

```
In [1]: # Step 1: Mount Google Drive
# This step connects Google Drive to access the dataset and save trained models.

from google.colab import drive

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

Mounted at /content/drive

```
In [2]: # Step 2: Install Required Libraries
# This step installs timm for the Swin Transformer model and Gradio for the web interface.

!pip install timm gradio -q
```

```
In [3]: # Step 3: Import Required Libraries
# This step imports all required libraries for data processing, training, evaluation, visualization, and the web interface.

import os
import shutil
import random
import numpy as np
import matplotlib.pyplot as plt

import torch
import timm
import gradio as gr

from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

from PIL import Image

from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    accuracy_score
)
```

```
In [4]: # Step 4: Set Device and Random Seed
# This step uses GPU if available and fixes randomness for more stable results.

SEED = 42

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

print("Device:", device)
```

Device: cuda

```
In [5]: # Step 5: Define Dataset Path
# This step searches for the correct dataset folder inside Google Drive.

possible_paths = [
    "/content/drive/MyDrive/ASL_Project/ASL_Dataset/asl_dataset",
]

ORIGINAL_DATASET_DIR = None

for path in possible_paths:
    if os.path.isdir(path):
        ORIGINAL_DATASET_DIR = path
        break

if ORIGINAL_DATASET_DIR is None:
    print("Dataset folder not found.")
    print("Folders inside MyDrive:")
    print(os.listdir("/content/drive/MyDrive"))
    raise FileNotFoundError("Dataset path is not correct.")

print("Dataset Path:", ORIGINAL_DATASET_DIR)
print("Folders:", os.listdir(ORIGINAL_DATASET_DIR))

Dataset Path: /content/drive/MyDrive/ASL_Project/ASL_Dataset/asl_dataset
Folders: ['x', '2', 'y', 't', 'v', 'q', 'm', 's', 'p', 'w', 'o', 'l', 'n', 'u', 'r', 'k', 'h', 'i', 'e', 'f', 'b', '7', 'd', 'g', 'c', 'a', '9', '8', '6', 'j', '5', '4', '0', '1', '2', '3']
```

```
In [6]: # Step 6: Define Class Labels
# This step defines the valid letters and digits used for ASL classification.

VALID_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'
]

print("Number of Classes:", len(VALID_CLASSES))
print("Classes:", VALID_CLASSES)

Number of Classes: 36
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']
```

```
In [7]: # Step 7: Create Dataset Split Directories
# This step creates fresh train, validation, and test folders for the dataset split.

SPLIT_DATASET_DIR = "/content/drive/MyDrive/ASL_Split_Dataset_Swin"

train_dir = os.path.join(SPLIT_DATASET_DIR, "train")
validation_dir = os.path.join(SPLIT_DATASET_DIR, "validation")
test_dir = os.path.join(SPLIT_DATASET_DIR, "test")

if os.path.exists(SPLIT_DATASET_DIR):
    shutil.rmtree(SPLIT_DATASET_DIR)

os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

print("Split folders created successfully.")

Split folders created successfully.
```

```
In [8]: # Step 8: Split the Dataset
# This step splits the dataset into training, validation, and testing sets.
```

```
train_ratio = 0.70
validation_ratio = 0.10
test_ratio = 0.20

def find_class_folder(dataset_dir, class_name):
    possible_names = [class_name, class_name.upper(), class_name.lower()]

    for name in possible_names:
        path = os.path.join(dataset_dir, name)
        if os.path.isdir(path):
            return path

    return None

def split_and_copy_files(class_dir, class_name):

    all_images = []

    for root, dirs, files in os.walk(class_dir):
        for file in files:
            if file.lower().endswith((".jpg", ".jpeg", ".png", ".bmp", ".webp")):
                all_images.append(os.path.join(root, file))

    if len(all_images) < 3:
        print(f"Not enough images in class: {class_name}")
        return 0, 0, 0

    train_images, temp_images = train_test_split(
        all_images,
        test_size=0.30,
        random_state=SEED,
        shuffle=True
    )

    validation_images, test_images = train_test_split(
        temp_images,
        test_size=2/3,
        random_state=SEED,
        shuffle=True
    )

    split_data = {
        "train": train_images,
        "validation": validation_images,
        "test": test_images
    }

    for split_name, images in split_data.items():
        target_dir = os.path.join(
            SPLIT_DATASET_DIR,
            split_name,
            class_name
        )

        os.makedirs(target_dir, exist_ok=True)

        for img_path in images:
            shutil.copy2(img_path, target_dir)

    return len(train_images), len(validation_images), len(test_images)
```

```
total_train = 0
total_validation = 0
total_test = 0

for class_name in VALID_CLASSES:

    class_dir = find_class_folder(
        ORIGINAL_DATASET_DIR,
        class_name
    )

    if class_dir is None:
        print("Missing class folder:", class_name)
        continue

    train_count, validation_count, test_count = split_and_copy_files(
        class_dir,
        class_name
    )

    total_train += train_count
    total_validation += validation_count
    total_test += test_count

    print(
        f"{class_name}: "
        f"Train={train_count}, "
        f"Validation={validation_count}, "
        f"Test={test_count}"
    )

print("\nFinal Split:")
print("Training Images:", total_train)
print("Validation Images:", total_validation)
print("Testing Images:", total_test)
```

```
0: Train=49, Validation=7, Test=14
1: Train=49, Validation=7, Test=14
2: Train=49, Validation=7, Test=14
3: Train=49, Validation=7, Test=14
4: Train=49, Validation=7, Test=14
5: Train=49, Validation=7, Test=14
6: Train=49, Validation=7, Test=14
7: Train=49, Validation=7, Test=14
8: Train=49, Validation=7, Test=14
9: Train=49, Validation=7, Test=14
a: Train=49, Validation=7, Test=15
b: Train=49, Validation=7, Test=14
c: Train=49, Validation=7, Test=14
d: Train=49, Validation=7, Test=14
e: Train=49, Validation=7, Test=14
f: Train=49, Validation=7, Test=14
g: Train=49, Validation=7, Test=14
h: Train=49, Validation=7, Test=14
i: Train=49, Validation=7, Test=14
j: Train=49, Validation=7, Test=14
k: Train=49, Validation=7, Test=14
l: Train=49, Validation=7, Test=14
m: Train=49, Validation=7, Test=14
n: Train=49, Validation=7, Test=14
o: Train=49, Validation=7, Test=14
p: Train=49, Validation=7, Test=14
q: Train=49, Validation=7, Test=14
r: Train=49, Validation=7, Test=14
s: Train=49, Validation=7, Test=14
t: Train=45, Validation=6, Test=14
u: Train=49, Validation=7, Test=14
v: Train=49, Validation=7, Test=14
w: Train=49, Validation=7, Test=14
x: Train=49, Validation=7, Test=14
y: Train=49, Validation=7, Test=14
z: Train=49, Validation=7, Test=14

Final Split:
Training Images: 1760
Validation Images: 251
Testing Images: 505
```

```
In [9]: # Step 9: Define Training Parameters and Save Paths
# This step defines image size, batch size, epochs, learning rate, and Google Drive save paths.
```

```
IMG_SIZE = 224
BATCH_SIZE = 16
EPOCHS = 30
LEARNING_RATE = 0.0001

MODEL_DIR = "/content/drive/MyDrive/ASL_Transformer_Project"
os.makedirs(MODEL_DIR, exist_ok=True)

BEST_MODEL_PATH = os.path.join(MODEL_DIR, "best_swin_asl_model.pth")
FINAL_MODEL_PATH = os.path.join(MODEL_DIR, "final_swin_asl_model.pth")
CLASS_NAMES_PATH = os.path.join(MODEL_DIR, "swin_class_names.txt")
CONFUSION_MATRIX_PATH = os.path.join(MODEL_DIR, "swin_confusion_matrix.png")
```

```
print("Image Size:", IMG_SIZE)
print("Batch Size:", BATCH_SIZE)
print("Epochs:", EPOCHS)
print("Learning Rate:", LEARNING_RATE)
print("Model Save Folder:", MODEL_DIR)
```

```
Image Size: 224
Batch Size: 16
Epochs: 30
Learning Rate: 0.0001
Model Save Folder: /content/drive/MyDrive/ASL_Transformer_Project
```

```
In [10]: # Step 10: Create Image Transformations
# This step prepares image transformations for training, validation, and testing.
```

```
train_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomRotation(15),
    transforms.RandomHorizontalFlip(p=0.2),
    transforms.ColorJitter(
        brightness=0.2,
        contrast=0.2
    ),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

```
val_test_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

```
In [11]: # Step 11: Load the Dataset
# This step loads the training, validation, and testing datasets using ImageFolder.
```

```
train_dataset = datasets.ImageFolder(
    root=train_dir,
    transform=train_transform
)
```

```
validation_dataset = datasets.ImageFolder(
    root=validation_dir,
    transform=val_test_transform
)
```

```
test_dataset = datasets.ImageFolder(
    root=test_dir,
    transform=val_test_transform
)
```

```
CLASS_NAMES = train_dataset.classes
NUM_CLASSES = len(CLASS_NAMES)
```

```
print("Class Names:", CLASS_NAMES)
print("Number of Classes:", NUM_CLASSES)
print("Train Images:", len(train_dataset))
print("Validation Images:", len(validation_dataset))
print("Test Images:", len(test_dataset))
```

```
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']
Number of Classes: 36
Train Images: 1760
Validation Images: 251
Test Images: 505
```

```
In [12]: # Step 12: Create Data Loaders
# This step creates DataLoaders for training, validation, and testing.
```

```
train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=2
)
```

```
validation_loader = DataLoader(
    validation_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2
)
```

```
test_loader = DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2
)
```

```
print("Train Batches:", len(train_loader))
print("Validation Batches:", len(validation_loader))
print("Test Batches:", len(test_loader))
```

```
Train Batches: 110
Validation Batches: 16
Test Batches: 32
```

```
In [13]: # Step 13: Build the Swin Transformer Model
# This step loads the pre-trained Swin Transformer model and updates the output layer.
```

```
model = timm.create_model(
    "swin_tiny_patch4_window7_224",
    pretrained=True,
    num_classes=NUM_CLASSES
)
```

```
model = model.to(device)
```

```
print("Swin Transformer model created successfully.")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:124: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

```
model.safetensors:   0% |          | 0.00/114M [00:00<?, ?B/s]
```

```
Swin Transformer model created successfully.
```

```
In [14]: # Step 14: Configure the Training Setup
# This step defines the loss function and optimizer used for training.
```

```
criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.AdamW(
    model.parameters(),
    lr=LEARNING_RATE,
    weight_decay=0.01
)
```

```
In [15]: # Step 15: Create the Training Function
# This step trains the Swin Transformer model for one epoch and calculates the training loss and accuracy.

def train_one_epoch(model, loader, criterion, optimizer, device):

    model.train()

    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in loader:

        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = model(images)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        running_loss += loss.item() * images.size(0)

        _, predicted = torch.max(outputs, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / total
    epoch_accuracy = correct / total

    return epoch_loss, epoch_accuracy
```

```
In [16]: # Step 16: Create the Evaluation Function
# This step evaluates the model on the validation or test dataset without updating the model weights.

def evaluate_model(model, loader, criterion, device):

    model.eval()

    running_loss = 0.0
    correct = 0
    total = 0

    all_labels = []
    all_predictions = []

    with torch.no_grad():

        for images, labels in loader:

            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)

            loss = criterion(outputs, labels)

            running_loss += loss.item() * images.size(0)

            _, predicted = torch.max(outputs, 1)

            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    epoch_loss = running_loss / total
    epoch_accuracy = correct / total

    return epoch_loss, epoch_accuracy, all_labels, all_predictions
```

```
In [17]: # Step 17: Train the Swin Transformer Model
# This step trains the model, monitors validation performance, and saves the best model.

best_val_accuracy = 0.0

train_losses = []
validation_losses = []

train_accuracies = []
validation_accuracies = []

for epoch in range(EPOCHS):

    train_loss, train_accuracy = train_one_epoch(
        model,
        train_loader,
        criterion,
        optimizer,
        device
    )

    validation_loss, validation_accuracy, _, _ = evaluate_model(
        model,
        validation_loader,
        criterion,
        device
    )

    train_losses.append(train_loss)
    validation_losses.append(validation_loss)

    train_accuracies.append(train_accuracy)
    validation_accuracies.append(validation_accuracy)

    print(f"Epoch [{epoch + 1}/{EPOCHS}]]")
    print(f"Train Loss: {train_loss:.4f} | Train Accuracy: {train_accuracy * 100:.2f}%")
    print(f"Validation Loss: {validation_loss:.4f} | Validation Accuracy: {validation_accuracy * 100:.2f}%")

    if validation_accuracy > best_val_accuracy:
        best_val_accuracy = validation_accuracy
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        print("Best model saved.")

print("-" * 60)

print("Training finished.")
```

```
Epoch [1/30]
Train Loss: 1.9006 | Train Accuracy: 47.50%
Validation Loss: 0.4615 | Validation Accuracy: 81.27%
Best model saved.
-----
Epoch [2/30]
Train Loss: 0.3829 | Train Accuracy: 87.44%
Validation Loss: 0.2419 | Validation Accuracy: 91.24%
Best model saved.
-----
Epoch [3/30]
Train Loss: 0.2412 | Train Accuracy: 92.16%
Validation Loss: 0.2720 | Validation Accuracy: 89.24%
-----
Epoch [4/30]
Train Loss: 0.1834 | Train Accuracy: 93.75%
Validation Loss: 0.1431 | Validation Accuracy: 94.82%
Best model saved.
-----
Epoch [5/30]
Train Loss: 0.1357 | Train Accuracy: 95.97%
Validation Loss: 0.1075 | Validation Accuracy: 95.62%
Best model saved.
-----
Epoch [6/30]
Train Loss: 0.1256 | Train Accuracy: 96.02%
Validation Loss: 0.1273 | Validation Accuracy: 94.82%
-----
Epoch [7/30]
Train Loss: 0.1316 | Train Accuracy: 95.51%
Validation Loss: 0.1075 | Validation Accuracy: 94.82%
-----
Epoch [8/30]
Train Loss: 0.0831 | Train Accuracy: 97.50%
Validation Loss: 0.1531 | Validation Accuracy: 93.23%
-----
Epoch [9/30]
Train Loss: 0.0934 | Train Accuracy: 97.05%
Validation Loss: 0.0987 | Validation Accuracy: 97.21%
Best model saved.
-----
Epoch [10/30]
Train Loss: 0.0525 | Train Accuracy: 98.30%
Validation Loss: 0.1191 | Validation Accuracy: 98.01%
Best model saved.
-----
Epoch [11/30]
Train Loss: 0.0415 | Train Accuracy: 98.92%
Validation Loss: 0.0501 | Validation Accuracy: 98.41%
Best model saved.
-----
Epoch [12/30]
Train Loss: 0.0587 | Train Accuracy: 98.35%
Validation Loss: 0.0475 | Validation Accuracy: 98.41%
-----
Epoch [13/30]
Train Loss: 0.1113 | Train Accuracy: 97.16%
Validation Loss: 0.0557 | Validation Accuracy: 97.21%
-----
Epoch [14/30]
Train Loss: 0.0583 | Train Accuracy: 97.90%
Validation Loss: 0.0553 | Validation Accuracy: 98.01%
-----
Epoch [15/30]
Train Loss: 0.0529 | Train Accuracy: 98.52%
Validation Loss: 0.0754 | Validation Accuracy: 97.61%
-----
Epoch [16/30]
Train Loss: 0.0613 | Train Accuracy: 98.12%
Validation Loss: 0.1520 | Validation Accuracy: 94.42%
-----
Epoch [17/30]
Train Loss: 0.0577 | Train Accuracy: 98.58%
Validation Loss: 0.1042 | Validation Accuracy: 96.81%
-----
Epoch [18/30]
Train Loss: 0.0537 | Train Accuracy: 98.64%
Validation Loss: 0.0345 | Validation Accuracy: 99.20%
Best model saved.
-----
Epoch [19/30]
Train Loss: 0.0774 | Train Accuracy: 97.73%
Validation Loss: 0.1265 | Validation Accuracy: 96.41%
-----
Epoch [20/30]
Train Loss: 0.0740 | Train Accuracy: 98.07%
Validation Loss: 0.1244 | Validation Accuracy: 97.21%
-----
Epoch [21/30]
Train Loss: 0.0397 | Train Accuracy: 98.58%
Validation Loss: 0.1329 | Validation Accuracy: 96.81%
-----
Epoch [22/30]
Train Loss: 0.0341 | Train Accuracy: 99.09%
Validation Loss: 0.0754 | Validation Accuracy: 98.41%
-----
Epoch [23/30]
Train Loss: 0.0311 | Train Accuracy: 99.03%
Validation Loss: 0.0275 | Validation Accuracy: 99.60%
Best model saved.
-----
Epoch [24/30]
Train Loss: 0.0295 | Train Accuracy: 98.92%
Validation Loss: 0.0599 | Validation Accuracy: 98.01%
-----
Epoch [25/30]
Train Loss: 0.0409 | Train Accuracy: 98.86%
Validation Loss: 0.0670 | Validation Accuracy: 97.21%
-----
Epoch [26/30]
Train Loss: 0.0529 | Train Accuracy: 98.75%
Validation Loss: 0.1430 | Validation Accuracy: 97.61%
-----
Epoch [27/30]
Train Loss: 0.0277 | Train Accuracy: 99.20%
Validation Loss: 0.1286 | Validation Accuracy: 95.62%
-----
Epoch [28/30]
Train Loss: 0.0614 | Train Accuracy: 98.41%
Validation Loss: 0.1360 | Validation Accuracy: 98.41%
-----
Epoch [29/30]
Train Loss: 0.0310 | Train Accuracy: 99.26%
```

```
Validation Loss: 0.1457 | Validation Accuracy: 98.01%
-----
Epoch [30/30]
Train Loss: 0.0383 | Train Accuracy: 98.86%
Validation Loss: 0.0650 | Validation Accuracy: 98.01%
-----
Training finished.
```

```
In [18]: # Step 18: Load the Best Saved Swin Transformer Model
# This step loads the best model weights saved during training.

model = timm.create_model(
    "swin_tiny_patch4_window7_224",
    pretrained=False,
    num_classes=NUM_CLASSES
)

model.load_state_dict(
    torch.load(BEST_MODEL_PATH, map_location=device)
)

model = model.to(device)
model.eval()

print("Best Swin Transformer model loaded successfully.")

Best Swin Transformer model loaded successfully.
```

```
In [19]: # Step 19: Evaluate the Swin Transformer Model
# This step evaluates the trained model on the test dataset.

test_loss, test_accuracy, true_labels, predicted_labels = evaluate_model(
    model,
    test_loader,
    criterion,
    device
)

print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

Test Loss: 0.0255
Test Accuracy: 99.21%
```

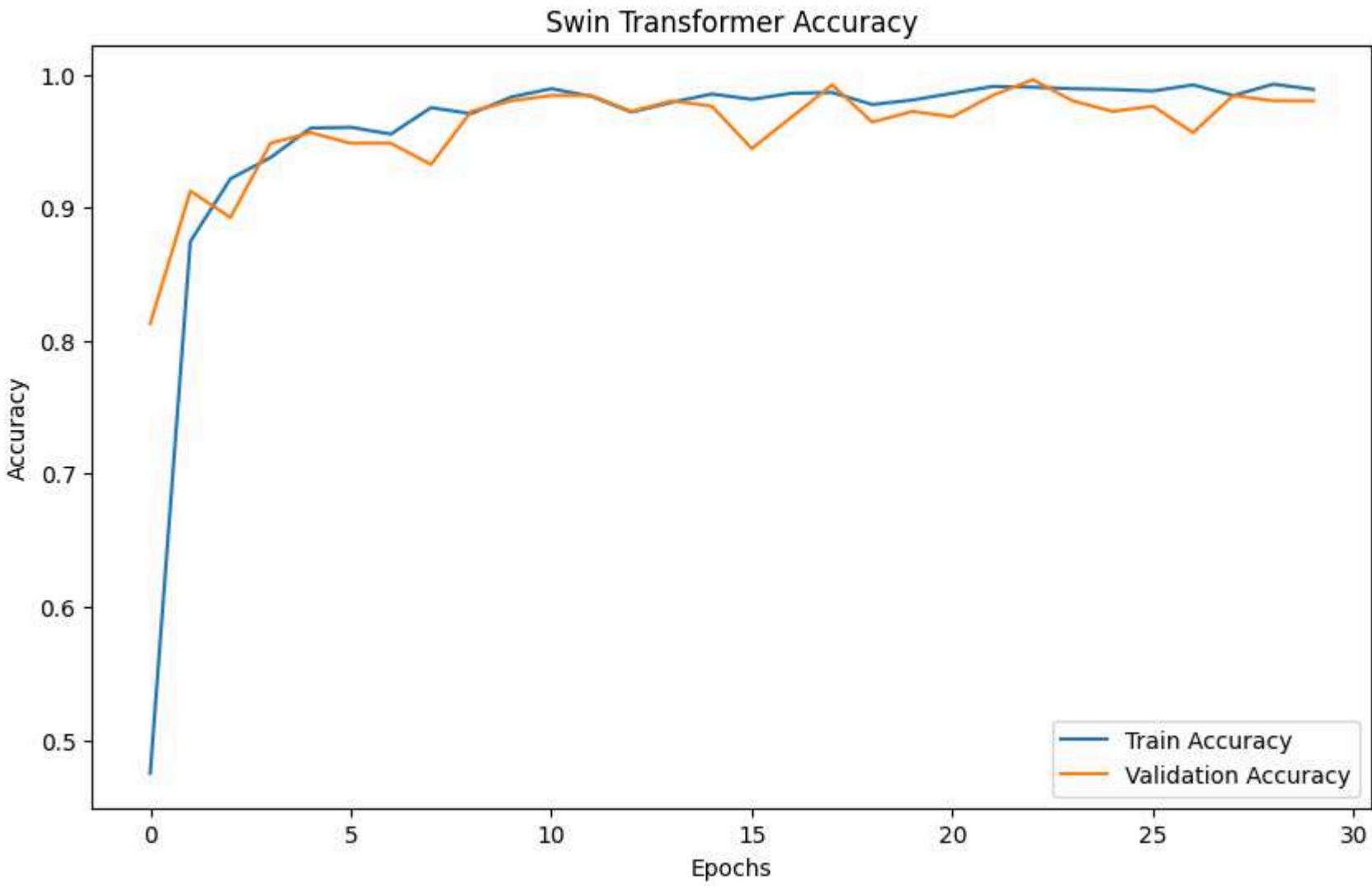
```
In [20]: # Step 20: Plot Accuracy Curves
# This step plots training and validation accuracy across epochs.

plt.figure(figsize=(10, 6))

plt.plot(train_accuracies, label="Train Accuracy")
plt.plot(validation_accuracies, label="Validation Accuracy")

plt.title("Swin Transformer Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")

plt.legend()
plt.show()
```



