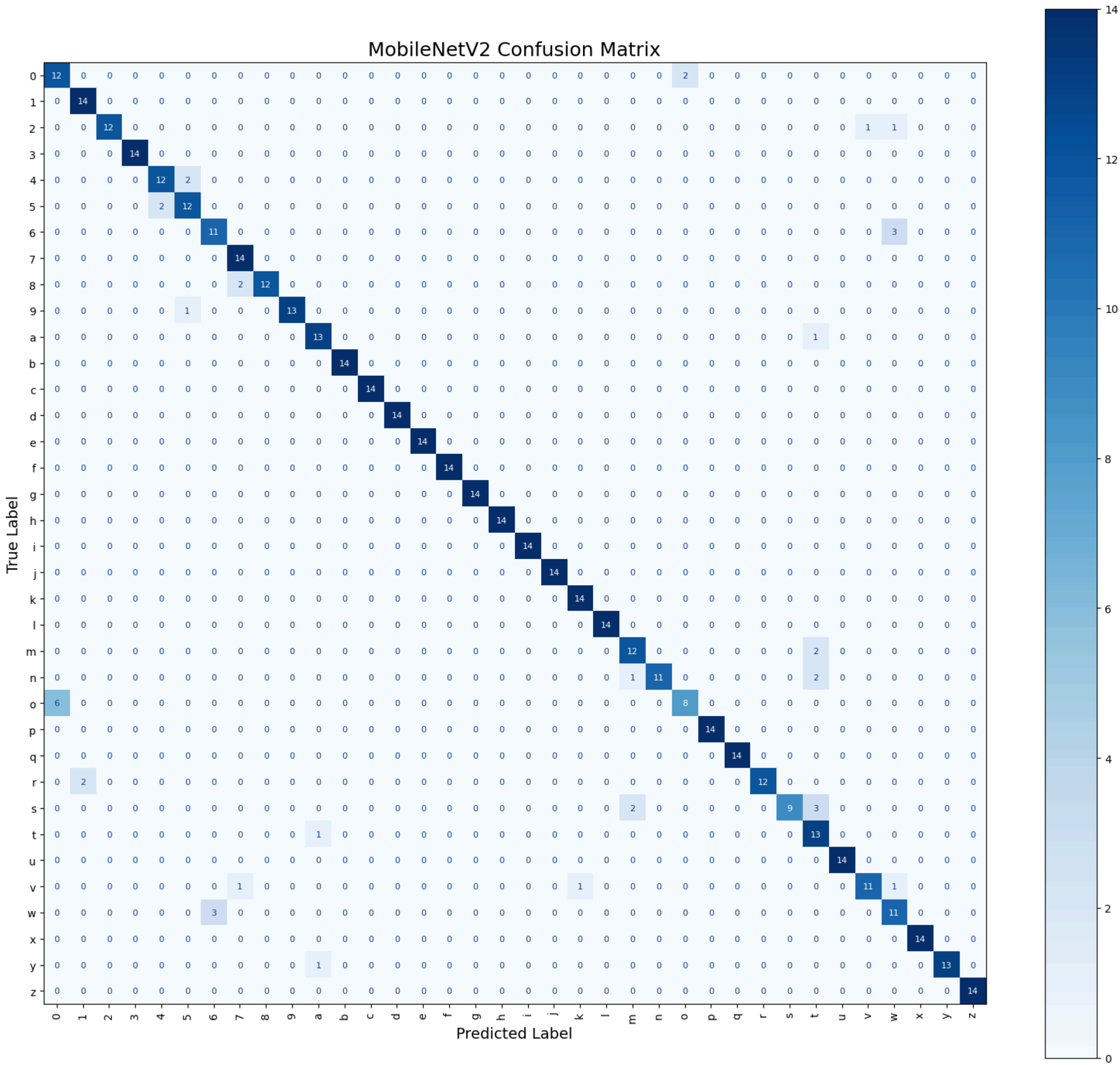
        plt.text(
            j,
            i,
            str(cm[i, j]),
            ha="center",
            va="center",
            fontsize=8,
            color=text_color
        )

plt.tight_layout()

plt.savefig(
    CONFUSION_MATRIX_PATH,
    dpi=600,
    bbox_inches="tight",
    facecolor="white"
)

plt.show()

print("=" * 60)
print("Confusion Matrix Saved Successfully")
print("=" * 60)
print(f"Saved Path : {CONFUSION_MATRIX_PATH}")
print("=" * 60)
```



