

```
In [6]: # 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')
```

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

```
In [7]: # Step 2: Import Required Libraries
# This step imports all required libraries for data processing, model training, evaluation, and prediction.
```

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

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

from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Dense,
    Dropout,
    GlobalAveragePooling2D,
    RandomRotation,
    RandomZoom,
    RandomTranslation,
    RandomContrast
)

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 EfficientNetB0
from tensorflow.keras.applications.efficientnet import preprocess_input

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

TensorFlow Version: 2.20.0

```
In [8]: # 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 [9]: # Step 4: Define Class Labels
# This step defines all letters and digits used for 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'
]

NUM_CLASSES = len(VALID_CLASSES)

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

Number of Classes: 36

```
In [10]: # Step 5: Create Dataset Split Directories
# This step creates separate folders for training, validation, and testing datasets.
```

```
SPLIT_DATASET_DIR = "/content/drive/MyDrive/ASL_Split_Dataset_EfficientNet"

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)

for folder in [train_dir, validation_dir, test_dir]:
    os.makedirs(folder, exist_ok=True)

print("Split dataset folders created successfully.")
```

Split dataset folders created successfully.

```
In [11]: # Step 6: Split the Dataset
# This step splits the dataset into training, validation, and testing sets.
```

```
train_ratio = 0.7
validation_ratio = 0.1
test_ratio = 0.2

def split_and_copy_files(class_dir, dest_base_dir):

    all_images = [
        os.path.join(class_dir, f)
        for f in os.listdir(class_dir)
        if os.path.isfile(os.path.join(class_dir, f))
        and f.lower().endswith(('.jpg', '.jpeg', '.png'))
    ]

    if len(all_images) == 0:
        return 0, 0, 0

    train_images, temp_images = train_test_split(
        all_images,
        test_size=(1 - train_ratio),
        random_state=42
    )

    validation_images, test_images = train_test_split(
        temp_images,
        test_size=test_ratio / (validation_ratio + test_ratio),
        random_state=42
    )

    for images, folder_name in zip(
        [train_images, validation_images, test_images],
        ["train", "validation", "test"]
    ):

        dest_dir = os.path.join(
            dest_base_dir,
            folder_name,
            os.path.basename(class_dir)
        )

        os.makedirs(dest_dir, exist_ok=True)

        for file in images:
            shutil.copy2(file, dest_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 = os.path.join(ORIGINAL_DATASET_DIR, class_name)

    if not os.path.isdir(class_dir):
        print("Missing class folder:", class_name)
        continue

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

    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=48, Validation=7, Test=15
1: Train=48, Validation=7, Test=15
2: Train=48, Validation=7, Test=15
3: Train=48, Validation=7, Test=15
4: Train=48, Validation=7, Test=15
5: Train=48, Validation=7, Test=15
6: Train=48, Validation=7, Test=15
7: Train=48, Validation=7, Test=15
8: Train=48, Validation=7, Test=15
9: Train=48, Validation=7, Test=15
a: Train=49, Validation=7, Test=15
b: Train=48, Validation=7, Test=15
c: Train=48, Validation=7, Test=15
d: Train=48, Validation=7, Test=15
e: Train=48, Validation=7, Test=15
f: Train=48, Validation=7, Test=15
g: Train=48, Validation=7, Test=15
h: Train=48, Validation=7, Test=15
i: Train=48, Validation=7, Test=15
j: Train=48, Validation=7, Test=15
k: Train=48, Validation=7, Test=15
l: Train=48, Validation=7, Test=15
m: Train=48, Validation=7, Test=15
n: Train=48, Validation=7, Test=15
o: Train=48, Validation=7, Test=15
p: Train=48, Validation=7, Test=15
q: Train=48, Validation=7, Test=15
r: Train=48, Validation=7, Test=15
s: Train=48, Validation=7, Test=15
t: Train=45, Validation=6, Test=14
u: Train=48, Validation=7, Test=15
v: Train=48, Validation=7, Test=15
w: Train=48, Validation=7, Test=15
x: Train=48, Validation=7, Test=15
y: Train=48, Validation=7, Test=15
z: Train=48, Validation=7, Test=15

Final Split:
Training Images: 1726
Validation Images: 251
Testing Images: 539
```

```
In [12]: # Step 7: Define Training Parameters
# This step defines the image size, batch size, number of 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 [13]: # Step 8: Create Image Data Generators
# This step creates image data generators for training, validation, and testing.
```

```
train_datagen = ImageDataGenerator()

val_test_datagen = ImageDataGenerator()
```

```
In [14]: # Step 9: Create the Training Data Generator
# This step loads the training images using the predefined class labels.
```

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

```
Found 1726 images belonging to 36 classes.
Train Images: 1726
Train Classes Count: 36
```

In [15]: # Step 10: Create the Validation Data Generator
This step loads the validation images using the predefined class labels.