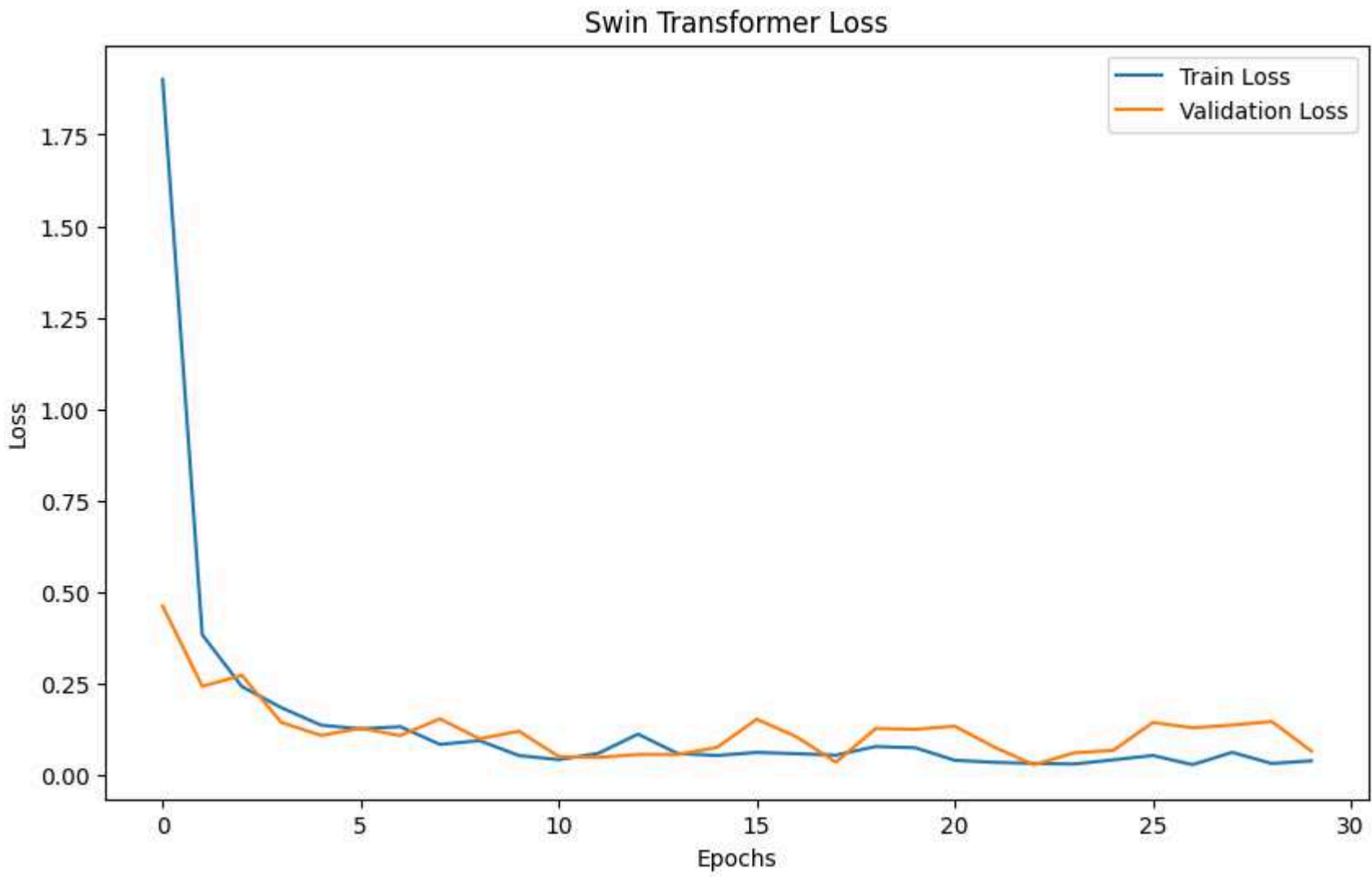
```
In [21]: # Step 21: Plot Loss Curves
# This step plots training and validation loss across epochs.

plt.figure(figsize=(10, 6))

plt.plot(train_losses, label="Train Loss")
plt.plot(validation_losses, label="Validation Loss")

plt.title("Swin Transformer Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")

plt.legend()
plt.show()
```



In [22]: # Step 22: Generate the Classification Report
This step displays the precision, recall, F1-score, and accuracy for each class.

```
print("Swin Transformer Classification Report")  
  
print(  
    classification_report(  
        true_labels,  
        predicted_labels,  
        target_names=CLASS_NAMES,  
        zero_division=0  
    )  
)
```

Swin Transformer Classification Report				
	precision	recall	f1-score	support
0	0.93	1.00	0.97	14
1	1.00	0.93	0.96	14
2	1.00	1.00	1.00	14
3	1.00	1.00	1.00	14
4	1.00	1.00	1.00	14
5	1.00	1.00	1.00	14
6	1.00	0.86	0.92	14
7	1.00	1.00	1.00	14
8	1.00	1.00	1.00	14
9	1.00	1.00	1.00	14
a	1.00	1.00	1.00	15
b	1.00	1.00	1.00	14
c	1.00	1.00	1.00	14
d	1.00	1.00	1.00	14
e	1.00	1.00	1.00	14
f	1.00	1.00	1.00	14
g	1.00	1.00	1.00	14
h	1.00	1.00	1.00	14
i	1.00	1.00	1.00	14
j	1.00	1.00	1.00	14
k	1.00	1.00	1.00	14
l	1.00	1.00	1.00	14
m	1.00	1.00	1.00	14
n	1.00	1.00	1.00	14
o	1.00	0.93	0.96	14
p	1.00	1.00	1.00	14
q	1.00	1.00	1.00	14
r	1.00	1.00	1.00	14
s	1.00	1.00	1.00	14
t	1.00	1.00	1.00	14
u	1.00	1.00	1.00	14
v	1.00	1.00	1.00	14
w	0.88	1.00	0.93	14
x	1.00	1.00	1.00	14
y	1.00	1.00	1.00	14
z	0.93	1.00	0.97	14
accuracy			0.99	505
macro avg	0.99	0.99	0.99	505
weighted avg	0.99	0.99	0.99	505

```
In [23]: # Step 23: Generate the Confusion Matrix
# This step visualizes the classification results using a confusion matrix.

cm = confusion_matrix(true_labels, predicted_labels)

plt.figure(figsize=(18, 16))

plt.imshow(cm)

plt.title("Swin Transformer Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

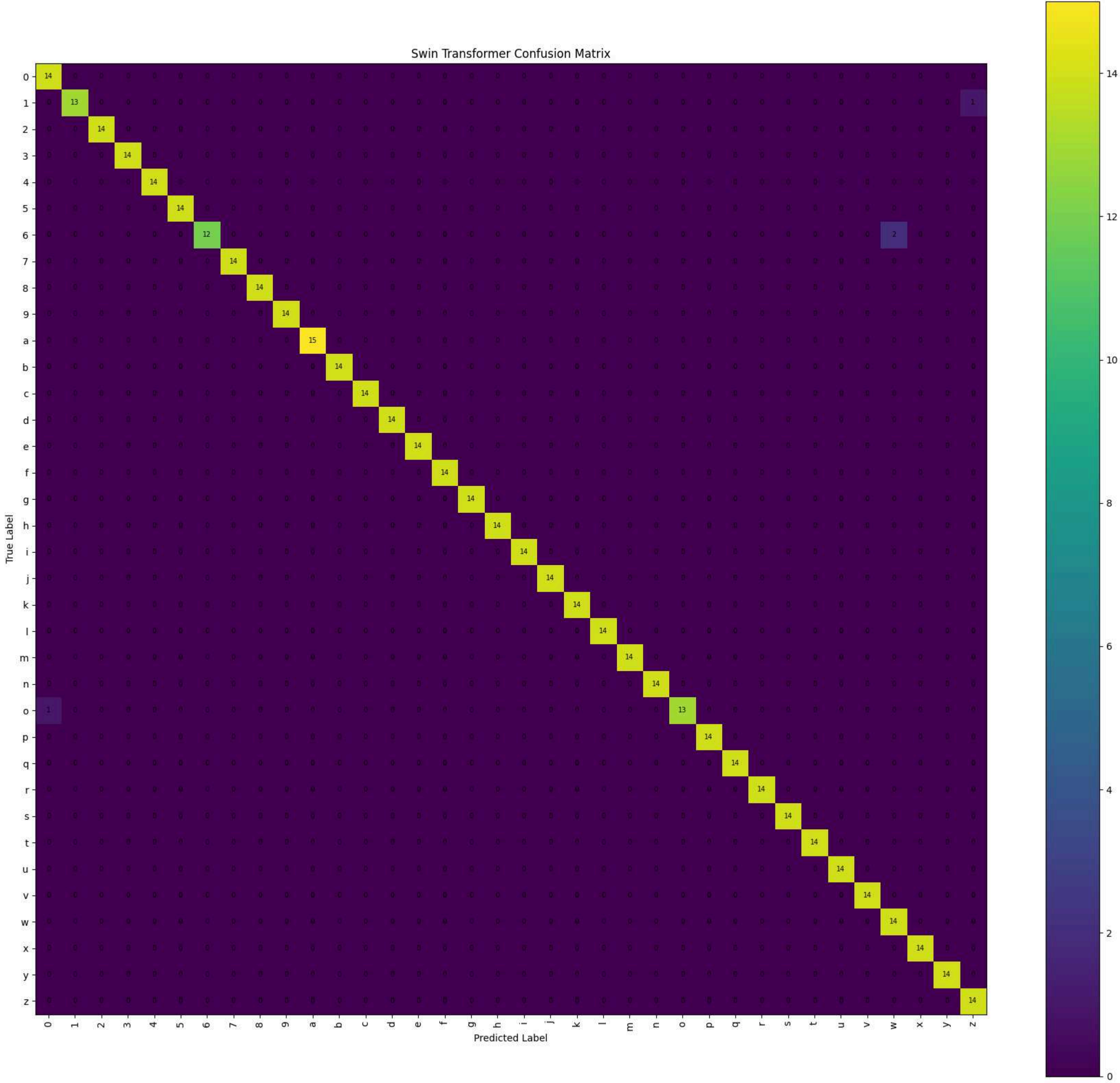
plt.xticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    rotation=90
)

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

plt.colorbar()

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        plt.text(
            j,
            i,
            cm[i, j],
            ha="center",
            va="center",
            fontsize=7
        )

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



```
In [24]: # Step 24: Save the Confusion Matrix
# This step saves the confusion matrix image to Google Drive for reporting.
```

```
CONFUSION_MATRIX_PATH = os.path.join(
    MODEL_DIR,
    "swin_confusion_matrix.png"
)

plt.figure(figsize=(18, 16))

plt.imshow(cm)

plt.title("Swin Transformer Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

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

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

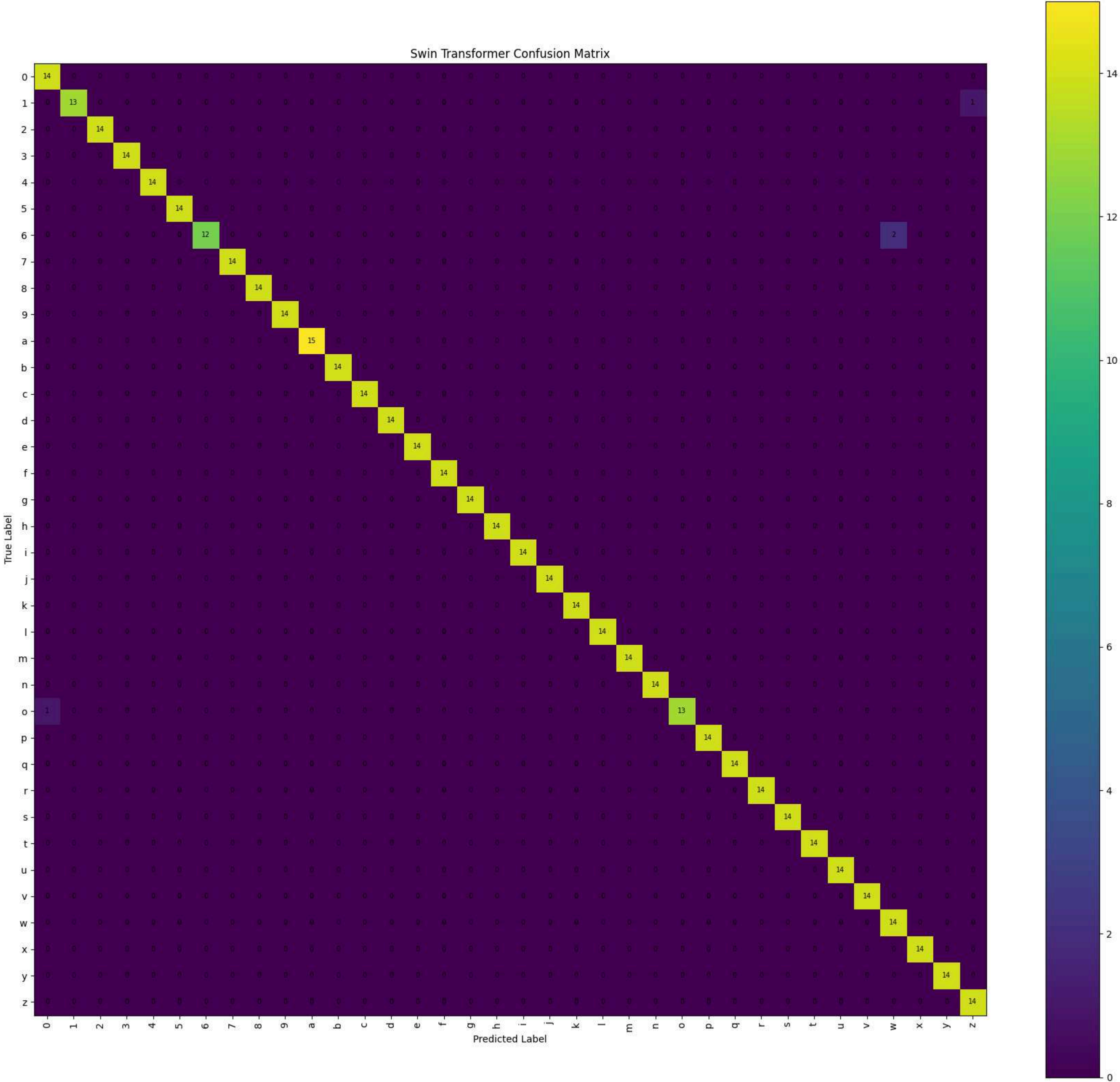
plt.colorbar()

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        plt.text(
            j,
            i,
            cm[i, j],
            ha="center",
            va="center",
            fontsize=7
        )

plt.tight_layout()
plt.savefig(
    CONFUSION_MATRIX_PATH,
    dpi=300,
    bbox_inches="tight"
)

plt.show()

print("Confusion Matrix saved at:", CONFUSION_MATRIX_PATH)
```



Confusion Matrix saved at: /content/drive/MyDrive/ASL_Transformer_Project/swin_confusion_matrix.png

```
In [25]: # Step 25: Save the Trained Model and Class Names
# This step saves the final Swin Transformer model and class labels to Google Drive.
```

```
torch.save(model.state_dict(), FINAL_MODEL_PATH)

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

print("Final model saved at:", FINAL_MODEL_PATH)
print("Class names saved at:", CLASS_NAMES_PATH)

Final model saved at: /content/drive/MyDrive/ASL_Transformer_Project/final_swin_asl_model.pth
Class names saved at: /content/drive/MyDrive/ASL_Transformer_Project/swin_class_names.txt
```

In [26]: # Step 26: Load the Saved Swin Transformer Model
This step loads the saved model and class names from Google Drive without retraining.

```
model = timm.create_model(
    "swin_tiny_patch4_window7_224",
    pretrained=False,
    num_classes=NUM_CLASSES
)

model.load_state_dict(
    torch.load(FINAL_MODEL_PATH, map_location=device)
)

model = model.to(device)
model.eval()

with open(CLASS_NAMES_PATH, "r") as file:
    CLASS_NAMES = [line.strip() for line in file.readlines()]

print("Swin Transformer model loaded successfully.")
print("Number of Classes:", len(CLASS_NAMES))
print("Class Names:", CLASS_NAMES)
```

Swin Transformer model loaded successfully.
Number of 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 [27]: # Step 27: Create the Prediction Function
This step prepares an uploaded image and predicts the ASL class using the saved Swin Transformer model.

```
def predict_asl_swin(image):

    if image is None:
        return "Please upload an image."

    model.eval()

    img = Image.fromarray(image).convert("RGB")

    img_tensor = val_test_transform(img).unsqueeze(0).to(device)

    with torch.no_grad():
        outputs = model(img_tensor)
        probabilities = torch.softmax(outputs, dim=1)
        confidence, predicted_index = torch.max(probabilities, 1)

    predicted_class = CLASS_NAMES[predicted_index.item()]
    confidence_value = confidence.item() * 100

    return f"Predicted Class: {predicted_class.upper()}\nConfidence: {confidence_value:.2f}%"
```

In [28]: # Step 28: Launch the Gradio Web Interface
This step creates a web interface for ASL image prediction using the Swin Transformer model.

```
app = gr.Interface(
    fn=predict_asl_swin,

    inputs=gr.Image(
        type="numpy",
        label="Upload ASL Image"
    ),

    outputs=gr.Textbox(
        label="Prediction Result"
    ),

    title="ASL Recognition using Swin Transformer",

    description="Upload an ASL hand sign image and the Swin Transformer model will predict the corresponding class.",

    live=False
)

app.launch(share=True)
```

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: <https://9a4a8431ce3a8ac129.gradio.live>

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (<https://huggingface.co/spaces>)

ASL Recognition using Swin Transformer

Upload an ASL hand sign image and the Swin Transformer model will predict the corresponding class.

Upload ASL Image

Drop Image Here

- Or -

Click to Upload

📁

📷

📄

Prediction Result

Flag

Clear

Submit

Use via API 🚀 · Built with Gradio 🍷 · Settings ⚙️

```
In [1]: # Step 1: Mount Google Drive
# This step connects Google Drive to access the ASL dataset and save the trained model.