```
=====
Confusion Matrix Saved Successfully
=====
Saved Path : /content/drive/MyDrive/ASL_Final_Project/03_Results/Confusion_Matrices/mobilenetv2_confusion_matrix.png
=====
```

In [18]:

```
# =====
# Step 18: Plot Training History
# This step plots the training and validation accuracy and
# loss curves and saves them as high-quality images.
# =====

ACCURACY_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Accuracy_Curves/"
    "mobilenetv2_accuracy_curve.png"
)

LOSS_CURVE_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Loss_Curves/"
    "mobilenetv2_loss_curve.png"
)

# Accuracy Curve
plt.figure(figsize=(8, 6))

plt.plot(
    history.history["accuracy"],
    label="Training Accuracy"
)

plt.plot(
    history.history["val_accuracy"],
    label="Validation Accuracy"
)

plt.title("MobileNetV2 Accuracy Curve")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)

plt.savefig(
    ACCURACY_CURVE_PATH,
    dpi=600,
    bbox_inches="tight",
    facecolor="white"
)

plt.show()

# Loss Curve
plt.figure(figsize=(8, 6))

plt.plot(
    history.history["loss"],
    label="Training Loss"
)

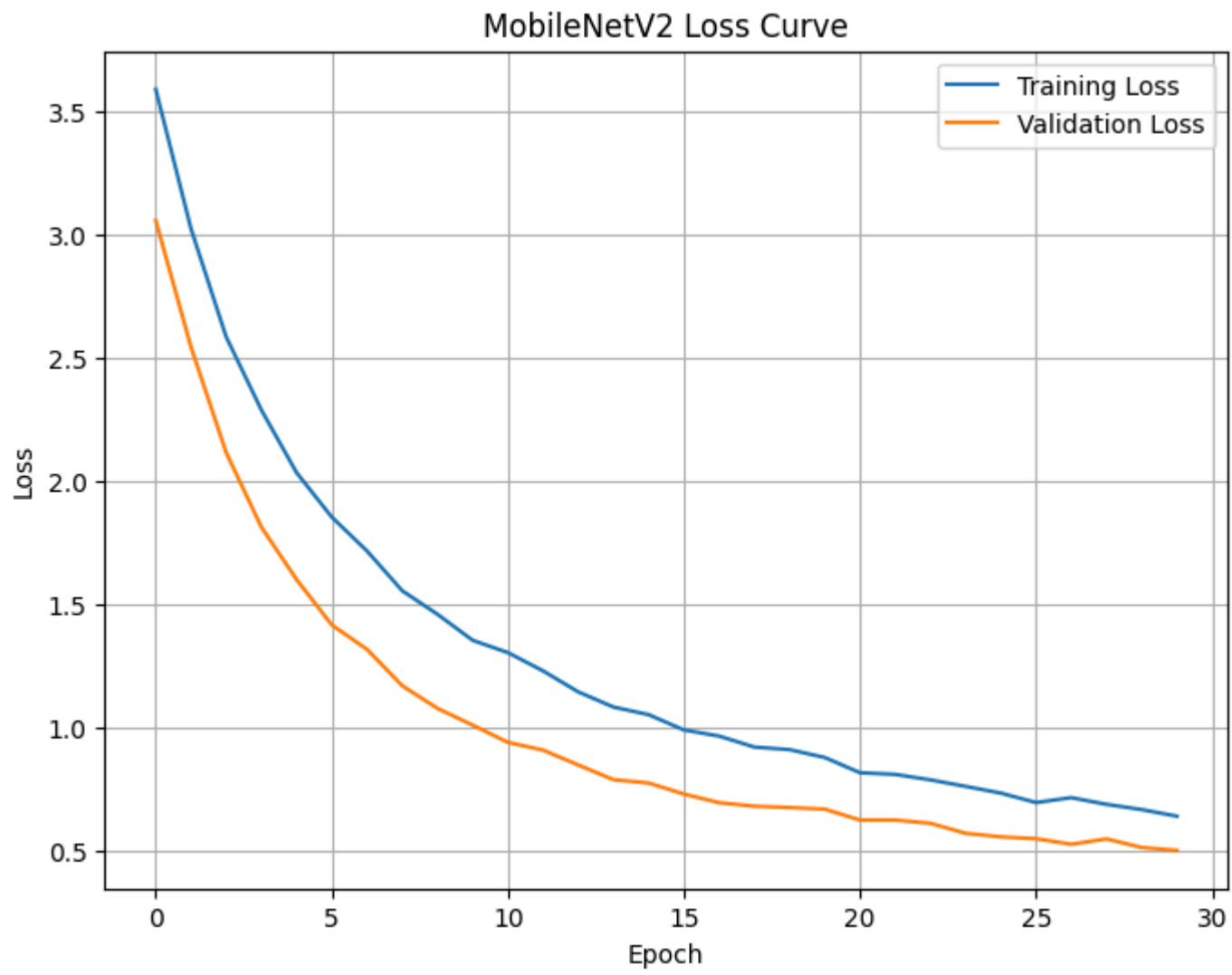
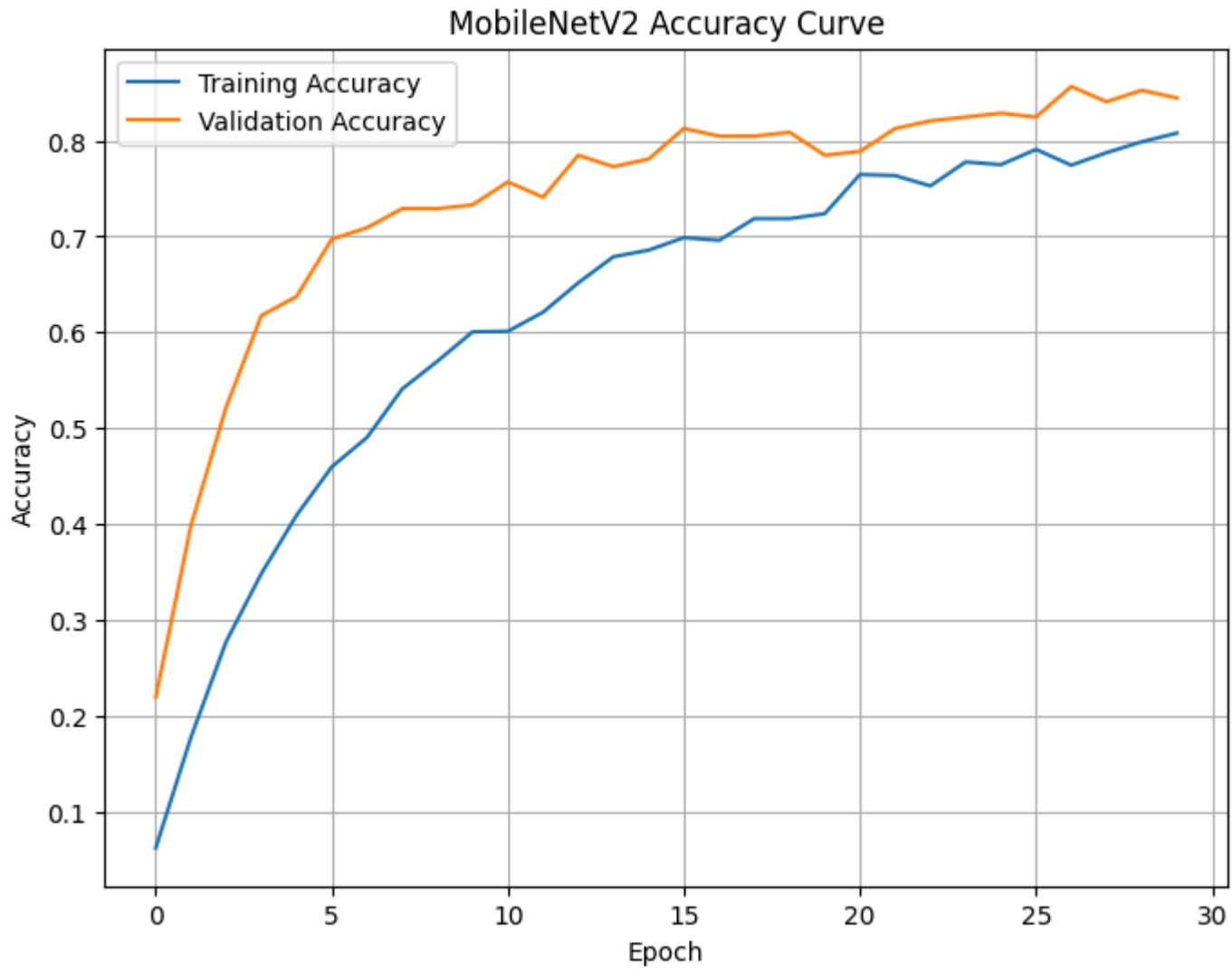
plt.plot(
    history.history["val_loss"],
    label="Validation Loss"
)