```
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 Count:", validation_generator.num_classes)
```

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

In [16]: # Step 11: Create the Test Data Generator
This step loads the test images and retrieves the class names.

```
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 = train_generator.num_classes
```

```
print("Class Names:", CLASS_NAMES)
print("Number of Classes:", NUM_CLASSES)
print("Train Images:", train_generator.n)
print("Validation Images:", validation_generator.n)
print("Test Images:", test_generator.n)
```

Found 539 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
Train Images: 1726
Validation Images: 251
Test Images: 539

In [17]: # Step 12: Create the Data Augmentation Layer
This step creates a data augmentation layer to improve model generalization.

```
transform_layer = tf.keras.Sequential([
    RandomRotation(0.1),
    RandomZoom(0.1),
    RandomTranslation(0.1, 0.1),
    RandomContrast(0.1)
], name="Transform_Layer")
```

In [18]: # Step 13: Build the EfficientNetB0 Base Model
This step loads EfficientNetB0 as a pre-trained feature extractor.

```
base_model = EfficientNetB0(
    weights="imagenet",
    include_top=False,
    input_shape=(IMG_SIZE, IMG_SIZE, 3)
)
```

base_model.trainable = False

Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
[1m16705208/16705208[0m [32m-----[0m [0m[37m[0m [0m [1m0s[0m [0m 0us/step

In [19]: # Step 14: Build the Classification Model
This step adds the classification layers on top of EfficientNetB0.

```
inputs = tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

x = transform_layer(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)

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

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

model = Model(
    inputs=inputs,
    outputs=output,
    name="EfficientNetB0_AS_L_Model"
)
```

model.summary()

Model: "EfficientNetB0_AS_L_Model"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
Transform_Layer (Sequential)	(None, 224, 224, 3)	0
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,049,571
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dense (Dense)	(None, 256)	327,936
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 36)	9,252

Total params: 4,386,759 (16.73 MB)

Trainable params: 337,188 (1.29 MB)

Non-trainable params: 4,049,571 (15.45 MB)

In [20]: # Step 15: Compile the Model
This step configures the optimizer, loss function, and evaluation metric.

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

In [21]: # Step 16: Configure Training Callbacks
This step defines callbacks for early stopping, learning rate reduction, and saving the best model.

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

```
In [22]: # Step 17: Train the Model
# This step trains the EfficientNetB0 model using the training and validation datasets.
```

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

```
Epoch 1/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.0389 - loss: 3.6111
Epoch 1: val_accuracy improved from None to 0.11554, saving model to best_efficientnet_asl.keras

Epoch 1: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m177s][0m 2s/step - accuracy: 0.0655 - loss: 3.5067 - val_accuracy: 0.1155 - val_loss: 3.2887 - learning_rate: 1.0000e-04
Epoch 2/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.1395 - loss: 3.2594
Epoch 2: val_accuracy improved from 0.11554 to 0.25896, saving model to best_efficientnet_asl.keras

Epoch 2: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m159s][0m 1s/step - accuracy: 0.1518 - loss: 3.1943 - val_accuracy: 0.2590 - val_loss: 2.9869 - learning_rate: 1.0000e-04
Epoch 3/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.2218 - loss: 2.9428
Epoch 3: val_accuracy improved from 0.25896 to 0.35458, saving model to best_efficientnet_asl.keras

Epoch 3: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m152s][0m 1s/step - accuracy: 0.2346 - loss: 2.8889 - val_accuracy: 0.3546 - val_loss: 2.7091 - learning_rate: 1.0000e-04
Epoch 4/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.2747 - loss: 2.7113
Epoch 4: val_accuracy improved from 0.35458 to 0.40637, saving model to best_efficientnet_asl.keras

Epoch 4: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m151s][0m 1s/step - accuracy: 0.2926 - loss: 2.6409 - val_accuracy: 0.4064 - val_loss: 2.4708 - learning_rate: 1.0000e-04
Epoch 5/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.3327 - loss: 2.4667
Epoch 5: val_accuracy improved from 0.40637 to 0.47012, saving model to best_efficientnet_asl.keras

Epoch 5: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m159s][0m 1s/step - accuracy: 0.3401 - loss: 2.4241 - val_accuracy: 0.4701 - val_loss: 2.2536 - learning_rate: 1.0000e-04
Epoch 6/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.3710 - loss: 2.2746
Epoch 6: val_accuracy improved from 0.47012 to 0.49402, saving model to best_efficientnet_asl.keras