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

Mounted at /content/drive

```
In [2]: # Step 2: Import required libraries
# This step imports all libraries needed for data processing, training, evaluation, and application.

import os
import shutil
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam

from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input

print("TensorFlow Version:", tf.__version__)
```

TensorFlow Version: 2.20.0

```
In [3]: # Step 3: Define dataset path
# This step finds the correct ASL dataset folder inside Google Drive.

possible_paths = [
    "/content/drive/MyDrive/ASL_Project/ASL_Dataset/asl_dataset",
]

ORIGINAL_DATASET_DIR = None

for path in possible_paths:
    if os.path.isdir(path):
        ORIGINAL_DATASET_DIR = path
        break

if ORIGINAL_DATASET_DIR is None:
    print("Dataset folder not found.")
    print("Folders inside MyDrive:")
    print(os.listdir("/content/drive/MyDrive"))
    raise FileNotFoundError("Dataset path is not correct.")

print("Dataset Path:", ORIGINAL_DATASET_DIR)
print("Folders:", os.listdir(ORIGINAL_DATASET_DIR))

Dataset Path: /content/drive/MyDrive/ASL_Project/ASL_Dataset/asl_dataset
Folders: ['x', 'z', 'y', 't', 'v', 'q', 'm', 's', 'p', 'w', 'o', 'l', 'n', 'u', 'r', 'k', 'h', 'i', 'e', 'f', 'b', '7', 'd', 'g', 'c', 'a', '9', '8', '6', 'j', '5', '4', '0', '1', '2', '3']
```

```
In [4]: # Step 4: Define valid classes
# This step defines the 36 target classes: digits from 0 to 9 and letters from a to z.

VALID_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'
]

NUM_CLASSES = len(VALID_CLASSES)

print("Number of Classes:", NUM_CLASSES)
print("Classes:", VALID_CLASSES)

Number of Classes: 36
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']
```

```
In [5]: # Step 5: Clean the dataset
# This step creates a clean dataset that contains only the required classes and image files.

CLEAN_DATASET_DIR = "/content/drive/MyDrive/ASL_Clean_Dataset_ResNet"

if os.path.exists(CLEAN_DATASET_DIR):
    shutil.rmtree(CLEAN_DATASET_DIR)

os.makedirs(CLEAN_DATASET_DIR, exist_ok=True)

def find_class_folder(dataset_dir, class_name):
    possible_names = [class_name, class_name.upper(), class_name.lower()]

    for name in possible_names:
        path = os.path.join(dataset_dir, name)
        if os.path.isdir(path):
            return path

    return None

total_clean_images = 0

for class_name in VALID_CLASSES:
    source_class_dir = find_class_folder(ORIGINAL_DATASET_DIR, class_name)
    target_class_dir = os.path.join(CLEAN_DATASET_DIR, class_name)
    os.makedirs(target_class_dir, exist_ok=True)

    if source_class_dir is None:
        print("Missing class folder:", class_name)
        continue

    count = 0

    for root, dirs, files in os.walk(source_class_dir):
        for file in files:
            if file.lower().endswith((".jpg", ".jpeg", ".png", ".bmp", ".webp")):
                src_file = os.path.join(root, file)
                dst_file = os.path.join(target_class_dir, file)
                shutil.copy2(src_file, dst_file)
                count += 1

    total_clean_images += count
    print(f"{class_name}: {count} images copied")

print("\nClean Dataset Created Successfully")
print("Total Clean Images:", total_clean_images)
print("Clean Dataset Path:", CLEAN_DATASET_DIR)
```

```
0: 70 images copied
1: 70 images copied
2: 70 images copied
3: 70 images copied
4: 70 images copied
5: 70 images copied
6: 70 images copied
7: 70 images copied
8: 70 images copied
9: 70 images copied
a: 71 images copied
b: 70 images copied
c: 70 images copied
d: 70 images copied
e: 70 images copied
f: 70 images copied
g: 70 images copied
h: 70 images copied
i: 70 images copied
j: 70 images copied
k: 70 images copied
l: 70 images copied
m: 70 images copied
n: 70 images copied
o: 70 images copied
p: 70 images copied
q: 70 images copied
r: 70 images copied
s: 70 images copied
t: 65 images copied
u: 70 images copied
v: 70 images copied
w: 70 images copied
x: 70 images copied
y: 70 images copied
z: 70 images copied

Clean Dataset Created Successfully
Total Clean Images: 2516
Clean Dataset Path: /content/drive/MyDrive/ASL_Clean_Dataset_ResNet
```

```
In [6]: # Step 6: Create split folders
# This step creates train, validation, and test folders for the dataset split.

SPLIT_DATASET_DIR = "/content/drive/MyDrive/ASL_Split_Dataset_ResNet"

train_dir = os.path.join(SPLIT_DATASET_DIR, "train")
validation_dir = os.path.join(SPLIT_DATASET_DIR, "validation")
test_dir = os.path.join(SPLIT_DATASET_DIR, "test")

if os.path.exists(SPLIT_DATASET_DIR):
    shutil.rmtree(SPLIT_DATASET_DIR)

os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

print("Split folders created successfully.")

Split folders created successfully.
```

```
In [7]: # Step 7: Split the dataset
# This step splits the clean dataset into 70% training, 10% validation, and 20% testing.

train_ratio = 0.70
validation_ratio = 0.10
test_ratio = 0.20

def split_and_copy_files(class_name):
    class_dir = os.path.join(CLEAN_DATASET_DIR, class_name)

    all_images = [
        os.path.join(class_dir, file)
        for file in os.listdir(class_dir)
        if file.lower().endswith((".jpg", ".jpeg", ".png", ".bmp", ".webp"))
    ]

    if len(all_images) < 3:
        print(f"Not enough images in class: {class_name}")
        return 0, 0, 0

    train_images, temp_images = train_test_split(
        all_images,
        test_size=0.30,
        random_state=42,
        shuffle=True
    )

    validation_images, test_images = train_test_split(
        temp_images,
        test_size=2/3,
        random_state=42,
        shuffle=True
    )

    split_data = {
        "train": train_images,
        "validation": validation_images,
        "test": test_images
    }

    for split_name, images in split_data.items():
        target_dir = os.path.join(SPLIT_DATASET_DIR, split_name, class_name)
        os.makedirs(target_dir, exist_ok=True)

        for image_path in images:
            shutil.copy2(image_path, target_dir)

    return len(train_images), len(validation_images), len(test_images)

total_train = 0
total_validation = 0
total_test = 0

for class_name in VALID_CLASSES:
    train_count, validation_count, test_count = split_and_copy_files(class_name)

    total_train += train_count
    total_validation += validation_count
    total_test += test_count

    print(f"{class_name}: Train={train_count}, Validation={validation_count}, Test={test_count}")

print("\nFinal Split:")
print("Training Images:", total_train)
print("Validation Images:", total_validation)
print("Testing Images:", total_test)
```

```
0: Train=49, Validation=7, Test=14
1: Train=49, Validation=7, Test=14
2: Train=49, Validation=7, Test=14
3: Train=49, Validation=7, Test=14
4: Train=49, Validation=7, Test=14
5: Train=49, Validation=7, Test=14
6: Train=49, Validation=7, Test=14
7: Train=49, Validation=7, Test=14
8: Train=49, Validation=7, Test=14
9: Train=49, Validation=7, Test=14
a: Train=49, Validation=7, Test=15
b: Train=49, Validation=7, Test=14
c: Train=49, Validation=7, Test=14
d: Train=49, Validation=7, Test=14
e: Train=49, Validation=7, Test=14
f: Train=49, Validation=7, Test=14
g: Train=49, Validation=7, Test=14
h: Train=49, Validation=7, Test=14
i: Train=49, Validation=7, Test=14
j: Train=49, Validation=7, Test=14
k: Train=49, Validation=7, Test=14
l: Train=49, Validation=7, Test=14
m: Train=49, Validation=7, Test=14
n: Train=49, Validation=7, Test=14
o: Train=49, Validation=7, Test=14
p: Train=49, Validation=7, Test=14
q: Train=49, Validation=7, Test=14
r: Train=49, Validation=7, Test=14
s: Train=49, Validation=7, Test=14
t: Train=45, Validation=6, Test=14
u: Train=49, Validation=7, Test=14
v: Train=49, Validation=7, Test=14
w: Train=49, Validation=7, Test=14
x: Train=49, Validation=7, Test=14
y: Train=49, Validation=7, Test=14
z: Train=49, Validation=7, Test=14

Final Split:
Training Images: 1760
Validation Images: 251
Testing Images: 505
```

```
In [9]: # Step 8: Define training parameters
# This step defines image size, batch size, epochs, and learning rate.

IMG_SIZE = 224
BATCH_SIZE = 16
EPOCHS = 30
LEARNING_RATE = 0.0001

print("Image Size:", IMG_SIZE)
print("Batch Size:", BATCH_SIZE)
print("Epochs:", EPOCHS)
print("Learning Rate:", LEARNING_RATE)

Image Size: 224
Batch Size: 16
Epochs: 30
Learning Rate: 0.0001
```

```
In [10]: # Step 9: Prepare data augmentation
# This step applies augmentation only to the training data to improve model generalization.

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"
)

val_test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input
)
```

```
In [11]: # Step 10: Create training generator
# This step loads the training images from the train folder.

train_generator = train_datagen.flow_from_directory(
    train_dir,
    classes=VALID_CLASSES,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=True,
    seed=42
)

print("Training Images:", train_generator.n)
print("Training Classes:", train_generator.num_classes)

Found 1760 images belonging to 36 classes.
Training Images: 1760
Training Classes: 36
```

In [12]: # Step 11: Create validation generator
This step loads the validation images from the validation folder.

```
validation_generator = val_test_datagen.flow_from_directory(
    validation_dir,
    classes=VALID_CLASSES,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

print("Validation Images:", validation_generator.n)
print("Validation Classes:", validation_generator.num_classes)
```

Found 251 images belonging to 36 classes.
Validation Images: 251
Validation Classes: 36

In [13]: # Step 12: Create test generator
This step loads the test images from the test folder without shuffling.

```
test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    classes=VALID_CLASSES,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

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

print("Class Names:", CLASS_NAMES)
print("Number of Classes:", NUM_CLASSES)

print("Test Images:", test_generator.n)
print("Test Classes:", test_generator.num_classes)
```

Found 505 images belonging to 36 classes.
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']
Number of Classes: 36
Test Images: 505
Test Classes: 36

In [14]: # Step 13: Build ResNet50 model
This step uses ResNet50 as a pre-trained feature extractor.

```
base_model = ResNet50(
    weights="imagenet",
    include_top=False,
    input_shape=(IMG_SIZE, IMG_SIZE, 3)
)

base_model.trainable = False

print("ResNet50 base model loaded successfully.")
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
[1m94765736/94765736][0m [32m-----][0m [37m][0m [1m1s][0m 0us/step
ResNet50 base model loaded successfully.

```
In [15]: # Step 14: Add classification layers
# This step adds custom layers to classify ASL images into 36 classes.
```

```
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="ResNet50_ASL_Model"
)

model.summary()
```

Model: "ResNet50_ASL_Model"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 224, 224, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_layer[0][0]
conv1_conv (Conv2D)	(None, 112, 112, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalizatio...	(None, 112, 112, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalizatio...	(None, 56, 56, 64)	256	conv2_block1_1_c...
conv2_block1_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_block1_1_b...
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_block1_1_r...
conv2_block1_2_bn (BatchNormalizatio...	(None, 56, 56, 64)	256	conv2_block1_2_c...
conv2_block1_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_block1_2_b...
conv2_block1_0_conv (Conv2D)	(None, 56, 56, 256)	16,640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 56, 56, 256)	16,640	conv2_block1_2_r...
conv2_block1_0_bn (BatchNormalizatio...	(None, 56, 56, 256)	1,024	conv2_block1_0_c...
conv2_block1_3_bn (BatchNormalizatio...	(None, 56, 56, 256)	1,024	conv2_block1_3_c...
conv2_block1_add (Add)	(None, 56, 56, 256)	0	conv2_block1_0_b... conv2_block1_3_b...
conv2_block1_out (Activation)	(None, 56, 56, 256)	0	conv2_block1_add...
conv2_block2_1_conv (Conv2D)	(None, 56, 56, 64)	16,448	conv2_block1_out...
conv2_block2_1_bn (BatchNormalizatio...	(None, 56, 56, 64)	256	conv2_block2_1_c...
conv2_block2_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_block2_1_b...
conv2_block2_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_block2_1_r...
conv2_block2_2_bn (BatchNormalizatio...	(None, 56, 56, 64)	256	conv2_block2_2_c...
conv2_block2_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_block2_2_b...
conv2_block2_3_conv (Conv2D)	(None, 56, 56, 256)	16,640	conv2_block2_2_r...
conv2_block2_3_bn (BatchNormalizatio...	(None, 56, 56, 256)	1,024	conv2_block2_3_c...
conv2_block2_add (Add)	(None, 56, 56, 256)	0	conv2_block1_out... conv2_block2_3_b...
conv2_block2_out (Activation)	(None, 56, 56, 256)	0	conv2_block2_add...
conv2_block3_1_conv (Conv2D)	(None, 56, 56, 64)	16,448	conv2_block2_out...
conv2_block3_1_bn (BatchNormalizatio...	(None, 56, 56, 64)	256	conv2_block3_1_c...
conv2_block3_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_block3_1_b...
conv2_block3_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_block3_1_r...
conv2_block3_2_bn (BatchNormalizatio...	(None, 56, 56, 64)	256	conv2_block3_2_c...
conv2_block3_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_block3_2_b...
conv2_block3_3_conv (Conv2D)	(None, 56, 56, 256)	16,640	conv2_block3_2_r...
conv2_block3_3_bn (BatchNormalizatio...	(None, 56, 56, 256)	1,024	conv2_block3_3_c...
conv2_block3_add (Add)	(None, 56, 56, 256)	0	conv2_block2_out... conv2_block3_3_b...
conv2_block3_out (Activation)	(None, 56, 56, 256)	0	conv2_block3_add...
conv3_block1_1_conv (Conv2D)	(None, 28, 28, 128)	32,896	conv2_block3_out...
conv3_block1_1_bn (BatchNormalizatio...	(None, 28, 28, 128)	512	conv3_block1_1_c...
conv3_block1_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block1_1_b...
conv3_block1_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_block1_1_r...
conv3_block1_2_bn (BatchNormalizatio...	(None, 28, 28, 128)	512	conv3_block1_2_c...
conv3_block1_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_block1_2_b...
conv3_block1_0_conv (Conv2D)	(None, 28, 28, 512)	131,584	conv2_block3_out...
conv3_block1_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_block1_2_r...
conv3_block1_0_bn (BatchNormalizatio...	(None, 28, 28, 512)	2,048	conv3_block1_0_c...
conv3_block1_3_bn (BatchNormalizatio...	(None, 28, 28, 512)	2,048	conv3_block1_3_c...
conv3_block1_add (Add)	(None, 28, 28, 512)	0	conv3_block1_0_b... conv3_block1_3_b...
conv3_block1_out (Activation)	(None, 28, 28, 512)	0	conv3_block1_add...
conv3_block2_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_block1_out...
conv3_block2_1_bn (BatchNormalizatio...	(None, 28, 28, 128)	512	conv3_block2_1_c...
conv3_block2_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block2_1_b...
conv3_block2_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_block2_1_r...
conv3_block2_2_bn (BatchNormalizatio...	(None, 28, 28, 128)	512	conv3_block2_2_c...
conv3_block2_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_block2_2_b...
conv3_block2_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_block2_2_r...
conv3_block2_3_bn (BatchNormalizatio...	(None, 28, 28, 512)	2,048	conv3_block2_3_c...
conv3_block2_add	(None, 28, 28, 512)	0	conv3_block1_out...

(Add)	512)		conv3_block2_3_b...
conv3_block2_out (Activation)	(None, 28, 28, 512)	0	conv3_block2_add...
conv3_block3_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_block2_out...
conv3_block3_1_bn (BatchNormalizatio...	(None, 28, 28, 128)	512	conv3_block3_1_c...
conv3_block3_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block3_1_b...
conv3_block3_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_block3_1_r...
conv3_block3_2_bn (BatchNormalizatio...	(None, 28, 28, 128)	512	conv3_block3_2_c...
conv3_block3_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_block3_2_b...
conv3_block3_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_block3_2_r...
conv3_block3_3_bn (BatchNormalizatio...	(None, 28, 28, 512)	2,048	conv3_block3_3_c...
conv3_block3_add (Add)	(None, 28, 28, 512)	0	conv3_block2_out... conv3_block3_3_b...
conv3_block3_out (Activation)	(None, 28, 28, 512)	0	conv3_block3_add...
conv3_block4_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_block3_out...
conv3_block4_1_bn (BatchNormalizatio...	(None, 28, 28, 128)	512	conv3_block4_1_c...
conv3_block4_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block4_1_b...
conv3_block4_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_block4_1_r...
conv3_block4_2_bn (BatchNormalizatio...	(None, 28, 28, 128)	512	conv3_block4_2_c...
conv3_block4_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_block4_2_b...
conv3_block4_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_block4_2_r...
conv3_block4_3_bn (BatchNormalizatio...	(None, 28, 28, 512)	2,048	conv3_block4_3_c...
conv3_block4_add (Add)	(None, 28, 28, 512)	0	conv3_block3_out... conv3_block4_3_b...
conv3_block4_out (Activation)	(None, 28, 28, 512)	0	conv3_block4_add...
conv4_block1_1_conv (Conv2D)	(None, 14, 14, 256)	131,328	conv3_block4_out...
conv4_block1_1_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block1_1_c...
conv4_block1_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block1_1_b...
conv4_block1_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block1_1_r...
conv4_block1_2_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block1_2_c...
conv4_block1_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block1_2_b...
conv4_block1_0_conv (Conv2D)	(None, 14, 14, 1024)	525,312	conv3_block4_out...
conv4_block1_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block1_2_r...
conv4_block1_0_bn (BatchNormalizatio...	(None, 14, 14, 1024)	4,096	conv4_block1_0_c...
conv4_block1_3_bn (BatchNormalizatio...	(None, 14, 14, 1024)	4,096	conv4_block1_3_c...
conv4_block1_add (Add)	(None, 14, 14, 1024)	0	conv4_block1_0_b... conv4_block1_3_b...
conv4_block1_out (Activation)	(None, 14, 14, 1024)	0	conv4_block1_add...
conv4_block2_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_block1_out...
conv4_block2_1_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block2_1_c...
conv4_block2_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block2_1_b...
conv4_block2_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block2_1_r...
conv4_block2_2_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block2_2_c...
conv4_block2_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block2_2_b...
conv4_block2_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block2_2_r...
conv4_block2_3_bn (BatchNormalizatio...	(None, 14, 14, 1024)	4,096	conv4_block2_3_c...
conv4_block2_add (Add)	(None, 14, 14, 1024)	0	conv4_block1_out... conv4_block2_3_b...
conv4_block2_out (Activation)	(None, 14, 14, 1024)	0	conv4_block2_add...
conv4_block3_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_block2_out...
conv4_block3_1_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block3_1_c...
conv4_block3_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block3_1_b...
conv4_block3_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block3_1_r...
conv4_block3_2_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block3_2_c...
conv4_block3_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block3_2_b...
conv4_block3_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block3_2_r...
conv4_block3_3_bn (BatchNormalizatio...	(None, 14, 14, 1024)	4,096	conv4_block3_3_c...
conv4_block3_add (Add)	(None, 14, 14, 1024)	0	conv4_block2_out... conv4_block3_3_b...
conv4_block3_out (Activation)	(None, 14, 14, 1024)	0	conv4_block3_add...
conv4_block4_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_block3_out...
conv4_block4_1_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block4_1_c...
conv4_block4_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block4_1_b...
conv4_block4_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block4_1_r...
conv4_block4_2_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block4_2_c...
conv4_block4_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block4_2_b...
conv4_block4_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block4_2_r...
conv4_block4_3_bn (BatchNormalizatio...	(None, 14, 14, 1024)	4,096	conv4_block4_3_c...
conv4_block4_add (Add)	(None, 14, 14, 1024)	0	conv4_block3_out... conv4_block4_3_b...
conv4_block4_out (Activation)	(None, 14, 14, 1024)	0	conv4_block4_add...
conv4_block5_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_block4_out...
conv4_block5_1_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block5_1_c...
conv4_block5_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block5_1_b...
conv4_block5_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block5_1_r...
conv4_block5_2_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block5_2_c...

conv4_block5_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block5_2_b...
conv4_block5_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block5_2_r...
conv4_block5_3_bn (BatchNormalizatio...	(None, 14, 14, 1024)	4,096	conv4_block5_3_c...
conv4_block5_add (Add)	(None, 14, 14, 1024)	0	conv4_block4_out... conv4_block5_3_b...
conv4_block5_out (Activation)	(None, 14, 14, 1024)	0	conv4_block5_add...
conv4_block6_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_block5_out...
conv4_block6_1_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block6_1_c...
conv4_block6_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_block6_1_b...
conv4_block6_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_block6_1_r...
conv4_block6_2_bn (BatchNormalizatio...	(None, 14, 14, 256)	1,024	conv4_block6_2_c...
conv4_block6_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_block6_2_b...
conv4_block6_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_block6_2_r...
conv4_block6_3_bn (BatchNormalizatio...	(None, 14, 14, 1024)	4,096	conv4_block6_3_c...
conv4_block6_add (Add)	(None, 14, 14, 1024)	0	conv4_block5_out... conv4_block6_3_b...
conv4_block6_out (Activation)	(None, 14, 14, 1024)	0	conv4_block6_add...
conv5_block1_1_conv (Conv2D)	(None, 7, 7, 512)	524,800	conv4_block6_out...
conv5_block1_1_bn (BatchNormalizatio...	(None, 7, 7, 512)	2,048	conv5_block1_1_c...
conv5_block1_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_block1_1_b...
conv5_block1_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_block1_1_r...
conv5_block1_2_bn (BatchNormalizatio...	(None, 7, 7, 512)	2,048	conv5_block1_2_c...
conv5_block1_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_block1_2_b...
conv5_block1_0_conv (Conv2D)	(None, 7, 7, 2048)	2,099,200	conv4_block6_out...
conv5_block1_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_block1_2_r...
conv5_block1_0_bn (BatchNormalizatio...	(None, 7, 7, 2048)	8,192	conv5_block1_0_c...
conv5_block1_3_bn (BatchNormalizatio...	(None, 7, 7, 2048)	8,192	conv5_block1_3_c...
conv5_block1_add (Add)	(None, 7, 7, 2048)	0	conv5_block1_0_b... conv5_block1_3_b...
conv5_block1_out (Activation)	(None, 7, 7, 2048)	0	conv5_block1_add...
conv5_block2_1_conv (Conv2D)	(None, 7, 7, 512)	1,049,088	conv5_block1_out...
conv5_block2_1_bn (BatchNormalizatio...	(None, 7, 7, 512)	2,048	conv5_block2_1_c...
conv5_block2_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_block2_1_b...
conv5_block2_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_block2_1_r...
conv5_block2_2_bn (BatchNormalizatio...	(None, 7, 7, 512)	2,048	conv5_block2_2_c...
conv5_block2_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_block2_2_b...
conv5_block2_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_block2_2_r...
conv5_block2_3_bn (BatchNormalizatio...	(None, 7, 7, 2048)	8,192	conv5_block2_3_c...
conv5_block2_add (Add)	(None, 7, 7, 2048)	0	conv5_block1_out... conv5_block2_3_b...
conv5_block2_out (Activation)	(None, 7, 7, 2048)	0	conv5_block2_add...
conv5_block3_1_conv (Conv2D)	(None, 7, 7, 512)	1,049,088	conv5_block2_out...
conv5_block3_1_bn (BatchNormalizatio...	(None, 7, 7, 512)	2,048	conv5_block3_1_c...
conv5_block3_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_block3_1_b...
conv5_block3_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_block3_1_r...
conv5_block3_2_bn (BatchNormalizatio...	(None, 7, 7, 512)	2,048	conv5_block3_2_c...
conv5_block3_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_block3_2_b...
conv5_block3_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_block3_2_r...
conv5_block3_3_bn (BatchNormalizatio...	(None, 7, 7, 2048)	8,192	conv5_block3_3_c...
conv5_block3_add (Add)	(None, 7, 7, 2048)	0	conv5_block2_out... conv5_block3_3_b...
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	conv5_block3_add...
global_average_poo... (GlobalAveragePool...	(None, 2048)	0	conv5_block3_out...
dense (Dense)	(None, 256)	524,544	global_average_p...
dropout (Dropout)	(None, 256)	0	dense[0][0]
dense_1 (Dense)	(None, 36)	9,252	dropout[0][0]

Total params: 24,121,508 (92.02 MB)

Trainable params: 533,796 (2.04 MB)

Non-trainable params: 23,587,712 (89.98 MB)

```
In [16]: # Step 15: Compile the model
# This step prepares the model for training using Adam optimizer and categorical_crossentropy.