plt.title("MobileNetV2 Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

plt.savefig(
    LOSS_CURVE_PATH,
    dpi=600,
    bbox_inches="tight",
    facecolor="white"
)

plt.show()

print("=" * 60)
print("Training Curves Saved Successfully")
print("=" * 60)
print(f"Accuracy Curve : {ACCURACY_CURVE_PATH}")
print(f"Loss Curve      : {LOSS_CURVE_PATH}")
print("=" * 60)
```



```
=====
Training Curves Saved Successfully
=====
Accuracy Curve : /content/drive/MyDrive/ASL_Final_Project/03_Results/Accuracy_Curves/mobilenetv2_accuracy_curve.png
Loss Curve      : /content/drive/MyDrive/ASL_Final_Project/03_Results/Loss_Curves/mobilenetv2_loss_curve.png
=====
```

In [19]:

```
# =====
# Step 19: Save the Final MobileNetV2 Model
# This step saves the trained MobileNetV2 model and the class
# names for future inference and deployment.
# =====

MODEL_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/MobileNetV2"
)

os.makedirs(MODEL_DIR, exist_ok=True)

FINAL_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "final_mobilenetv2_asl_model.keras"
)

CLASS_NAMES_PATH = os.path.join(
    MODEL_DIR,
    "mobilenetv2_class_names.txt"
)

model.save(FINAL_MODEL_PATH)

with open(CLASS_NAMES_PATH, "w") as file:
    for class_name in CLASS_NAMES:
        file.write(f"{class_name}\n")

print("=" * 60)
print("Model Saved Successfully")
print("=" * 60)
print(f"Final Model : {FINAL_MODEL_PATH}")
print(f"Class Names : {CLASS_NAMES_PATH}")
print("=" * 60)

=====
Model Saved Successfully
=====
Final Model : /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/final_mobilenetv2_asl_model.keras
Class Names : /content/drive/MyDrive/ASL_Final_Project/02_Models/MobileNetV2/mobilenetv2_class_names.txt
=====
```

In [20]:

```
# =====
# Step 20: Measure Standardized Inference Time
# This step measures the model inference time on the complete
# test dataset using a warm-up run and repeated measurements.
# =====

INFERENCE_REPEATS = 3

# Warm-up run
test_generator.reset()
_ = model.predict(
    test_generator,
    verbose=0
)

inference_times = []

for repeat in range(INFERENCE_REPEATS):

    test_generator.reset()

    start_time = time.perf_counter()