Epoch 6: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m152s][0m 1s/step - accuracy: 0.3830 - loss: 2.2314 - val_accuracy: 0.4940 - val_loss: 2.0981 - learning_rate: 1.0000e-04
Epoch 7/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.4280 - loss: 2.1101
Epoch 7: val_accuracy improved from 0.49402 to 0.51793, saving model to best_efficientnet_asl.keras

Epoch 7: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m147s][0m 1s/step - accuracy: 0.4299 - loss: 2.0873 - val_accuracy: 0.5179 - val_loss: 1.9585 - learning_rate: 1.0000e-04
Epoch 8/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.4428 - loss: 1.9520
Epoch 8: val_accuracy improved from 0.51793 to 0.57371, saving model to best_efficientnet_asl.keras

Epoch 8: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m153s][0m 1s/step - accuracy: 0.4409 - loss: 1.9661 - val_accuracy: 0.5737 - val_loss: 1.8259 - learning_rate: 1.0000e-04
Epoch 9/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.4749 - loss: 1.8202
Epoch 9: val_accuracy did not improve from 0.57371
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m154s][0m 1s/step - accuracy: 0.4919 - loss: 1.7913 - val_accuracy: 0.5697 - val_loss: 1.7274 - learning_rate: 1.0000e-04
Epoch 10/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.5119 - loss: 1.7268
Epoch 10: val_accuracy did not improve from 0.57371
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m152s][0m 1s/step - accuracy: 0.5243 - loss: 1.7142 - val_accuracy: 0.5737 - val_loss: 1.6756 - learning_rate: 1.0000e-04
Epoch 11/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.5009 - loss: 1.6833
Epoch 11: val_accuracy improved from 0.57371 to 0.58566, saving model to best_efficientnet_asl.keras

Epoch 11: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m151s][0m 1s/step - accuracy: 0.5220 - loss: 1.6460 - val_accuracy: 0.5857 - val_loss: 1.5911 - learning_rate: 1.0000e-04
Epoch 12/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.5463 - loss: 1.5840
Epoch 12: val_accuracy did not improve from 0.58566
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m152s][0m 1s/step - accuracy: 0.5556 - loss: 1.5542 - val_accuracy: 0.5259 - val_loss: 1.5587 - learning_rate: 1.0000e-04
Epoch 13/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.5408 - loss: 1.5434
Epoch 13: val_accuracy improved from 0.58566 to 0.59363, saving model to best_efficientnet_asl.keras

Epoch 13: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m156s][0m 1s/step - accuracy: 0.5504 - loss: 1.4961 - val_accuracy: 0.5936 - val_loss: 1.4759 - learning_rate: 1.0000e-04
Epoch 14/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.5739 - loss: 1.4849
Epoch 14: val_accuracy improved from 0.59363 to 0.59761, saving model to best_efficientnet_asl.keras

Epoch 14: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m153s][0m 1s/step - accuracy: 0.5886 - loss: 1.4283 - val_accuracy: 0.5976 - val_loss: 1.4270 - learning_rate: 1.0000e-04
Epoch 15/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.5882 - loss: 1.3933
Epoch 15: val_accuracy improved from 0.59761 to 0.60558, saving model to best_efficientnet_asl.keras

Epoch 15: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m152s][0m 1s/step - accuracy: 0.6060 - loss: 1.3715 - val_accuracy: 0.6056 - val_loss: 1.3631 - learning_rate: 1.0000e-04
Epoch 16/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.6220 - loss: 1.3335
Epoch 16: val_accuracy improved from 0.60558 to 0.63745, saving model to best_efficientnet_asl.keras

Epoch 16: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m154s][0m 1s/step - accuracy: 0.6315 - loss: 1.3266 - val_accuracy: 0.6375 - val_loss: 1.3150 - learning_rate: 1.0000e-04
Epoch 17/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.6357 - loss: 1.2456
Epoch 17: val_accuracy did not improve from 0.63745
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m146s][0m 1s/step - accuracy: 0.6379 - loss: 1.2496 - val_accuracy: 0.6375 - val_loss: 1.3152 - learning_rate: 1.0000e-04
Epoch 18/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.6414 - loss: 1.2165
Epoch 18: val_accuracy did not improve from 0.63745
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m204s][0m 1s/step - accuracy: 0.6275 - loss: 1.2388 - val_accuracy: 0.6135 - val_loss: 1.2964 - learning_rate: 1.0000e-04
Epoch 19/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.6350 - loss: 1.2049
Epoch 19: val_accuracy improved from 0.63745 to 0.64940, saving model to best_efficientnet_asl.keras