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

```
In [17]: # Step 16: Prepare callbacks
# This step saves the best model and stops training when validation loss stops improving.

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(
        "best_resnet50_asl_model.keras",
        monitor="val_accuracy",
        save_best_only=True,
        verbose=1
    )
]
```

```
In [18]: # Step 17: Train the model
# This step trains ResNet50 using training data and validates it using validation data.
```

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

Epoch 1/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.0536 - loss: 3.9140
Epoch 1: val_accuracy improved from None to 0.25498, saving model to best_resnet50_asl_model.keras

Epoch 1: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m40s][0m 4s/step - accuracy: 0.0739 - loss: 3.5861 - val_accuracy: 0.2550 - val_loss: 3.0121 - learning_rate: 1.0000e-04
Epoch 2/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.1712 - loss: 3.0588
Epoch 2: val_accuracy improved from 0.25498 to 0.38247, saving model to best_resnet50_asl_model.keras

Epoch 2: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m433s][0m 4s/step - accuracy: 0.1937 - loss: 2.9695 - val_accuracy: 0.3825 - val_loss: 2.5889 - learning_rate: 1.0000e-04
Epoch 3/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.2527 - loss: 2.7278
Epoch 3: val_accuracy improved from 0.38247 to 0.50598, saving model to best_resnet50_asl_model.keras

Epoch 3: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m395s][0m 4s/step - accuracy: 0.2591 - loss: 2.6634 - val_accuracy: 0.5060 - val_loss: 2.2805 - learning_rate: 1.0000e-04
Epoch 4/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.3131 - loss: 2.4162
Epoch 4: val_accuracy improved from 0.50598 to 0.54980, saving model to best_resnet50_asl_model.keras

Epoch 4: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m393s][0m 4s/step - accuracy: 0.3222 - loss: 2.3741 - val_accuracy: 0.5498 - val_loss: 2.0274 - learning_rate: 1.0000e-04
Epoch 5/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.3532 - loss: 2.2365
Epoch 5: val_accuracy improved from 0.54980 to 0.63347, saving model to best_resnet50_asl_model.keras

Epoch 5: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m392s][0m 4s/step - accuracy: 0.3682 - loss: 2.1764 - val_accuracy: 0.6335 - val_loss: 1.7959 - learning_rate: 1.0000e-04
Epoch 6/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.4352 - loss: 1.9991
Epoch 6: val_accuracy did not improve from 0.63347
[1m10/110][0m @][32m-----][0m][37m][0m @][1m405s][0m 4s/step - accuracy: 0.4352 - loss: 1.9776 - val_accuracy: 0.6295 - val_loss: 1.6661 - learning_rate: 1.0000e-04
Epoch 7/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.4477 - loss: 1.8928
Epoch 7: val_accuracy improved from 0.63347 to 0.68526, saving model to best_resnet50_asl_model.keras

Epoch 7: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m408s][0m 4s/step - accuracy: 0.4585 - loss: 1.8705 - val_accuracy: 0.6853 - val_loss: 1.5262 - learning_rate: 1.0000e-04
Epoch 8/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.4665 - loss: 1.7908
Epoch 8: val_accuracy improved from 0.68526 to 0.70518, saving model to best_resnet50_asl_model.keras

Epoch 8: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m430s][0m 4s/step - accuracy: 0.4824 - loss: 1.7504 - val_accuracy: 0.7052 - val_loss: 1.4132 - learning_rate: 1.0000e-04
Epoch 9/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.5087 - loss: 1.6192
Epoch 9: val_accuracy improved from 0.70518 to 0.71713, saving model to best_resnet50_asl_model.keras

Epoch 9: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m405s][0m 4s/step - accuracy: 0.5267 - loss: 1.5928 - val_accuracy: 0.7171 - val_loss: 1.3043 - learning_rate: 1.0000e-04
Epoch 10/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.5329 - loss: 1.5669
Epoch 10: val_accuracy did not improve from 0.71713
[1m10/110][0m @][32m-----][0m][37m][0m @][1m427s][0m 4s/step - accuracy: 0.5455 - loss: 1.5378 - val_accuracy: 0.6932 - val_loss: 1.2566 - learning_rate: 1.0000e-04
Epoch 11/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.5579 - loss: 1.4675
Epoch 11: val_accuracy improved from 0.71713 to 0.74900, saving model to best_resnet50_asl_model.keras

Epoch 11: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m396s][0m 4s/step - accuracy: 0.5756 - loss: 1.4296 - val_accuracy: 0.7490 - val_loss: 1.1508 - learning_rate: 1.0000e-04
Epoch 12/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.5853 - loss: 1.3659
Epoch 12: val_accuracy did not improve from 0.74900
[1m10/110][0m @][32m-----][0m][37m][0m @][1m395s][0m 4s/step - accuracy: 0.6028 - loss: 1.3539 - val_accuracy: 0.7490 - val_loss: 1.0494 - learning_rate: 1.0000e-04
Epoch 13/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.6096 - loss: 1.2881
Epoch 13: val_accuracy improved from 0.74900 to 0.75299, saving model to best_resnet50_asl_model.keras

Epoch 13: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m394s][0m 4s/step - accuracy: 0.6091 - loss: 1.2886 - val_accuracy: 0.7530 - val_loss: 1.0311 - learning_rate: 1.0000e-04
Epoch 14/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.6304 - loss: 1.2444
Epoch 14: val_accuracy improved from 0.75299 to 0.77291, saving model to best_resnet50_asl_model.keras

Epoch 14: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m433s][0m 4s/step - accuracy: 0.6278 - loss: 1.2284 - val_accuracy: 0.7729 - val_loss: 0.9465 - learning_rate: 1.0000e-04
Epoch 15/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.6448 - loss: 1.1792
Epoch 15: val_accuracy improved from 0.77291 to 0.80080, saving model to best_resnet50_asl_model.keras

Epoch 15: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m396s][0m 4s/step - accuracy: 0.6386 - loss: 1.1919 - val_accuracy: 0.8008 - val_loss: 0.9233 - learning_rate: 1.0000e-04
Epoch 16/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.6541 - loss: 1.1419
Epoch 16: val_accuracy did not improve from 0.80080
[1m10/110][0m @][32m-----][0m][37m][0m @][1m397s][0m 4s/step - accuracy: 0.6477 - loss: 1.1295 - val_accuracy: 0.7849 - val_loss: 0.8790 - learning_rate: 1.0000e-04
Epoch 17/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.6638 - loss: 1.1060
Epoch 17: val_accuracy improved from 0.80080 to 0.81275, saving model to best_resnet50_asl_model.keras

Epoch 17: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m395s][0m 4s/step - accuracy: 0.6699 - loss: 1.0045 - val_accuracy: 0.8127 - val_loss: 0.8165 - learning_rate: 1.0000e-04
Epoch 18/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7030 - loss: 1.0193
Epoch 18: val_accuracy improved from 0.81275 to 0.83267, saving model to best_resnet50_asl_model.keras

Epoch 18: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m396s][0m 4s/step - accuracy: 0.6960 - loss: 1.0181 - val_accuracy: 0.8327 - val_loss: 0.7962 - learning_rate: 1.0000e-04
Epoch 19/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7069 - loss: 0.9957
Epoch 19: val_accuracy did not improve from 0.83267
[1m10/110][0m @][32m-----][0m][37m][0m @][1m397s][0m 4s/step - accuracy: 0.7136 - loss: 0.9788 - val_accuracy: 0.8247 - val_loss: 0.7663 - learning_rate: 1.0000e-04
Epoch 20/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7142 - loss: 0.9705
Epoch 20: val_accuracy did not improve from 0.83267
[1m10/110][0m @][32m-----][0m][37m][0m @][1m432s][0m 4s/step - accuracy: 0.7284 - loss: 0.9157 - val_accuracy: 0.8207 - val_loss: 0.7367 - learning_rate: 1.0000e-04
Epoch 21/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7395 - loss: 0.9078
Epoch 21: val_accuracy did not improve from 0.83267
[1m10/110][0m @][32m-----][0m][37m][0m @][1m396s][0m 4s/step - accuracy: 0.7176 - loss: 0.9301 - val_accuracy: 0.8287 - val_loss: 0.7089 - learning_rate: 1.0000e-04
Epoch 22/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7230 - loss: 0.9285
Epoch 22: val_accuracy improved from 0.83267 to 0.84861, saving model to best_resnet50_asl_model.keras

Epoch 22: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m399s][0m 4s/step - accuracy: 0.7398 - loss: 0.8869 - val_accuracy: 0.8486 - val_loss: 0.6753 - learning_rate: 1.0000e-04
Epoch 23/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7342 - loss: 0.8615
Epoch 23: val_accuracy did not improve from 0.84861
[1m10/110][0m @][32m-----][0m][37m][0m @][1m397s][0m 4s/step - accuracy: 0.7312 - loss: 0.8608 - val_accuracy: 0.8367 - val_loss: 0.6519 - learning_rate: 1.0000e-04
Epoch 24/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7481 - loss: 0.8184
Epoch 24: val_accuracy did not improve from 0.84861
[1m10/110][0m @][32m-----][0m][37m][0m @][1m439s][0m 4s/step - accuracy: 0.7437 - loss: 0.8167 - val_accuracy: 0.8327 - val_loss: 0.6634 - learning_rate: 1.0000e-04
Epoch 25/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7474 - loss: 0.8532
Epoch 25: val_accuracy did not improve from 0.84861
[1m10/110][0m @][32m-----][0m][37m][0m @][1m394s][0m 4s/step - accuracy: 0.7659 - loss: 0.8069 - val_accuracy: 0.8406 - val_loss: 0.6294 - learning_rate: 1.0000e-04
Epoch 26/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7596 - loss: 0.7654
Epoch 26: val_accuracy did not improve from 0.84861
[1m10/110][0m @][32m-----][0m][37m][0m @][1m398s][0m 4s/step - accuracy: 0.7688 - loss: 0.7678 - val_accuracy: 0.8367 - val_loss: 0.6034 - learning_rate: 1.0000e-04
Epoch 27/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7800 - loss: 0.7267
Epoch 27: val_accuracy did not improve from 0.84861
[1m10/110][0m @][32m-----][0m][37m][0m @][1m395s][0m 4s/step - accuracy: 0.7756 - loss: 0.7397 - val_accuracy: 0.8446 - val_loss: 0.5757 - learning_rate: 1.0000e-04
Epoch 28/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7825 - loss: 0.7319
Epoch 28: val_accuracy improved from 0.84861 to 0.85259, saving model to best_resnet50_asl_model.keras

Epoch 28: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m397s][0m 4s/step - accuracy: 0.7795 - loss: 0.7279 - val_accuracy: 0.8526 - val_loss: 0.5694 - learning_rate: 1.0000e-04
Epoch 29/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7857 - loss: 0.6942
Epoch 29: val_accuracy improved from 0.85259 to 0.85657, saving model to best_resnet50_asl_model.keras

Epoch 29: finished saving model to best_resnet50_asl_model.keras
[1m10/110][0m @][32m-----][0m][37m][0m @][1m441s][0m 4s/step - accuracy: 0.7744 - loss: 0.7212 - val_accuracy: 0.8566 - val_loss: 0.5567 - learning_rate: 1.0000e-04
Epoch 30/30
[1m10/110][0m @][32m-----][0m][37m][0m @][1m0s][0m 3s/step - accuracy: 0.7834 - loss: 0.7058
Epoch 30: val_accuracy did not improve from 0.85657
[1m10/110][0m @][32m-----][0m][37m][0m @][1m396s][0m 4s/step - accuracy: 0.7937 - loss: 0.6992 - val_accuracy: 0.8446 - val_loss: 0.5437 - learning_rate: 1.0000e-04
Restoring model weights from the end of the best epoch: 30.

```
In [19]: # Step 18: Evaluate the model on validation and test data
# This step calculates final validation and test accuracy.

val_loss, val_accuracy = model.evaluate(validation_generator, verbose=0)
test_loss, test_accuracy = model.evaluate(test_generator, verbose=0)

print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
print(f"Validation Loss: {val_loss:.4f}")

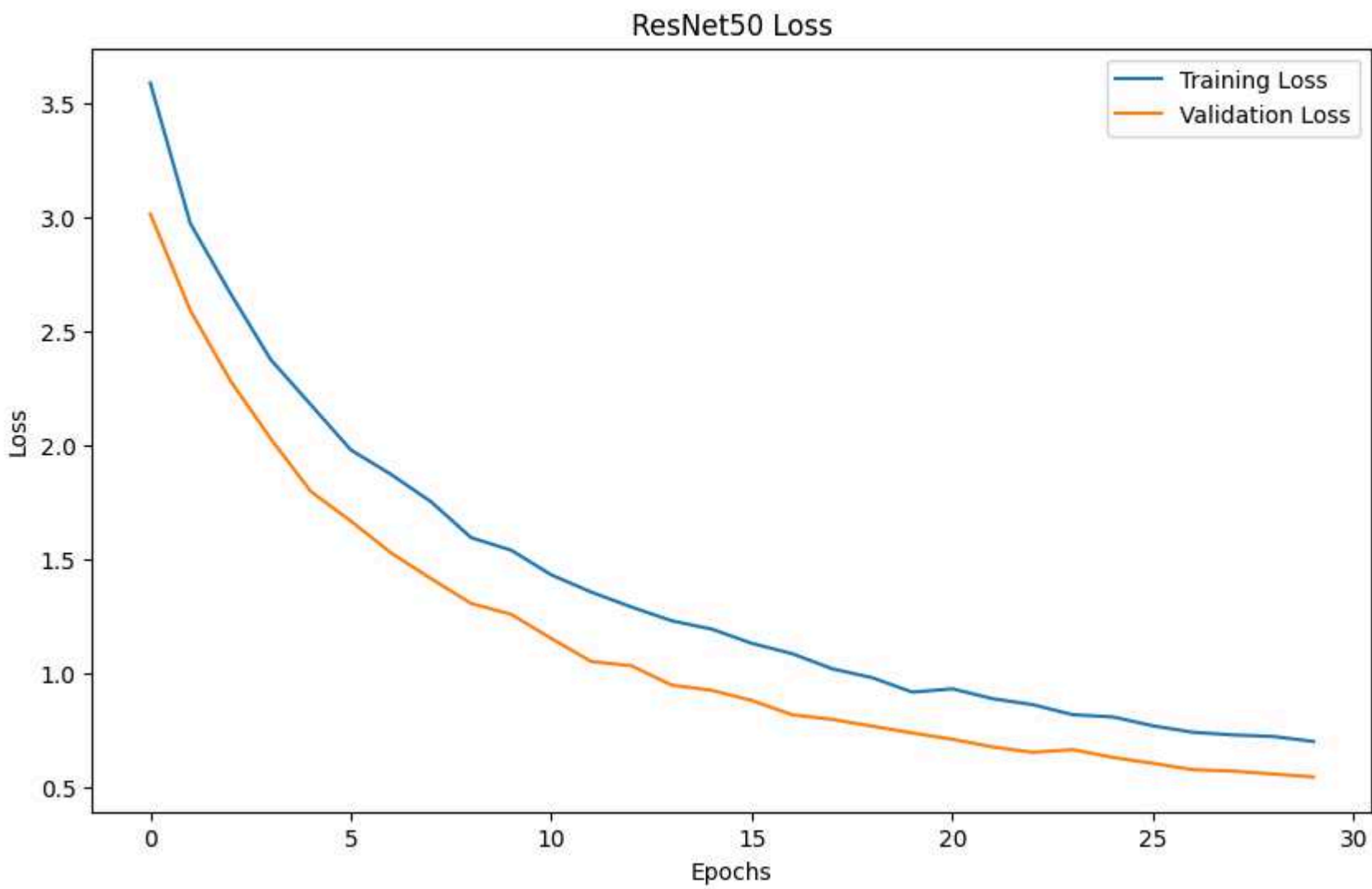
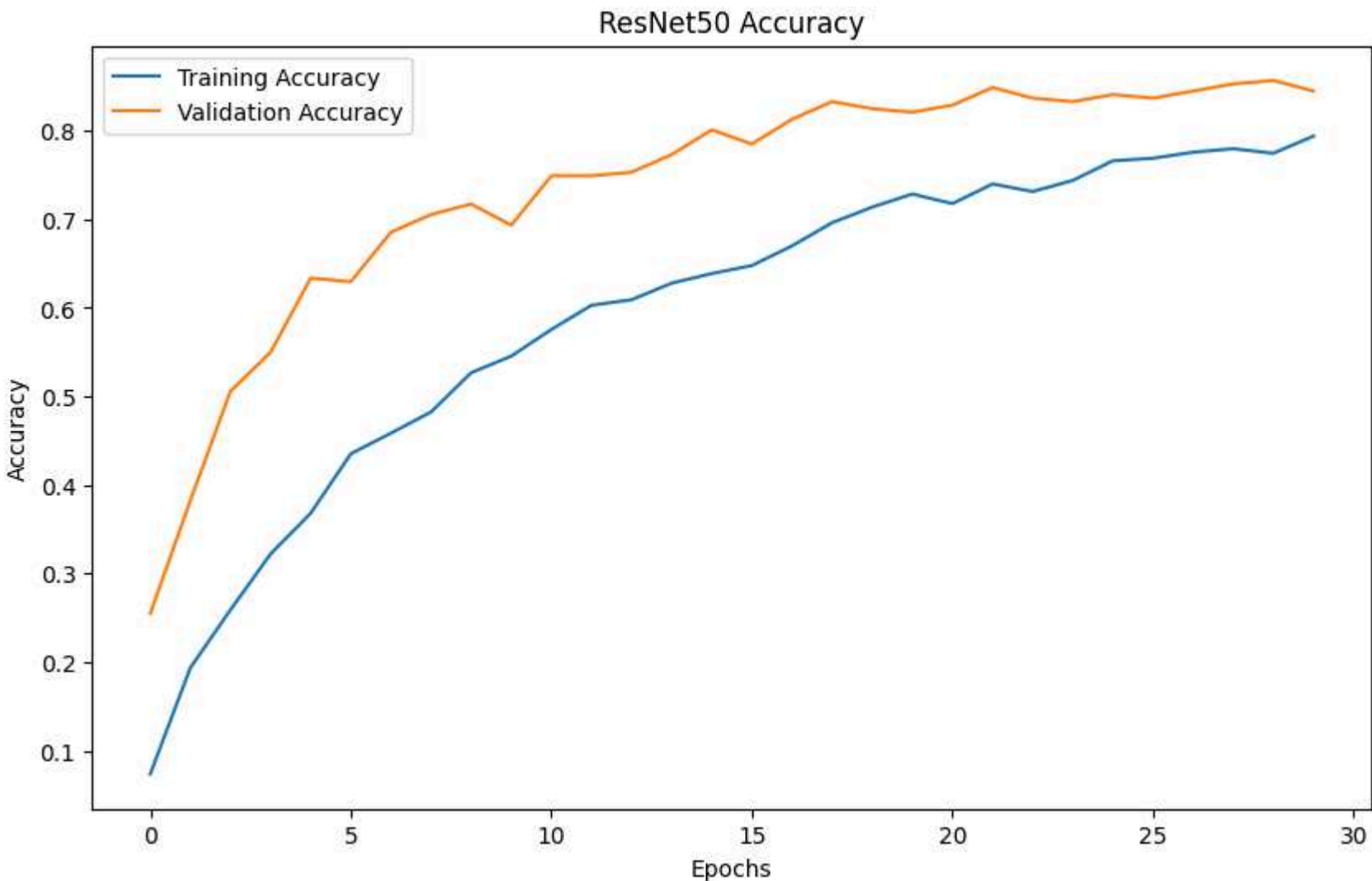
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")

Validation Accuracy: 84.46%
Validation Loss: 0.5437
Test Accuracy: 87.33%
Test Loss: 0.4907
```

```
In [20]: # Step 19: Plot training curves
# This step shows training and validation accuracy and loss across epochs.

plt.figure(figsize=(10, 6))
plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.title("ResNet50 Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title("ResNet50 Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



In [21]: # Step 20: Create Confusion Matrix
This step predicts test images and displays the confusion matrix.