    _ = model.predict(
        test_generator,
        verbose=0
    )

    end_time = time.perf_counter()

    inference_times.append(end_time - start_time)

average_inference_time = np.mean(inference_times)
std_inference_time = np.std(inference_times)

average_time_per_image = (
    average_inference_time / test_generator.samples
)

print("=" * 60)
print("Standardized Inference Time")
print("=" * 60)
print(f"Test Images           : {test_generator.samples}")
print(f"Repeated Measurements   : {INFERENCE_REPEATS}")
print(f"Average Total Time      : {average_inference_time:.4f} seconds")
print(f"Standard Deviation      : {std_inference_time:.4f} seconds")
print(f"Average Time per Image  : {average_time_per_image:.6f} seconds")
print("=" * 60)

=====
Standardized Inference Time
=====
Test Images           : 504
Repeated Measurements   : 3
Average Total Time      : 1.8144 seconds
Standard Deviation      : 0.0810 seconds
Average Time per Image  : 0.003600 seconds
=====
```

In [21]:

```
# =====
# Step 21: Save Final MobileNetV2 Metrics
# This step calculates and saves the final evaluation,
# inference, parameter, and model size results in JSON format.
# =====

import os
import json

from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score
)

MODEL_KEY = "mobilenetv2"
MODEL_DISPLAY_NAME = "MobileNetV2"

FINAL_SAVED_MODEL_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/MobileNetV2/final_mobilenetv2_asl_model.keras"
)

METRICS_DIR = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "03_Results/Metrics"
)

os.makedirs(METRICS_DIR, exist_ok=True)

METRICS_PATH = os.path.join(
    METRICS_DIR,
    "mobilenetv2_metrics.json"
)

weighted_precision = precision_score(
    y_true,
    y_pred,
    average="weighted",
    zero_division=0
)

weighted_recall = recall_score(
    y_true,
    y_pred,
    average="weighted",
    zero_division=0
)

weighted_f1 = f1_score(
    y_true,
    y_pred,
    average="weighted",
    zero_division=0
)

macro_f1 = f1_score(
    y_true,
    y_pred,
    average="macro",
    zero_division=0
)

model_size_mb = (
    os.path.getsize(FINAL_SAVED_MODEL_PATH)
    / (1024 ** 2)
)

gpus = tf.config.list_physical_devices("GPU")

if gpus:
    gpu_name = (
        tf.config.experimental
        .get_device_details(gpus[0])
        .get("device_name", gpus[0].name)
    )
else:
    gpu_name = "CPU"

metrics_data = {
    "model": MODEL_DISPLAY_NAME,
    "framework": "TensorFlow/Keras",
    "gpu": gpu_name,
    "test_images": int(test_generator.samples),
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "weighted_precision": float(weighted_precision),
    "weighted_recall": float(weighted_recall),
    "weighted_f1_score": float(weighted_f1),
    "macro_f1_score": float(macro_f1),
    "average_inference_time_seconds": float(
        average_inference_time
    ),
    "inference_standard_deviation_seconds": float(
        std_inference_time
    ),
    "average_time_per_image_seconds": float(
        average_time_per_image
    ),
    "parameters": int(model.count_params()),
    "model_size_mb": float(model_size_mb),
    "model_path": FINAL_SAVED_MODEL_PATH
}

with open(METRICS_PATH, "w") as metrics_file:
    json.dump(
        metrics_data,
        metrics_file,
        indent=4
    )

print("=" * 60)
print("MobileNetV2 Metrics Saved Successfully")
print("=" * 60)
print(f"Accuracy      : {test_accuracy:.4f}")
print(f"Precision      : {weighted_precision:.4f}")
print(f"Recall         : {weighted_recall:.4f}")
print(f"F1-Score       : {weighted_f1:.4f}")
print(f"Model Size     : {model_size_mb:.2f} MB")
print(f"Parameters     : {model.count_params():,}")
print(f"Metrics File   : {METRICS_PATH}")
print("=" * 60)

=====
MobileNetV2 Metrics Saved Successfully
=====
Accuracy      : 0.9187
Precision     : 0.9286
Recall        : 0.9187
F1-Score      : 0.9193
Model Size    : 13.03 MB
Parameters    : 2,595,172
Metrics File   : /content/drive/MyDrive/ASL_Final_Project/03_Results/Metrics/mobilenetv2_metrics.json
=====
```