```
test_generator.reset()

predictions = model.predict(test_generator)

y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(18, 16))
plt.imshow(cm)

plt.title("ResNet50 Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

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

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

plt.colorbar()

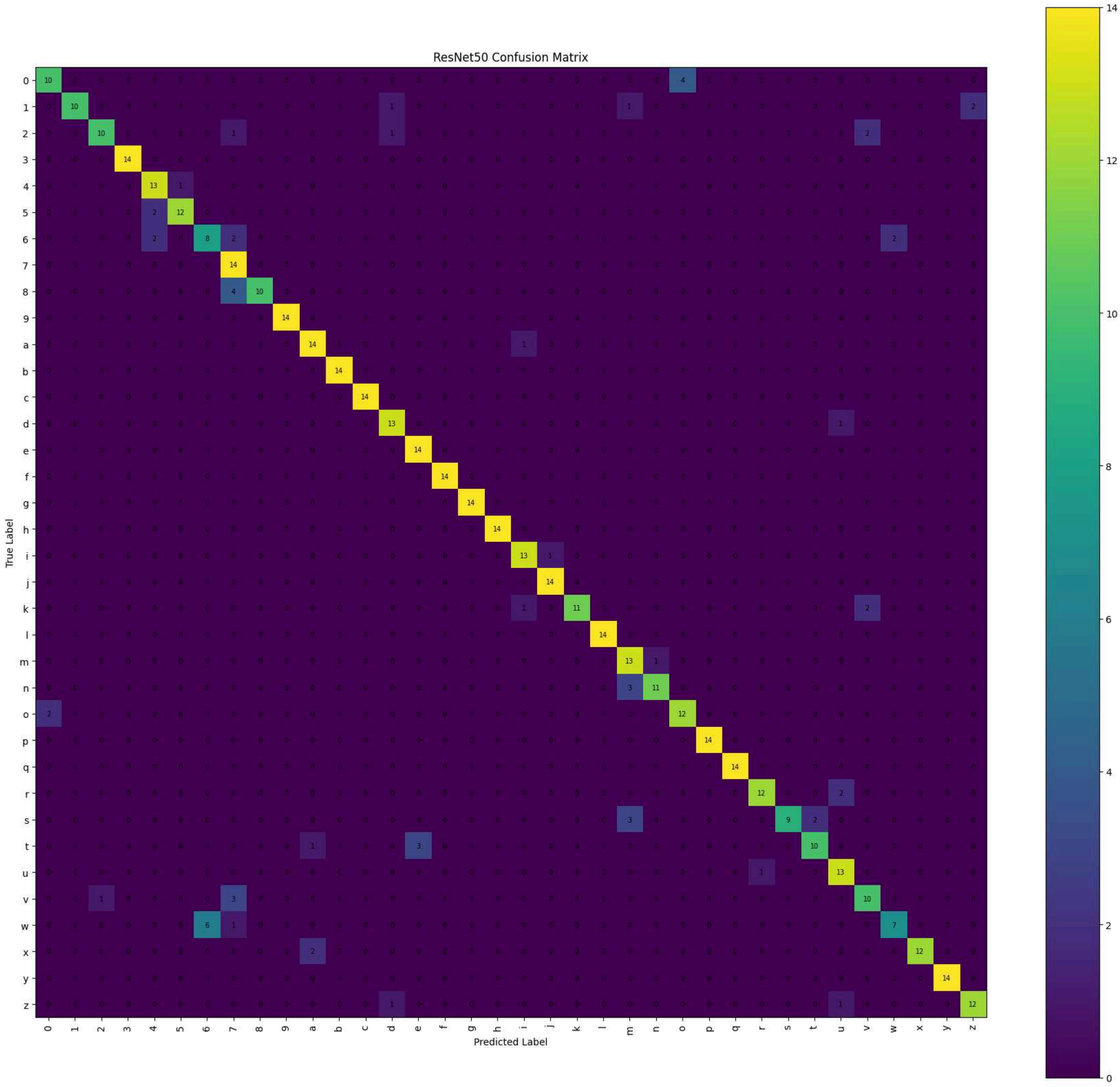
for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        plt.text(
            j,
            i,
            cm[i, j],
            ha="center",
            va="center",
            fontsize=7
        )

plt.tight_layout()
plt.show()

print("Test Accuracy:", accuracy_score(y_true, y_pred))

print("ResNet50 Classification Report")
print(classification_report(
    y_true,
    y_pred,
    target_names=CLASS_NAMES,
    zero_division=0
))
```

1m32/320m 32m 0m 37m0m 1m100s0m 3s/step



Test Accuracy: 0.8732673267326733

ResNet50 Classification Report				
	precision	recall	f1-score	support
0	0.83	0.71	0.77	14
1	1.00	0.71	0.83	14
2	0.91	0.71	0.80	14
3	1.00	1.00	1.00	14
4	0.76	0.93	0.84	14
5	0.92	0.86	0.89	14
6	0.57	0.57	0.57	14
7	0.56	1.00	0.72	14
8	1.00	0.71	0.83	14
9	1.00	1.00	1.00	14
a	0.82	0.93	0.88	15
b	1.00	1.00	1.00	14
c	1.00	1.00	1.00	14
d	0.81	0.93	0.87	14
e	0.82	1.00	0.90	14
f	1.00	1.00	1.00	14
g	1.00	1.00	1.00	14
h	1.00	1.00	1.00	14
i	0.87	0.93	0.90	14

j	0.93	1.00	0.97	14
k	1.00	0.79	0.88	14
l	1.00	1.00	1.00	14
m	0.65	0.93	0.76	14
n	0.92	0.79	0.85	14
o	0.75	0.86	0.80	14
p	1.00	1.00	1.00	14
q	1.00	1.00	1.00	14
r	0.92	0.86	0.89	14
s	1.00	0.64	0.78	14
t	0.83	0.71	0.77	14
u	0.76	0.93	0.84	14
v	0.71	0.71	0.71	14
w	0.78	0.50	0.61	14
x	1.00	0.86	0.92	14
y	1.00	1.00	1.00	14
z	0.86	0.86	0.86	14
accuracy			0.87	505
macro avg	0.89	0.87	0.87	505
weighted avg	0.89	0.87	0.87	505

```
In [22]: # Step 21: Save the confusion matrix image
# This step saves the confusion matrix as a PNG file.

plt.figure(figsize=(18, 16))
plt.imshow(cm)

plt.title("ResNet50 Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

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

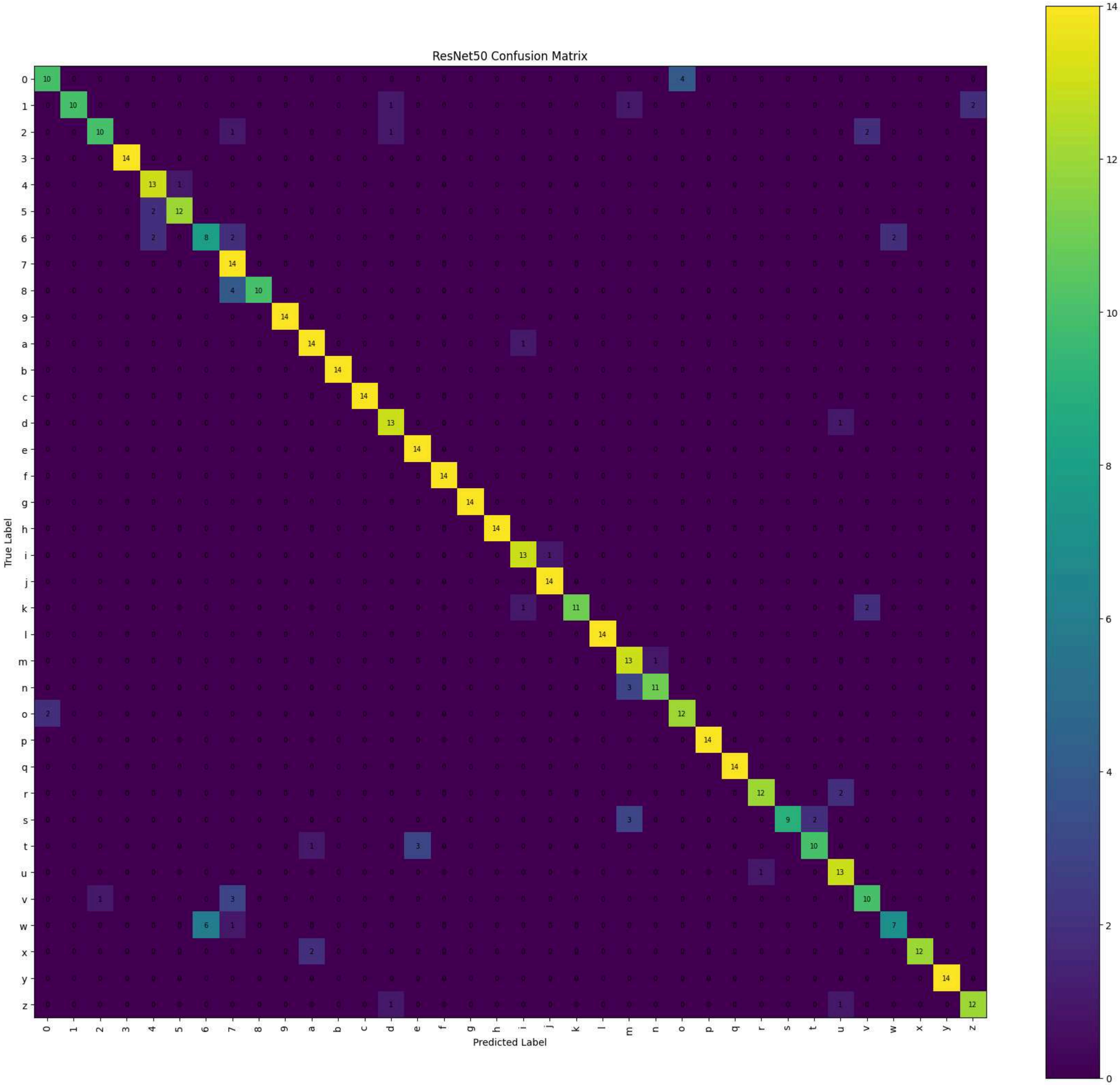
plt.yticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES
)

plt.colorbar()

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        plt.text(
            j,
            i,
            cm[i, j],
            ha="center",
            va="center",
            fontsize=7
        )

plt.tight_layout()
plt.savefig("resnet50_confusion_matrix.png", dpi=300, bbox_inches="tight")
plt.show()

print("Confusion Matrix saved as: resnet50_confusion_matrix.png")
```



Confusion Matrix saved as: resnet50_confusion_matrix.png

```
In [23]: # Step 22: Save the trained model and class names
# This step saves the final ResNet50 model and class labels to Google Drive.

import os

SAVE_PATH = "/content/drive/MyDrive/ASL_Project"
os.makedirs(SAVE_PATH, exist_ok=True)

# Save the trained model
model.save(os.path.join(SAVE_PATH, "final_resnet50_asl_model.keras"))

# Save class names
with open(os.path.join(SAVE_PATH, "asl_class_names.txt"), "w") as file:
    for class_name in CLASS_NAMES:
        file.write(class_name + "\n")

print("Model saved to:", os.path.join(SAVE_PATH, "final_resnet50_asl_model.keras"))
print("Class names saved to:", os.path.join(SAVE_PATH, "asl_class_names.txt"))
```

Model saved to: /content/drive/MyDrive/ASL_Project/final_resnet50_asl_model.keras
Class names saved to: /content/drive/MyDrive/ASL_Project/asl_class_names.txt

```
In [24]: # Step 23: Test prediction on one image
# This step allows testing the model on a single image manually.

from tensorflow.keras.preprocessing import image

def predict_single_image(image_path):
    img = image.load_img(image_path, target_size=(IMG_SIZE, IMG_SIZE))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    prediction = model.predict(img_array)

    predicted_index = np.argmax(prediction)
    predicted_class = CLASS_NAMES[predicted_index]
    confidence = np.max(prediction) * 100

    print("Predicted Class:", predicted_class)
    print(f"Confidence: {confidence:.2f}%")

    plt.imshow(image.load_img(image_path))
    plt.title(f"Prediction: {predicted_class.upper()} | Confidence: {confidence:.2f}%")
    plt.axis("off")
    plt.show()

    return predicted_class, confidence
```

```
In [25]: # Step 24: Install Gradio
# This step installs Gradio to run the ASL recognition application.

!pip install gradio -q
```

```
In [26]: # Step 25: Import application libraries
# This step imports the required libraries to run the web application.

import gradio as gr
from PIL import Image
import tensorflow as tf
```

```
In [27]: # Step 26: Load the saved model and class names
# This step loads the trained ResNet50 model and class labels from Google Drive.

import os
import tensorflow as tf

# Google Drive folder path
LOAD_PATH = "/content/drive/MyDrive/ASL_Project"

# Load the trained model
model = tf.keras.models.load_model(
    os.path.join(LOAD_PATH, "final_resnet50_asl_model.keras")
)

# Load class names
with open(os.path.join(LOAD_PATH, "asl_class_names.txt"), "r") as file:
    CLASS_NAMES = [line.strip() for line in file.readlines()]

print("Application model loaded successfully.")
print("Number of Classes:", len(CLASS_NAMES))
print("Class Names:", CLASS_NAMES)

Application model loaded successfully.
Number of 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 [28]: # Step 27: Create prediction function for the application
# This step prepares uploaded images and returns the predicted ASL class.

IMG_SIZE = 224

def predict_asl_resnet(image_input):
    if image_input is None:
        return "Please upload an image."

    img = Image.fromarray(image_input).convert("RGB")
    img = img.resize((IMG_SIZE, IMG_SIZE))

    img_array = np.array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    prediction = model.predict(img_array)

    predicted_index = np.argmax(prediction)
    predicted_class = CLASS_NAMES[predicted_index]
    confidence = np.max(prediction) * 100

    result = f"Predicted Class: {predicted_class.upper()}\nConfidence: {confidence:.2f}%"

    return result
```

```
In [29]: # Step 28: Run the application
# This step launches a Gradio web application to test the ResNet50 ASL model.

app = gr.Interface(
    fn=predict_asl_resnet,
    inputs=gr.Image(type="numpy", label="Upload ASL Image"),
    outputs=gr.Textbox(label="Prediction Result"),
    title="ASL Recognition using ResNet50",
    description="Upload an ASL hand sign image and the ResNet50 model will predict the class."
)

app.launch(share=True)

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://8dfb2cb23805b50c09.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

ASL Recognition using ResNet50

Upload an ASL hand sign image and the ResNet50 model will predict the class.

Upload ASL Image

Drop Image Here

- or -

Click to Upload

Clear

Submit

Prediction Result

Flag


```
In [1]: # Step 1: Mount Google Drive
# This step connects Google Drive to access the ASL dataset and save the trained model.

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

Mounted at /content/drive

```
In [2]: # Step 2: Import required libraries
# This step imports all libraries needed for data cleaning, splitting, training, evaluation, and application.

import os
import shutil
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam

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

print("TensorFlow Version:", tf.__version__)
```

TensorFlow Version: 2.20.0

```
In [4]: # Step 3: Define dataset path
# This step finds the correct ASL dataset folder inside Google Drive.

possible_paths = [
    "/content/drive/MyDrive/ASL_Project/ASL_Dataset/asl_dataset",
]

ORIGINAL_DATASET_DIR = None

for path in possible_paths:
    if os.path.isdir(path):
        ORIGINAL_DATASET_DIR = path
        break

if ORIGINAL_DATASET_DIR is None:
    print("Dataset folder not found.")
    print("Folders inside MyDrive:")
    print(os.listdir("/content/drive/MyDrive"))
    raise FileNotFoundError("Dataset path is not correct.")

print("Dataset Path:", ORIGINAL_DATASET_DIR)
print("Folders:", os.listdir(ORIGINAL_DATASET_DIR))

Dataset Path: /content/drive/MyDrive/ASL_Project/ASL_Dataset/asl_dataset
Folders: ['x', 'z', 'y', 't', 'v', 'q', 'm', 's', 'p', 'w', 'o', 'l', 'n', 'u', 'r', 'k', 'h', 'i', 'e', 'f', 'b', '7', 'd', 'g', 'c', 'a', '9', '8', '6', 'j', '5', '4', '0', '1', '2', '3']
```

```
In [5]: # Step 4: Define valid classes
# This step defines the 36 target classes: digits from 0 to 9 and letters from a to z.

VALID_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'
]

NUM_CLASSES = len(VALID_CLASSES)

print("Number of Classes:", NUM_CLASSES)
print("Classes:", VALID_CLASSES)

Number of Classes: 36
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']
```

```
In [6]: # Step 5: Clean the dataset
# This step creates a clean dataset that contains only the required classes and image files.

CLEAN_DATASET_DIR = "/content/drive/MyDrive/ASL_Clean_Dataset_MobileNet"

if os.path.exists(CLEAN_DATASET_DIR):
    shutil.rmtree(CLEAN_DATASET_DIR)

os.makedirs(CLEAN_DATASET_DIR, exist_ok=True)

def find_class_folder(dataset_dir, class_name):
    possible_names = [class_name, class_name.upper(), class_name.lower()]

    for name in possible_names:
        path = os.path.join(dataset_dir, name)
        if os.path.isdir(path):
            return path

    return None

total_clean_images = 0

for class_name in VALID_CLASSES:
    source_class_dir = find_class_folder(ORIGINAL_DATASET_DIR, class_name)
    target_class_dir = os.path.join(CLEAN_DATASET_DIR, class_name)
    os.makedirs(target_class_dir, exist_ok=True)

    if source_class_dir is None:
        print("Missing class folder:", class_name)
        continue

    count = 0

    for root, dirs, files in os.walk(source_class_dir):
        for file in files:
            if file.lower().endswith((".jpg", ".jpeg", ".png", ".bmp", ".webp")):
                src_file = os.path.join(root, file)
                dst_file = os.path.join(target_class_dir, file)
                shutil.copy2(src_file, dst_file)
                count += 1

    total_clean_images += count
    print(f"{class_name}: {count} images copied")

print("\nClean Dataset Created Successfully")
print("Total Clean Images:", total_clean_images)
print("Clean Dataset Path:", CLEAN_DATASET_DIR)
```

```
0: 70 images copied
1: 70 images copied
2: 70 images copied
3: 70 images copied
4: 70 images copied
5: 70 images copied
6: 70 images copied
7: 70 images copied
8: 70 images copied
9: 70 images copied
a: 71 images copied
b: 70 images copied
c: 70 images copied
d: 70 images copied
e: 70 images copied
f: 70 images copied
g: 70 images copied
h: 70 images copied
i: 70 images copied
j: 70 images copied
k: 70 images copied
l: 70 images copied
m: 70 images copied
n: 70 images copied
o: 70 images copied
p: 70 images copied
q: 70 images copied
r: 70 images copied
s: 70 images copied
t: 65 images copied
u: 70 images copied
v: 70 images copied
w: 70 images copied
x: 70 images copied
y: 70 images copied
z: 70 images copied

Clean Dataset Created Successfully
Total Clean Images: 2516
Clean Dataset Path: /content/drive/MyDrive/ASL_Clean_Dataset_MobileNet
```

```
In [7]: # Step 6: Create split folders
# This step creates train, validation, and test folders for the dataset split.

SPLIT_DATASET_DIR = "/content/drive/MyDrive/ASL_Split_Dataset_MobileNet"

train_dir = os.path.join(SPLIT_DATASET_DIR, "train")
validation_dir = os.path.join(SPLIT_DATASET_DIR, "validation")
test_dir = os.path.join(SPLIT_DATASET_DIR, "test")

if os.path.exists(SPLIT_DATASET_DIR):
    shutil.rmtree(SPLIT_DATASET_DIR)

os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

print("Split folders created successfully.")

Split folders created successfully.
```

```
In [8]: # Step 7: Split the dataset
# This step splits the clean dataset into 70% training, 10% validation, and 20% testing.

train_ratio = 0.70
validation_ratio = 0.10
test_ratio = 0.20

def split_and_copy_files(class_name):
    class_dir = os.path.join(CLEAN_DATASET_DIR, class_name)

    all_images = [
        os.path.join(class_dir, file)
        for file in os.listdir(class_dir)
        if file.lower().endswith((".jpg", ".jpeg", ".png", ".bmp", ".webp"))
    ]

    if len(all_images) < 3:
        print(f"Not enough images in class: {class_name}")
        return 0, 0, 0

    train_images, temp_images = train_test_split(
        all_images,
        test_size=0.30,
        random_state=42,
        shuffle=True
    )

    validation_images, test_images = train_test_split(
        temp_images,
        test_size=2/3,
        random_state=42,
        shuffle=True
    )

    split_data = {
        "train": train_images,
        "validation": validation_images,
        "test": test_images
    }

    for split_name, images in split_data.items():
        target_dir = os.path.join(SPLIT_DATASET_DIR, split_name, class_name)
        os.makedirs(target_dir, exist_ok=True)

        for image_path in images:
            shutil.copy2(image_path, target_dir)

    return len(train_images), len(validation_images), len(test_images)

total_train = 0
total_validation = 0
total_test = 0

for class_name in VALID_CLASSES:
    train_count, validation_count, test_count = split_and_copy_files(class_name)

    total_train += train_count
    total_validation += validation_count
    total_test += test_count

    print(f"{class_name}: Train={train_count}, Validation={validation_count}, Test={test_count}")

print("\nFinal Split:")
print("Training Images:", total_train)
print("Validation Images:", total_validation)
print("Testing Images:", total_test)
```

```
0: Train=49, Validation=7, Test=14
1: Train=49, Validation=7, Test=14
2: Train=49, Validation=7, Test=14
3: Train=49, Validation=7, Test=14
4: Train=49, Validation=7, Test=14
5: Train=49, Validation=7, Test=14
6: Train=49, Validation=7, Test=14
7: Train=49, Validation=7, Test=14
8: Train=49, Validation=7, Test=14
9: Train=49, Validation=7, Test=14
a: Train=49, Validation=7, Test=15
b: Train=49, Validation=7, Test=14
c: Train=49, Validation=7, Test=14
d: Train=49, Validation=7, Test=14
e: Train=49, Validation=7, Test=14
f: Train=49, Validation=7, Test=14
g: Train=49, Validation=7, Test=14
h: Train=49, Validation=7, Test=14
i: Train=49, Validation=7, Test=14
j: Train=49, Validation=7, Test=14
k: Train=49, Validation=7, Test=14
l: Train=49, Validation=7, Test=14
m: Train=49, Validation=7, Test=14
n: Train=49, Validation=7, Test=14
o: Train=49, Validation=7, Test=14
p: Train=49, Validation=7, Test=14
q: Train=49, Validation=7, Test=14
r: Train=49, Validation=7, Test=14
s: Train=49, Validation=7, Test=14
t: Train=45, Validation=6, Test=14
u: Train=49, Validation=7, Test=14
v: Train=49, Validation=7, Test=14
w: Train=49, Validation=7, Test=14
x: Train=49, Validation=7, Test=14
y: Train=49, Validation=7, Test=14
z: Train=49, Validation=7, Test=14
```

```
Final Split:
Training Images: 1760
Validation Images: 251
Testing Images: 505
```

```
In [10]: # Step 8: Define training parameters
# This step defines image size, batch size, epochs, and learning rate.
```

```
IMG_SIZE = 224
BATCH_SIZE = 16
EPOCHS = 30
LEARNING_RATE = 0.0001

print("Image Size:", IMG_SIZE)
print("Batch Size:", BATCH_SIZE)
print("Epochs:", EPOCHS)
print("Learning Rate:", LEARNING_RATE)
```

```
Image Size: 224
Batch Size: 16
Epochs: 30
Learning Rate: 0.0001
```

```
In [11]: # Step 9: Prepare data augmentation
# This step applies augmentation only to the training data to improve model generalization.
```

```
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"
)

val_test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input
)
```

```
In [12]: # Step 10: Create training generator
# This step loads the training images from the train folder.
```

```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    classes=VALID_CLASSES,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=True,
    seed=42
)

print("Training Images:", train_generator.n)
print("Training Classes:", train_generator.num_classes)
```

```
Found 1760 images belonging to 36 classes.
Training Images: 1760
Training Classes: 36
```

In [13]:

```
# Step 11: Create validation generator
# This step loads the validation images from the validation folder.

validation_generator = val_test_datagen.flow_from_directory(
    validation_dir,
    classes=VALID_CLASSES,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

print("Validation Images:", validation_generator.n)
print("Validation Classes:", validation_generator.num_classes)
```

```
Found 251 images belonging to 36 classes.
Validation Images: 251
Validation Classes: 36
```

In [14]:

```
# Step 12: Create test generator
# This step loads the test images from the test folder without shuffling.

test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    classes=VALID_CLASSES,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)

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

print("Class Names:", CLASS_NAMES)
print("Number of Classes:", NUM_CLASSES)

print("Test Images:", test_generator.n)
print("Test Classes:", test_generator.num_classes)
```

```
Found 505 images belonging to 36 classes.
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']
Number of Classes: 36
Test Images: 505
Test Classes: 36
```

In [15]:

```
# Step 13: Build MobileNetV2 model
# This step uses MobileNetV2 as a pre-trained feature extractor.

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

base_model.trainable = False

print("MobileNetV2 base model loaded successfully.")
```

```
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
[1m9406464/9406464][0m [32m-----[0m [0m[37m[0m [1m0s[0m 0us/step
MobileNetV2 base model loaded successfully.
```

In [16]: # Step 14: Add classification layers
This step adds custom layers to classify ASL images into 36 classes.

```
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"
)

model.summary()
```

Model: "MobileNetV2_ASL_Model"			
Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 224, 224, 3)	0	-
Conv1 (Conv2D)	(None, 112, 112, 32)	864	input_layer[0][0]
bn_Conv1 (BatchNormalization)	(None, 112, 112, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 112, 112, 32)	0	bn_Conv1[0][0]
expanded_conv_dept... (DepthwiseConv2D)	(None, 112, 112, 32)	288	Conv1_relu[0][0]
expanded_conv_dept... (BatchNormalization)	(None, 112, 112, 32)	128	expanded_conv_de...
expanded_conv_dept... (ReLU)	(None, 112, 112, 32)	0	expanded_conv_de...
expanded_conv_proj... (Conv2D)	(None, 112, 112, 16)	512	expanded_conv_de...
expanded_conv_proj... (BatchNormalization)	(None, 112, 112, 16)	64	expanded_conv_pr...
block_1_expand (Conv2D)	(None, 112, 112, 96)	1,536	expanded_conv_pr...
block_1_expand_BN (BatchNormalization)	(None, 112, 112, 96)	384	block_1_expand[0...
block_1_expand_relu (ReLU)	(None, 112, 112, 96)	0	block_1_expand_B...
block_1_pad (ZeroPadding2D)	(None, 113, 113, 96)	0	block_1_expand_r...
block_1_depthwise (DepthwiseConv2D)	(None, 56, 56, 96)	864	block_1_pad[0][0]
block_1_depthwise... (BatchNormalization)	(None, 56, 56, 96)	384	block_1_depthwis...
block_1_depthwise... (ReLU)	(None, 56, 56, 96)	0	block_1_depthwis...
block_1_project (Conv2D)	(None, 56, 56, 24)	2,304	block_1_depthwis...
block_1_project_BN (BatchNormalization)	(None, 56, 56, 24)	96	block_1_project[...
block_2_expand (Conv2D)	(None, 56, 56, 144)	3,456	block_1_project_...
block_2_expand_BN (BatchNormalization)	(None, 56, 56, 144)	576	block_2_expand[0...
block_2_expand_relu (ReLU)	(None, 56, 56, 144)	0	block_2_expand_B...
block_2_depthwise (DepthwiseConv2D)	(None, 56, 56, 144)	1,296	block_2_expand_r...
block_2_depthwise... (BatchNormalization)	(None, 56, 56, 144)	576	block_2_depthwis...
block_2_depthwise... (ReLU)	(None, 56, 56, 144)	0	block_2_depthwis...
block_2_project (Conv2D)	(None, 56, 56, 24)	3,456	block_2_depthwis...
block_2_project_BN (BatchNormalization)	(None, 56, 56, 24)	96	block_2_project[...
block_2_add (Add)	(None, 56, 56, 24)	0	block_1_project_... block_2_project_...
block_3_expand (Conv2D)	(None, 56, 56, 144)	3,456	block_2_add[0][0]
block_3_expand_BN (BatchNormalization)	(None, 56, 56, 144)	576	block_3_expand[0...
block_3_expand_relu (ReLU)	(None, 56, 56, 144)	0	block_3_expand_B...
block_3_pad (ZeroPadding2D)	(None, 57, 57, 144)	0	block_3_expand_r...
block_3_depthwise (DepthwiseConv2D)	(None, 28, 28, 144)	1,296	block_3_pad[0][0]
block_3_depthwise... (BatchNormalization)	(None, 28, 28, 144)	576	block_3_depthwis...
block_3_depthwise... (ReLU)	(None, 28, 28, 144)	0	block_3_depthwis...
block_3_project (Conv2D)	(None, 28, 28, 32)	4,608	block_3_depthwis...
block_3_project_BN (BatchNormalization)	(None, 28, 28, 32)	128	block_3_project[...
block_4_expand (Conv2D)	(None, 28, 28, 192)	6,144	block_3_project_...
block_4_expand_BN (BatchNormalization)	(None, 28, 28, 192)	768	block_4_expand[0...
block_4_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_4_expand_B...
block_4_depthwise (DepthwiseConv2D)	(None, 28, 28, 192)	1,728	block_4_expand_r...
block_4_depthwise... (BatchNormalization)	(None, 28, 28, 192)	768	block_4_depthwis...
block_4_depthwise... (ReLU)	(None, 28, 28, 192)	0	block_4_depthwis...
block_4_project (Conv2D)	(None, 28, 28, 32)	6,144	block_4_depthwis...
block_4_project_BN (BatchNormalization)	(None, 28, 28, 32)	128	block_4_project[...
block_4_add (Add)	(None, 28, 28, 32)	0	block_3_project_... block_4_project_...
block_5_expand (Conv2D)	(None, 28, 28, 192)	6,144	block_4_add[0][0]
block_5_expand_BN (BatchNormalization)	(None, 28, 28, 192)	768	block_5_expand[0...
block_5_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_5_expand_B...
block_5_depthwise (DepthwiseConv2D)	(None, 28, 28, 192)	1,728	block_5_expand_r...
block_5_depthwise... (BatchNormalization)	(None, 28, 28, 192)	768	block_5_depthwis...
block_5_depthwise... (ReLU)	(None, 28, 28, 192)	0	block_5_depthwis...
block_5_project (Conv2D)	(None, 28, 28, 32)	6,144	block_5_depthwis...
block_5_project_BN (BatchNormalization)	(None, 28, 28, 32)	128	block_5_project[...
block_5_add (Add)	(None, 28, 28, 32)	0	block_4_add[0][0... block_5_project_...
block_6_expand (Conv2D)	(None, 28, 28, 192)	6,144	block_5_add[0][0]
block_6_expand_BN (BatchNormalization)	(None, 28, 28, 192)	768	block_6_expand[0...
block_6_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_6_expand_B...
block_6_pad (ZeroPadding2D)	(None, 29, 29, 192)	0	block_6_expand_r...
block_6_depthwise (DepthwiseConv2D)	(None, 14, 14, 192)	1,728	block_6_pad[0][0]
block_6_depthwise...	(None, 14, 14, 192)	768	block_6_depthwis...

(BatchNormalizatio...	192)		
block_6_depthwise_... (ReLU)	(None, 14, 14, 192)	0	block_6_depthwis...
block_6_project (Conv2D)	(None, 14, 14, 64)	12,288	block_6_depthwis...
block_6_project_BN (BatchNormalizatio...	(None, 14, 14, 64)	256	block_6_project[...
block_7_expand (Conv2D)	(None, 14, 14, 384)	24,576	block_6_project_...
block_7_expand_BN (BatchNormalizatio...	(None, 14, 14, 384)	1,536	block_7_expand[0...
block_7_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_7_expand_B...
block_7_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3,456	block_7_expand_r...
block_7_depthwise_... (BatchNormalizatio...	(None, 14, 14, 384)	1,536	block_7_depthwis...
block_7_depthwise_... (ReLU)	(None, 14, 14, 384)	0	block_7_depthwis...
block_7_project (Conv2D)	(None, 14, 14, 64)	24,576	block_7_depthwis...
block_7_project_BN (BatchNormalizatio...	(None, 14, 14, 64)	256	block_7_project[...
block_7_add (Add)	(None, 14, 14, 64)	0	block_6_project_... block_7_project_...
block_8_expand (Conv2D)	(None, 14, 14, 384)	24,576	block_7_add[0][0]
block_8_expand_BN (BatchNormalizatio...	(None, 14, 14, 384)	1,536	block_8_expand[0...
block_8_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_8_expand_B...
block_8_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3,456	block_8_expand_r...
block_8_depthwise_... (BatchNormalizatio...	(None, 14, 14, 384)	1,536	block_8_depthwis...
block_8_depthwise_... (ReLU)	(None, 14, 14, 384)	0	block_8_depthwis...
block_8_project (Conv2D)	(None, 14, 14, 64)	24,576	block_8_depthwis...
block_8_project_BN (BatchNormalizatio...	(None, 14, 14, 64)	256	block_8_project[...
block_8_add (Add)	(None, 14, 14, 64)	0	block_7_add[0][0... block_8_project_...
block_9_expand (Conv2D)	(None, 14, 14, 384)	24,576	block_8_add[0][0]
block_9_expand_BN (BatchNormalizatio...	(None, 14, 14, 384)	1,536	block_9_expand[0...
block_9_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_9_expand_B...
block_9_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3,456	block_9_expand_r...
block_9_depthwise_... (BatchNormalizatio...	(None, 14, 14, 384)	1,536	block_9_depthwis...
block_9_depthwise_... (ReLU)	(None, 14, 14, 384)	0	block_9_depthwis...
block_9_project (Conv2D)	(None, 14, 14, 64)	24,576	block_9_depthwis...
block_9_project_BN (BatchNormalizatio...	(None, 14, 14, 64)	256	block_9_project[...
block_9_add (Add)	(None, 14, 14, 64)	0	block_8_add[0][0... block_9_project_...
block_10_expand (Conv2D)	(None, 14, 14, 384)	24,576	block_9_add[0][0]
block_10_expand_BN (BatchNormalizatio...	(None, 14, 14, 384)	1,536	block_10_expand[...
block_10_expand_re... (ReLU)	(None, 14, 14, 384)	0	block_10_expand_...
block_10_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3,456	block_10_expand_...
block_10_depthwise... (BatchNormalizatio...	(None, 14, 14, 384)	1,536	block_10_depthwi...
block_10_depthwise... (ReLU)	(None, 14, 14, 384)	0	block_10_depthwi...
block_10_project (Conv2D)	(None, 14, 14, 96)	36,864	block_10_depthwi...
block_10_project_BN (BatchNormalizatio...	(None, 14, 14, 96)	384	block_10_project...
block_11_expand (Conv2D)	(None, 14, 14, 576)	55,296	block_10_project...
block_11_expand_BN (BatchNormalizatio...	(None, 14, 14, 576)	2,304	block_11_expand[...
block_11_expand_re... (ReLU)	(None, 14, 14, 576)	0	block_11_expand_...
block_11_depthwise (DepthwiseConv2D)	(None, 14, 14, 576)	5,184	block_11_expand_...
block_11_depthwise... (BatchNormalizatio...	(None, 14, 14, 576)	2,304	block_11_depthwi...
block_11_depthwise... (ReLU)	(None, 14, 14, 576)	0	block_11_depthwi...
block_11_project (Conv2D)	(None, 14, 14, 96)	55,296	block_11_depthwi...
block_11_project_BN (BatchNormalizatio...	(None, 14, 14, 96)	384	block_11_project...
block_11_add (Add)	(None, 14, 14, 96)	0	block_10_project... block_11_project...
block_12_expand (Conv2D)	(None, 14, 14, 576)	55,296	block_11_add[0][...
block_12_expand_BN (BatchNormalizatio...	(None, 14, 14, 576)	2,304	block_12_expand[...
block_12_expand_re... (ReLU)	(None, 14, 14, 576)	0	block_12_expand_...
block_12_depthwise (DepthwiseConv2D)	(None, 14, 14, 576)	5,184	block_12_expand_...
block_12_depthwise... (BatchNormalizatio...	(None, 14, 14, 576)	2,304	block_12_depthwi...
block_12_depthwise... (ReLU)	(None, 14, 14, 576)	0	block_12_depthwi...
block_12_project (Conv2D)	(None, 14, 14, 96)	55,296	block_12_depthwi...
block_12_project_BN (BatchNormalizatio...	(None, 14, 14, 96)	384	block_12_project...
block_12_add (Add)	(None, 14, 14, 96)	0	block_11_add[0][... block_12_project...
block_13_expand (Conv2D)	(None, 14, 14, 576)	55,296	block_12_add[0][...
block_13_expand_BN (BatchNormalizatio...	(None, 14, 14, 576)	2,304	block_13_expand[...
block_13_expand_re... (ReLU)	(None, 14, 14, 576)	0	block_13_expand_...
block_13_pad (ZeroPadding2D)	(None, 15, 15, 576)	0	block_13_expand_...
block_13_depthwise (DepthwiseConv2D)	(None, 7, 7, 576)	5,184	block_13_pad[0][...
block_13_depthwise... (BatchNormalizatio...	(None, 7, 7, 576)	2,304	block_13_depthwi...
block_13_depthwise... (ReLU)	(None, 7, 7, 576)	0	block_13_depthwi...
block_13_project (Conv2D)	(None, 7, 7, 160)	92,160	block_13_depthwi...
block_13_project_BN (BatchNormalizatio...	(None, 7, 7, 160)	640	block_13_project...
block_14_expand (Conv2D)	(None, 7, 7, 960)	153,600	block_13_project...
block_14_expand_BN (BatchNormalizatio...	(None, 7, 7, 960)	3,840	block_14_expand[...
block_14_expand_re... (ReLU)	(None, 7, 7, 960)	0	block_14_expand_...

block_14_depthwise (DepthwiseConv2D)	(None, 7, 7, 960)	8,640	block_14_expand_...
block_14_depthwise... (BatchNormalizatio...	(None, 7, 7, 960)	3,840	block_14_depthwi...
block_14_depthwise... (ReLU)	(None, 7, 7, 960)	0	block_14_depthwi...
block_14_project (Conv2D)	(None, 7, 7, 160)	153,600	block_14_depthwi...
block_14_project_BN (BatchNormalizatio...	(None, 7, 7, 160)	640	block_14_project...
block_14_add (Add)	(None, 7, 7, 160)	0	block_13_project.. block_14_project...
block_15_expand (Conv2D)	(None, 7, 7, 960)	153,600	block_14_add[0][...
block_15_expand_BN (BatchNormalizatio...	(None, 7, 7, 960)	3,840	block_15_expand[...
block_15_expand_re... (ReLU)	(None, 7, 7, 960)	0	block_15_expand_...
block_15_depthwise (DepthwiseConv2D)	(None, 7, 7, 960)	8,640	block_15_expand_...
block_15_depthwise... (BatchNormalizatio...	(None, 7, 7, 960)	3,840	block_15_depthwi...
block_15_depthwise... (ReLU)	(None, 7, 7, 960)	0	block_15_depthwi...
block_15_project (Conv2D)	(None, 7, 7, 160)	153,600	block_15_depthwi...
block_15_project_BN (BatchNormalizatio...	(None, 7, 7, 160)	640	block_15_project...
block_15_add (Add)	(None, 7, 7, 160)	0	block_14_add[0][... block_15_project...
block_16_expand (Conv2D)	(None, 7, 7, 960)	153,600	block_15_add[0][...
block_16_expand_BN (BatchNormalizatio...	(None, 7, 7, 960)	3,840	block_16_expand[...
block_16_expand_re... (ReLU)	(None, 7, 7, 960)	0	block_16_expand_...
block_16_depthwise (DepthwiseConv2D)	(None, 7, 7, 960)	8,640	block_16_expand_...
block_16_depthwise... (BatchNormalizatio...	(None, 7, 7, 960)	3,840	block_16_depthwi...
block_16_depthwise... (ReLU)	(None, 7, 7, 960)	0	block_16_depthwi...
block_16_project (Conv2D)	(None, 7, 7, 320)	307,200	block_16_depthwi...
block_16_project_BN (BatchNormalizatio...	(None, 7, 7, 320)	1,280	block_16_project...
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409,600	block_16_project...
Conv_1_bn (BatchNormalizatio...	(None, 7, 7, 1280)	5,120	Conv_1[0][0]
out_relu (ReLU)	(None, 7, 7, 1280)	0	Conv_1_bn[0][0]
global_average_poo... (GlobalAveragePool...	(None, 1280)	0	out_relu[0][0]
dense (Dense)	(None, 256)	327,936	global_average_p...
dropout (Dropout)	(None, 256)	0	dense[0][0]
dense_1 (Dense)	(None, 36)	9,252	dropout[0][0]

Total params: 2,595,172 (9.90 MB)

Trainable params: 337,188 (1.29 MB)

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

```
In [17]: # Step 15: Compile the model
# This step prepares the model for training using Adam optimizer and categorical_crossentropy.

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

```
In [18]: # Step 16: Prepare callbacks
# This step saves the best model and stops training when validation loss stops improving.

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(
        "best_mobilenetv2_as1_model.keras",
        monitor="val_accuracy",
        save_best_only=True,
        verbose=1
    )
]
```

```
In [19]: # Step 17: Train the model
# This step trains MobileNetV2 using training data and validates it using validation data.

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

Epoch 1/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 939ms/step - accuracy: 0.0407 - loss: 3.9520
Epoch 1: val_accuracy improved from None to 0.20717, saving model to best_mobilenetv2_asl_model.keras

Epoch 1: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m126s @ 0m 1s/step - accuracy: 0.0636 - loss: 3.6473 - val_accuracy: 0.2072 - val_loss: 3.0609 - learning_rate: 1.0000e-04
Epoch 2/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 952ms/step - accuracy: 0.1444 - loss: 3.0957
Epoch 2: val_accuracy improved from 0.20717 to 0.37849, saving model to best_mobilenetv2_asl_model.keras

Epoch 2: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m116s @ 0m 1s/step - accuracy: 0.1733 - loss: 2.9944 - val_accuracy: 0.3785 - val_loss: 2.5608 - learning_rate: 1.0000e-04
Epoch 3/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 980ms/step - accuracy: 0.2541 - loss: 2.6570
Epoch 3: val_accuracy improved from 0.37849 to 0.48207, saving model to best_mobilenetv2_asl_model.keras

Epoch 3: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m149s @ 0m 1s/step - accuracy: 0.2693 - loss: 2.5811 - val_accuracy: 0.4821 - val_loss: 2.1596 - learning_rate: 1.0000e-04
Epoch 4/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 1s/step - accuracy: 0.3265 - loss: 2.3294
Epoch 4: val_accuracy improved from 0.48207 to 0.54582, saving model to best_mobilenetv2_asl_model.keras

Epoch 4: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m128s @ 0m 1s/step - accuracy: 0.3392 - loss: 2.2981 - val_accuracy: 0.5458 - val_loss: 1.8633 - learning_rate: 1.0000e-04
Epoch 5/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 960ms/step - accuracy: 0.3860 - loss: 2.1169
Epoch 5: val_accuracy improved from 0.54582 to 0.60159, saving model to best_mobilenetv2_asl_model.keras

Epoch 5: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m119s @ 0m 1s/step - accuracy: 0.4051 - loss: 2.0516 - val_accuracy: 0.6016 - val_loss: 1.6281 - learning_rate: 1.0000e-04
Epoch 6/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 955ms/step - accuracy: 0.4357 - loss: 1.8880
Epoch 6: val_accuracy improved from 0.60159 to 0.66534, saving model to best_mobilenetv2_asl_model.keras

Epoch 6: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m117s @ 0m 1s/step - accuracy: 0.4369 - loss: 1.8715 - val_accuracy: 0.6653 - val_loss: 1.4666 - learning_rate: 1.0000e-04
Epoch 7/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 950ms/step - accuracy: 0.5244 - loss: 1.7065
Epoch 7: val_accuracy improved from 0.66534 to 0.71713, saving model to best_mobilenetv2_asl_model.keras

Epoch 7: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m143s @ 0m 1s/step - accuracy: 0.5091 - loss: 1.7038 - val_accuracy: 0.7171 - val_loss: 1.3215 - learning_rate: 1.0000e-04
Epoch 8/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 934ms/step - accuracy: 0.5312 - loss: 1.6241
Epoch 8: val_accuracy improved from 0.71713 to 0.73705, saving model to best_mobilenetv2_asl_model.keras

Epoch 8: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m115s @ 0m 1s/step - accuracy: 0.5483 - loss: 1.5794 - val_accuracy: 0.7371 - val_loss: 1.2151 - learning_rate: 1.0000e-04
Epoch 9/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 944ms/step - accuracy: 0.5827 - loss: 1.4565
Epoch 9: val_accuracy improved from 0.73705 to 0.75697, saving model to best_mobilenetv2_asl_model.keras

Epoch 9: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m114s @ 0m 1s/step - accuracy: 0.5727 - loss: 1.4620 - val_accuracy: 0.7570 - val_loss: 1.1156 - learning_rate: 1.0000e-04
Epoch 10/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 946ms/step - accuracy: 0.5899 - loss: 1.4013
Epoch 10: val_accuracy did not improve from 0.75697
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m116s @ 0m 1s/step - accuracy: 0.6040 - loss: 1.3621 - val_accuracy: 0.7450 - val_loss: 1.0412 - learning_rate: 1.0000e-04
Epoch 11/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 930ms/step - accuracy: 0.6069 - loss: 1.3272
Epoch 11: val_accuracy did not improve from 0.75697
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m114s @ 0m 1s/step - accuracy: 0.6187 - loss: 1.2992 - val_accuracy: 0.7570 - val_loss: 0.9683 - learning_rate: 1.0000e-04
Epoch 12/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 907ms/step - accuracy: 0.6136 - loss: 1.2386
Epoch 12: val_accuracy improved from 0.75697 to 0.77291, saving model to best_mobilenetv2_asl_model.keras

Epoch 12: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m122s @ 0m 1s/step - accuracy: 0.6381 - loss: 1.2098 - val_accuracy: 0.7729 - val_loss: 0.8971 - learning_rate: 1.0000e-04
Epoch 13/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 968ms/step - accuracy: 0.6389 - loss: 1.1894
Epoch 13: val_accuracy did not improve from 0.77291
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m117s @ 0m 1s/step - accuracy: 0.6460 - loss: 1.1691 - val_accuracy: 0.7570 - val_loss: 0.8698 - learning_rate: 1.0000e-04
Epoch 14/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 934ms/step - accuracy: 0.6865 - loss: 1.0981
Epoch 14: val_accuracy improved from 0.77291 to 0.78088, saving model to best_mobilenetv2_asl_model.keras

Epoch 14: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m116s @ 0m 1s/step - accuracy: 0.6875 - loss: 1.0904 - val_accuracy: 0.7809 - val_loss: 0.8180 - learning_rate: 1.0000e-04
Epoch 15/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 939ms/step - accuracy: 0.7194 - loss: 1.0138
Epoch 15: val_accuracy improved from 0.78088 to 0.80478, saving model to best_mobilenetv2_asl_model.keras

Epoch 15: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m116s @ 0m 1s/step - accuracy: 0.7000 - loss: 1.0317 - val_accuracy: 0.8048 - val_loss: 0.7604 - learning_rate: 1.0000e-04
Epoch 16/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 924ms/step - accuracy: 0.6999 - loss: 0.9903
Epoch 16: val_accuracy did not improve from 0.80478
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m112s @ 0m 1s/step - accuracy: 0.7011 - loss: 0.9990 - val_accuracy: 0.8048 - val_loss: 0.7404 - learning_rate: 1.0000e-04
Epoch 17/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 967ms/step - accuracy: 0.7054 - loss: 0.9272
Epoch 17: val_accuracy did not improve from 0.80478
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m119s @ 0m 1s/step - accuracy: 0.7119 - loss: 0.9260 - val_accuracy: 0.7968 - val_loss: 0.7040 - learning_rate: 1.0000e-04
Epoch 18/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 946ms/step - accuracy: 0.7057 - loss: 0.9434
Epoch 18: val_accuracy did not improve from 0.80478
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m116s @ 0m 1s/step - accuracy: 0.7210 - loss: 0.9209 - val_accuracy: 0.8048 - val_loss: 0.6886 - learning_rate: 1.0000e-04
Epoch 19/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 940ms/step - accuracy: 0.7255 - loss: 0.9256
Epoch 19: val_accuracy improved from 0.80478 to 0.80876, saving model to best_mobilenetv2_asl_model.keras

Epoch 19: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m114s @ 0m 1s/step - accuracy: 0.7261 - loss: 0.9168 - val_accuracy: 0.8088 - val_loss: 0.6543 - learning_rate: 1.0000e-04
Epoch 20/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 928ms/step - accuracy: 0.7479 - loss: 0.8781
Epoch 20: val_accuracy improved from 0.80876 to 0.82072, saving model to best_mobilenetv2_asl_model.keras

Epoch 20: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m112s @ 0m 1s/step - accuracy: 0.7506 - loss: 0.8615 - val_accuracy: 0.8207 - val_loss: 0.6280 - learning_rate: 1.0000e-04
Epoch 21/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 953ms/step - accuracy: 0.7561 - loss: 0.8330
Epoch 21: val_accuracy did not improve from 0.82072
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m118s @ 0m 1s/step - accuracy: 0.7420 - loss: 0.8560 - val_accuracy: 0.8167 - val_loss: 0.6172 - learning_rate: 1.0000e-04
Epoch 22/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 923ms/step - accuracy: 0.7674 - loss: 0.7964
Epoch 22: val_accuracy did not improve from 0.82072
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m138s @ 0m 1s/step - accuracy: 0.7631 - loss: 0.8027 - val_accuracy: 0.8088 - val_loss: 0.6114 - learning_rate: 1.0000e-04
Epoch 23/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 951ms/step - accuracy: 0.7484 - loss: 0.7967
Epoch 23: val_accuracy did not improve from 0.82072
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m115s @ 0m 1s/step - accuracy: 0.7574 - loss: 0.7955 - val_accuracy: 0.8207 - val_loss: 0.5867 - learning_rate: 1.0000e-04
Epoch 24/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 946ms/step - accuracy: 0.7785 - loss: 0.7790
Epoch 24: val_accuracy did not improve from 0.82072
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m114s @ 0m 1s/step - accuracy: 0.7716 - loss: 0.7573 - val_accuracy: 0.8127 - val_loss: 0.5787 - learning_rate: 1.0000e-04
Epoch 25/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 954ms/step - accuracy: 0.7674 - loss: 0.7685
Epoch 25: val_accuracy improved from 0.82072 to 0.83665, saving model to best_mobilenetv2_asl_model.keras

Epoch 25: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m118s @ 0m 1s/step - accuracy: 0.7847 - loss: 0.7372 - val_accuracy: 0.8367 - val_loss: 0.5538 - learning_rate: 1.0000e-04
Epoch 26/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 934ms/step - accuracy: 0.8098 - loss: 0.6843
Epoch 26: val_accuracy did not improve from 0.83665
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m123s @ 0m 1s/step - accuracy: 0.8051 - loss: 0.6901 - val_accuracy: 0.8207 - val_loss: 0.5327 - learning_rate: 1.0000e-04
Epoch 27/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 901ms/step - accuracy: 0.8139 - loss: 0.6277
Epoch 27: val_accuracy did not improve from 0.83665
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m111s @ 0m 1s/step - accuracy: 0.7920 - loss: 0.6683 - val_accuracy: 0.8287 - val_loss: 0.5201 - learning_rate: 1.0000e-04
Epoch 28/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 901ms/step - accuracy: 0.7821 - loss: 0.7233
Epoch 28: val_accuracy did not improve from 0.83665
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m111s @ 0m 1s/step - accuracy: 0.7943 - loss: 0.6944 - val_accuracy: 0.8207 - val_loss: 0.5200 - learning_rate: 1.0000e-04
Epoch 29/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 911ms/step - accuracy: 0.7987 - loss: 0.6432
Epoch 29: val_accuracy did not improve from 0.83665
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m112s @ 0m 1s/step - accuracy: 0.7994 - loss: 0.6460 - val_accuracy: 0.8367 - val_loss: 0.5153 - learning_rate: 1.0000e-04
Epoch 30/30
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m0s @ 0m 904ms/step - accuracy: 0.8137 - loss: 0.6333
Epoch 30: val_accuracy improved from 0.83665 to 0.85259, saving model to best_mobilenetv2_asl_model.keras

Epoch 30: finished saving model to best_mobilenetv2_asl_model.keras
110/110 [0m @ 32m] [0m @ 37m @ 0m @ 1m111s @ 0m 1s/step - accuracy: 0.8040 - loss: 0.6428 - val_accuracy: 0.8526 - val_loss: 0.4784 - learning_rate: 1.0000e-04
Restoring model weights from the end of the best epoch: 30.

```
In [20]: # Step 18: Evaluate the model on validation and test data
# This step calculates final validation and test accuracy.

val_loss, val_accuracy = model.evaluate(validation_generator, verbose=0)
test_loss, test_accuracy = model.evaluate(test_generator, verbose=0)

print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
print(f"Validation Loss: {val_loss:.4f}")

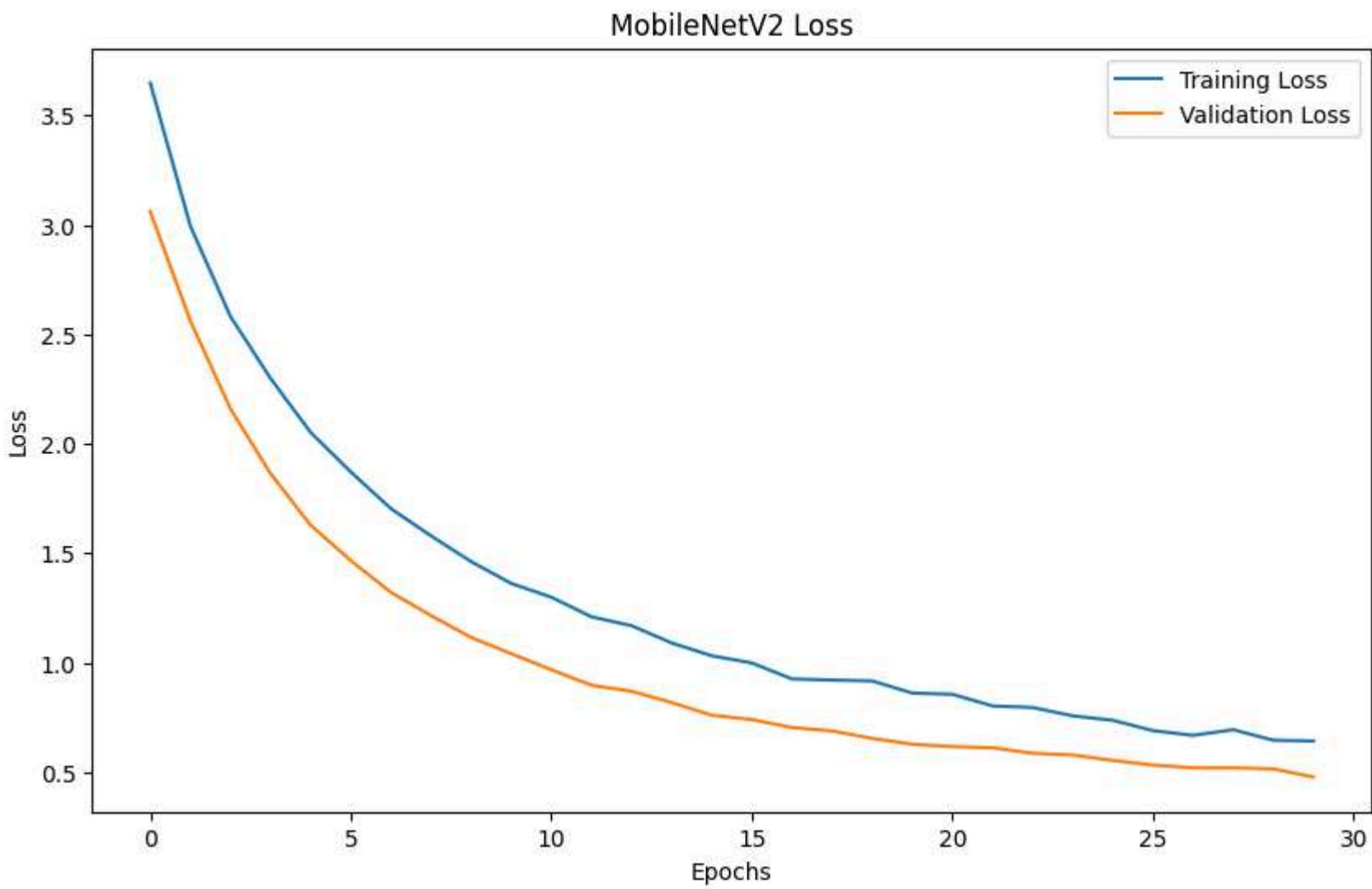
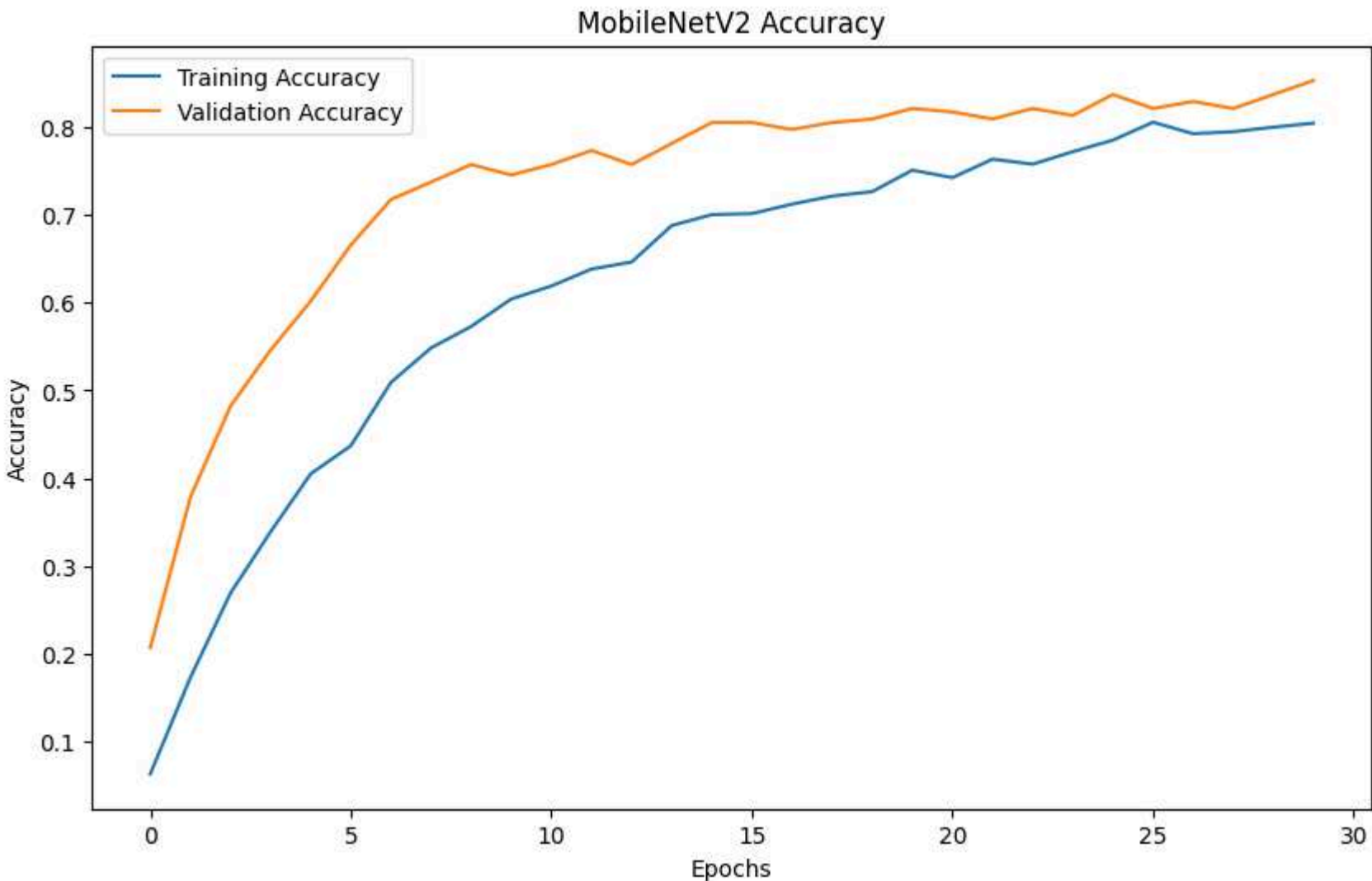
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")

Validation Accuracy: 85.26%
Validation Loss: 0.4784
Test Accuracy: 87.92%
Test Loss: 0.4512
```

```
In [21]: # Step 19: Plot training curves
# This step shows training and validation accuracy and loss across epochs.

plt.figure(figsize=(10, 6))
plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.title("MobileNetV2 Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

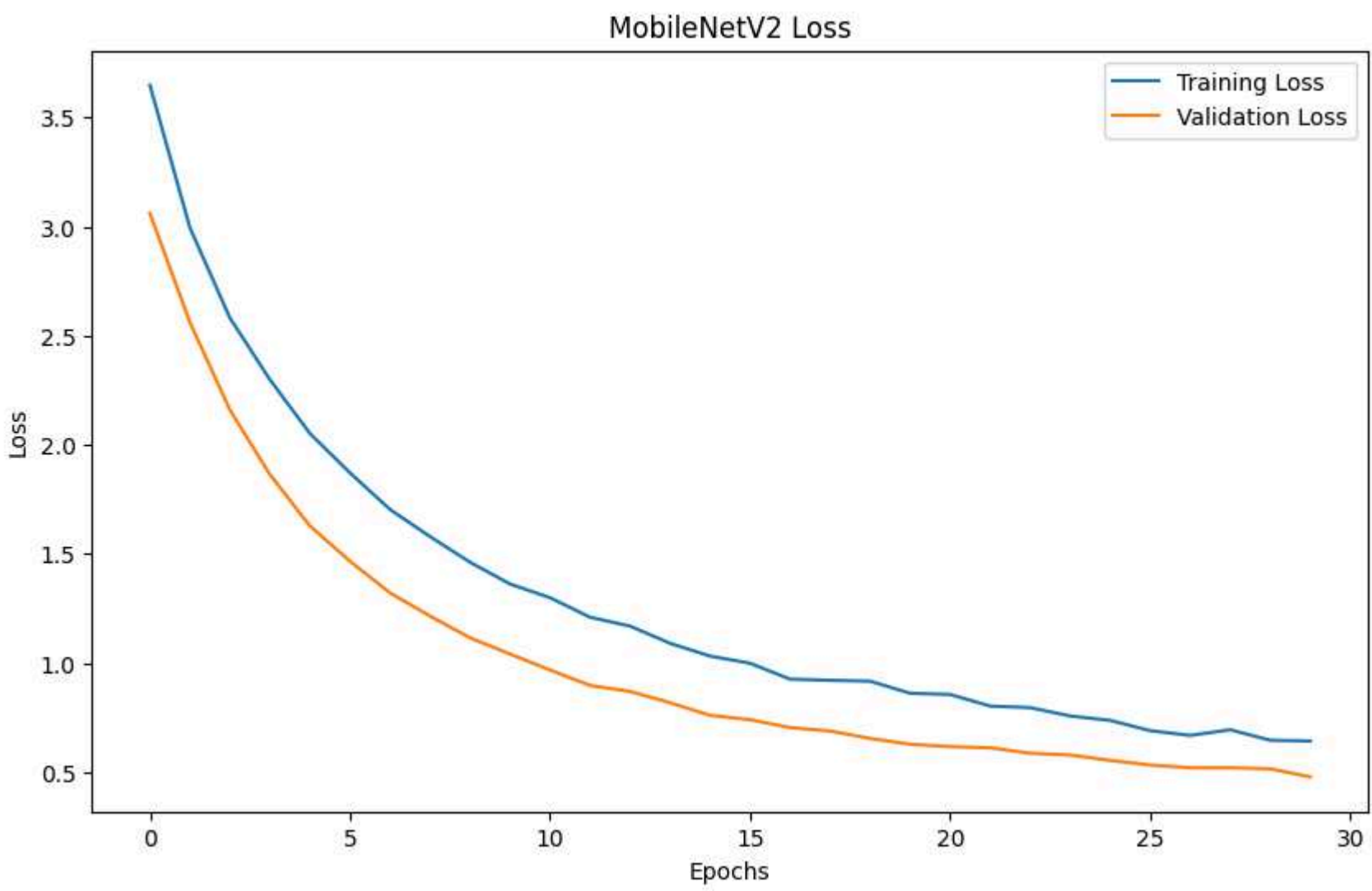
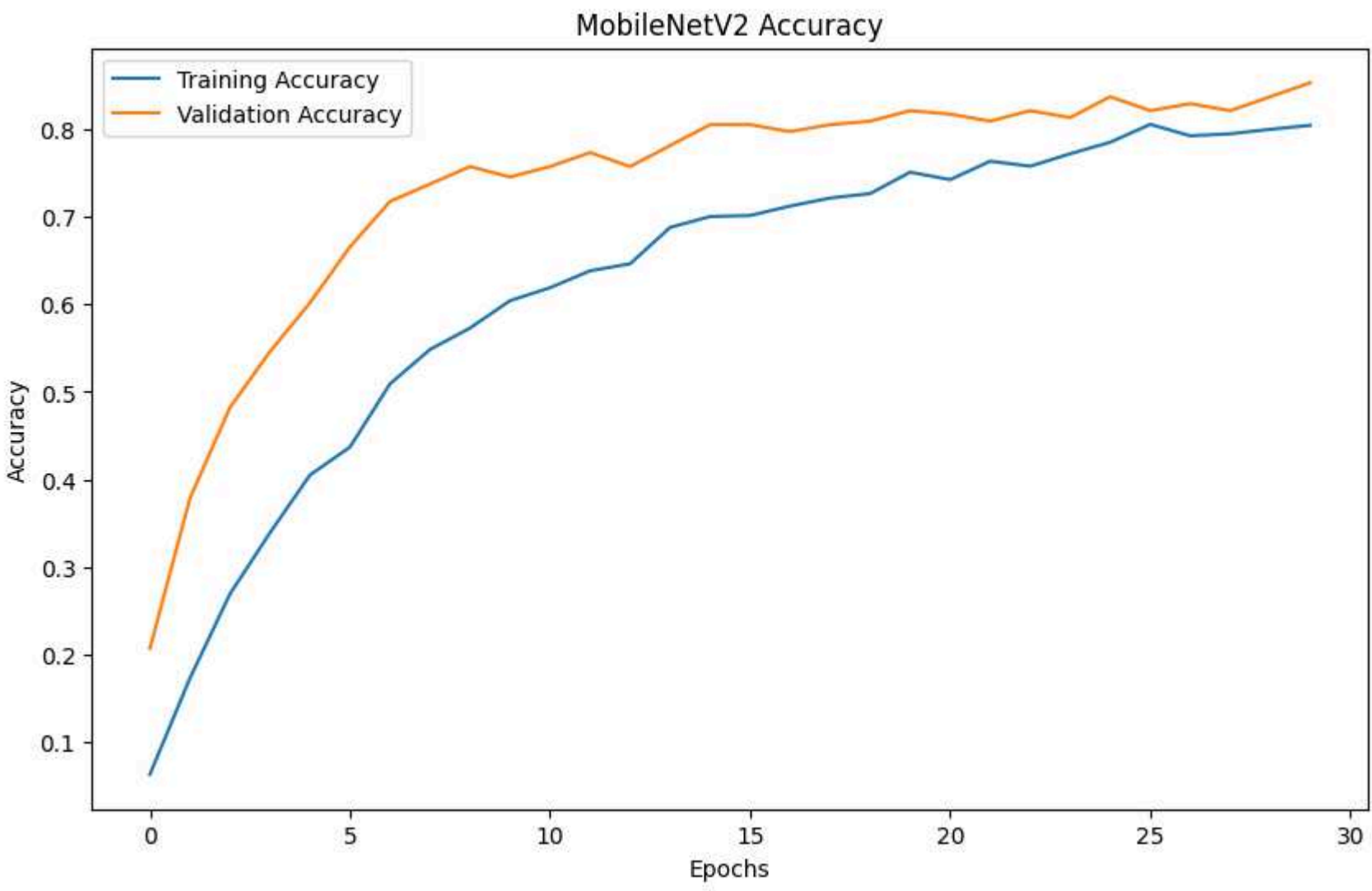
plt.figure(figsize=(10, 6))
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title("MobileNetV2 Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
In [22]: # Step 19: Plot training curves
# This step shows training and validation accuracy and loss across epochs.
```

```
plt.figure(figsize=(10, 6))
plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.title("MobileNetV2 Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title("MobileNetV2 Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



In [23]: # Step 20: Create Confusion Matrix
This step predicts test images and displays the confusion matrix.

```
test_generator.reset()

predictions = model.predict(test_generator)

y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(18, 16))
plt.imshow(cm)

plt.title("MobileNetV2 Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

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

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

plt.colorbar()

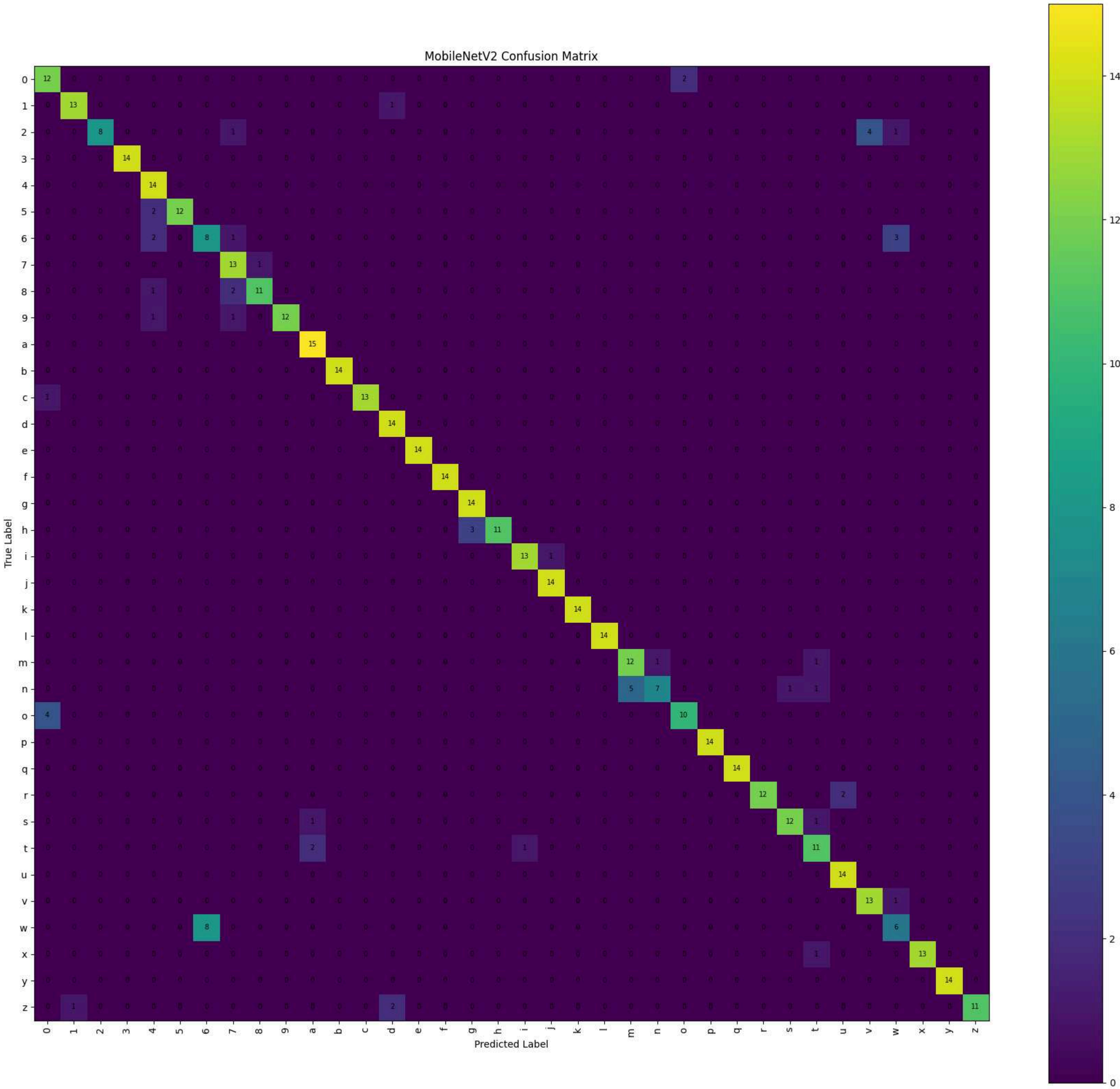
for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        plt.text(
            j,
            i,
            cm[i, j],
            ha="center",
            va="center",
            fontsize=7
        )

plt.tight_layout()
plt.show()

print("Test Accuracy:", accuracy_score(y_true, y_pred))

print("MobileNetV2 Classification Report")
print(classification_report(
    y_true,
    y_pred,
    target_names=CLASS_NAMES,
    zero_division=0
))
```

1m32/3200m 32m 0m 37m0m 1m26s0m 782ms/step



Test Accuracy: 0.8792079207920792				
MobileNetV2 Classification Report				
	precision	recall	f1-score	support
0	0.71	0.86	0.77	14
1	0.93	0.93	0.93	14
2	1.00	0.57	0.73	14
3	1.00	1.00	1.00	14
4	0.70	1.00	0.82	14
5	1.00	0.86	0.92	14
6	0.50	0.57	0.53	14
7	0.72	0.93	0.81	14
8	0.92	0.79	0.85	14
9	1.00	0.86	0.92	14
a	0.83	1.00	0.91	15
b	1.00	1.00	1.00	14
c	1.00	0.93	0.96	14
d	0.82	1.00	0.90	14
e	1.00	1.00	1.00	14
f	1.00	1.00	1.00	14
g	0.82	1.00	0.90	14
h	1.00	0.79	0.88	14
i	0.93	0.93	0.93	14

j	0.93	1.00	0.97	14
k	1.00	1.00	1.00	14
l	1.00	1.00	1.00	14
m	0.71	0.86	0.77	14
n	0.88	0.50	0.64	14
o	0.83	0.71	0.77	14
p	1.00	1.00	1.00	14
q	1.00	1.00	1.00	14
r	1.00	0.86	0.92	14
s	0.92	0.86	0.89	14
t	0.73	0.79	0.76	14
u	0.88	1.00	0.93	14
v	0.76	0.93	0.84	14
w	0.55	0.43	0.48	14
x	1.00	0.93	0.96	14
y	1.00	1.00	1.00	14
z	1.00	0.79	0.88	14
accuracy			0.88	505
macro avg	0.89	0.88	0.88	505
weighted avg	0.89	0.88	0.88	505

```
In [24]: # Step 21: Save the confusion matrix image
# This step saves the confusion matrix as a PNG file.

plt.figure(figsize=(18, 16))
plt.imshow(cm)

plt.title("MobileNetV2 Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

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

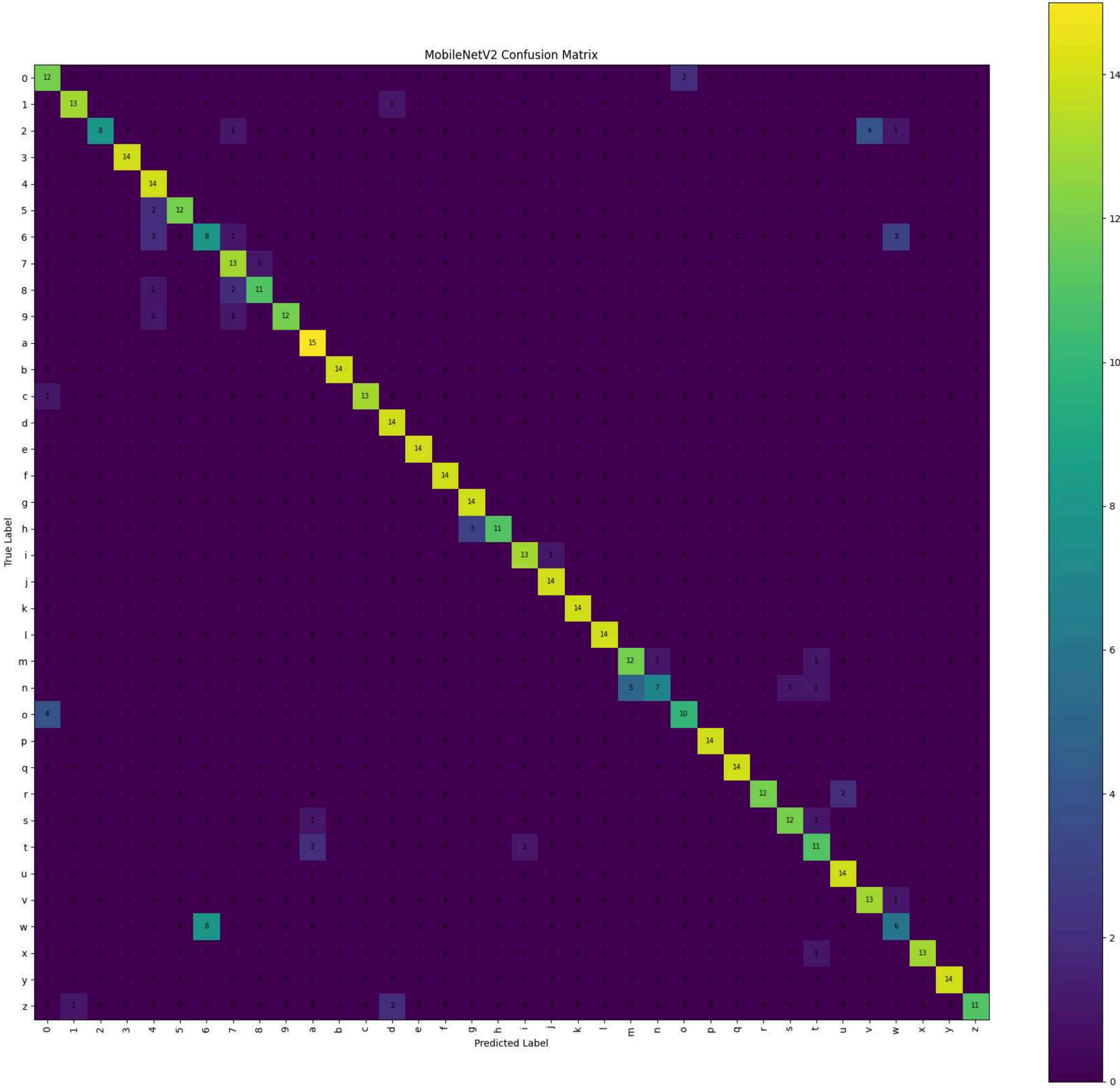
plt.yticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES
)

plt.colorbar()

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        plt.text(
            j,
            i,
            cm[i, j],
            ha="center",
            va="center",
            fontsize=7
        )

plt.tight_layout()
plt.savefig("mobilenetv2_confusion_matrix.png", dpi=300, bbox_inches="tight")
plt.show()

print("Confusion Matrix saved as: mobilenetv2_confusion_matrix.png")
```



Confusion Matrix saved as: mobilenetv2_confusion_matrix.png

```
In [25]: # Step 22: Save the trained model and class names
# This step saves the final MobileNetV2 model and class labels for later use.

model.save("final_mobilenetv2_asl_model.keras")

with open("asl_class_names.txt", "w") as file:
    for class_name in CLASS_NAMES:
        file.write(class_name + "\n")

print("Model saved as: final_mobilenetv2_asl_model.keras")
print("Class names saved as: asl_class_names.txt")

Model saved as: final_mobilenetv2_asl_model.keras
Class names saved as: asl_class_names.txt
```

```
from tensorflow.keras.preprocessing import image

def predict_single_image(image_path):
    img = image.load_img(image_path, target_size=(IMG_SIZE, IMG_SIZE))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    prediction = model.predict(img_array)

    predicted_index = np.argmax(prediction)
    predicted_class = CLASS_NAMES[predicted_index]
    confidence = np.max(prediction) * 100

    print("Predicted Class:", predicted_class)
    print(f"Confidence: {confidence:.2f}%")

    plt.imshow(image.load_img(image_path))
    plt.title(f"Prediction: {predicted_class.upper()} | Confidence: {confidence:.2f}%")
    plt.axis("off")
    plt.show()

    return predicted_class, confidence
```

```
!pip install gradio -q
```

```
import gradio as gr
from PIL import Image
import tensorflow as tf
```

```
import os
import shutil

SAVE_PATH = "/content/drive/MyDrive/ASL_MobileNet_Project"
os.makedirs(SAVE_PATH, exist_ok=True)

shutil.copy("final_mobilenetv2_asl_model.keras",
            os.path.join(SAVE_PATH, "final_mobilenetv2_asl_model.keras"))

shutil.copy("asl_class_names.txt",
            os.path.join(SAVE_PATH, "asl_class_names.txt"))

print("👍 Files copied to Google Drive successfully.")
```

```
import os
import tensorflow as tf

LOAD_PATH = "/content/drive/MyDrive/ASL_MobileNet_Project"

MODEL_PATH = os.path.join(LOAD_PATH, "final_mobilenetv2_asl_model.keras")
CLASS_PATH = os.path.join(LOAD_PATH, "asl_class_names.txt")

# Load MobileNetV2 model
model = tf.keras.models.load_model(MODEL_PATH)

# Load class names
with open(CLASS_PATH, "r") as file:
    CLASS_NAMES = [line.strip() for line in file.readlines()]

print("✅ MobileNetV2 model loaded successfully.")
print("Number of classes:", len(CLASS_NAMES))
print("Class Names:", CLASS_NAMES)
```

```
# Step 27: Create prediction function for the application
# This step prepares uploaded images and returns the predicted ASL class.

IMG_SIZE = 224

def predict_asl_mobilenet(image_input):
    if image_input is None:
        return "Please upload an image."

    img = Image.fromarray(image_input).convert("RGB")
    img = img.resize((IMG_SIZE, IMG_SIZE))

    img_array = np.array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    prediction = model.predict(img_array)

    predicted_index = np.argmax(prediction)
    predicted_class = CLASS_NAMES[predicted_index]
    confidence = np.max(prediction) * 100

    result = f"Predicted Class: {predicted_class.upper()}\nConfidence: {confidence:.2f}%"

    return result
```

```
# Step 28: Run the application
# This step launches a Gradio web application to test the MobileNetV2 ASL model.

app = gr.Interface(
    fn=predict_asl_mobilenet,
    inputs=gr.Image(type="numpy", label="Upload ASL Image"),
    outputs=gr.Textbox(label="Prediction Result"),
    title="ASL Recognition using MobileNetV2",
    description="Upload an ASL hand sign image and the MobileNetV2 model will predict the class."
)

app.launch(share=True)
```

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: <https://a22fad475a779f588a.gradio.live>

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run ``gradio deploy`` from the terminal in the working directory to deploy to Hugging Face Spaces (<https://huggingface.co/spaces>)

ASL Recognition using MobileNetV2

Upload an ASL hand sign image and the MobileNetV2 model will predict the class.