Epoch 19: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m149s][0m 1s/step - accuracy: 0.6454 - loss: 1.1829 - val_accuracy: 0.6494 - val_loss: 1.2463 - learning_rate: 1.0000e-04
Epoch 20/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.6719 - loss: 1.1089
Epoch 20: val_accuracy did not improve from 0.64940
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m147s][0m 1s/step - accuracy: 0.6703 - loss: 1.1297 - val_accuracy: 0.6494 - val_loss: 1.1858 - learning_rate: 1.0000e-04
Epoch 21/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.6880 - loss: 1.1033
Epoch 21: val_accuracy improved from 0.64940 to 0.66135, saving model to best_efficientnet_asl.keras

Epoch 21: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m152s][0m 1s/step - accuracy: 0.6715 - loss: 1.1207 - val_accuracy: 0.6614 - val_loss: 1.1598 - learning_rate: 1.0000e-04
Epoch 22/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.6858 - loss: 1.0798
Epoch 22: val_accuracy improved from 0.66135 to 0.67331, saving model to best_efficientnet_asl.keras

Epoch 22: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m154s][0m 1s/step - accuracy: 0.6819 - loss: 1.0963 - val_accuracy: 0.6733 - val_loss: 1.1267 - learning_rate: 1.0000e-04
Epoch 23/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.6737 - loss: 1.1057
Epoch 23: val_accuracy improved from 0.67331 to 0.67729, saving model to best_efficientnet_asl.keras

Epoch 23: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m148s][0m 1s/step - accuracy: 0.6819 - loss: 1.0738 - val_accuracy: 0.6773 - val_loss: 1.1284 - learning_rate: 1.0000e-04
Epoch 24/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.7074 - loss: 0.9929
Epoch 24: val_accuracy improved from 0.67729 to 0.68526, saving model to best_efficientnet_asl.keras

Epoch 24: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m147s][0m 1s/step - accuracy: 0.6999 - loss: 1.0250 - val_accuracy: 0.6853 - val_loss: 1.0862 - learning_rate: 1.0000e-04
Epoch 25/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.7247 - loss: 0.9819
Epoch 25: val_accuracy improved from 0.68526 to 0.70518, saving model to best_efficientnet_asl.keras

Epoch 25: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m158s][0m 1s/step - accuracy: 0.7063 - loss: 1.0194 - val_accuracy: 0.7052 - val_loss: 1.0717 - learning_rate: 1.0000e-04
Epoch 26/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.7098 - loss: 0.9998
Epoch 26: val_accuracy did not improve from 0.70518
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m153s][0m 1s/step - accuracy: 0.7167 - loss: 0.9953 - val_accuracy: 0.6853 - val_loss: 1.0884 - learning_rate: 1.0000e-04
Epoch 27/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.7079 - loss: 0.9758
Epoch 27: val_accuracy improved from 0.70518 to 0.71315, saving model to best_efficientnet_asl.keras

Epoch 27: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m157s][0m 1s/step - accuracy: 0.7207 - loss: 0.9458 - val_accuracy: 0.7131 - val_loss: 1.0239 - learning_rate: 1.0000e-04
Epoch 28/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.7225 - loss: 0.9344
Epoch 28: val_accuracy did not improve from 0.71315
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m152s][0m 1s/step - accuracy: 0.7167 - loss: 0.9238 - val_accuracy: 0.6853 - val_loss: 1.0282 - learning_rate: 1.0000e-04
Epoch 29/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.7186 - loss: 0.9455
Epoch 29: val_accuracy did not improve from 0.71315
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m158s][0m 1s/step - accuracy: 0.7375 - loss: 0.9194 - val_accuracy: 0.6972 - val_loss: 1.0022 - learning_rate: 1.0000e-04
Epoch 30/30
[1m108/108][0m 0/32m-----[0m0m[37m0[0m 0[1m0s][0m 1s/step - accuracy: 0.7189 - loss: 0.9476
Epoch 30: val_accuracy improved from 0.71315 to 0.71713, saving model to best_efficientnet_asl.keras
```

Epoch 30: finished saving model to best_efficientnet_asl.keras
[1m108/108][0m @32m [0m [37m[0m [1m156s[0m 1s/step - accuracy: 0.7121 - loss: 0.9398 - val_accuracy: 0.7171 - val_loss: 0.9717 - learning_rate: 1.0000e-04
Restoring model weights from the end of the best epoch: 30.

```
In [23]: # Step 18: Evaluate the Model on the Validation Set
# This step evaluates the model performance using the validation dataset.

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

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

Validation Accuracy: 71.71%
Validation Loss: 0.9717

```
In [24]: # Step 19: Evaluate the Model on the Test Set
# This step evaluates the final model performance using the test dataset.

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

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

Test Accuracy: 73.65%
Test Loss: 0.9495

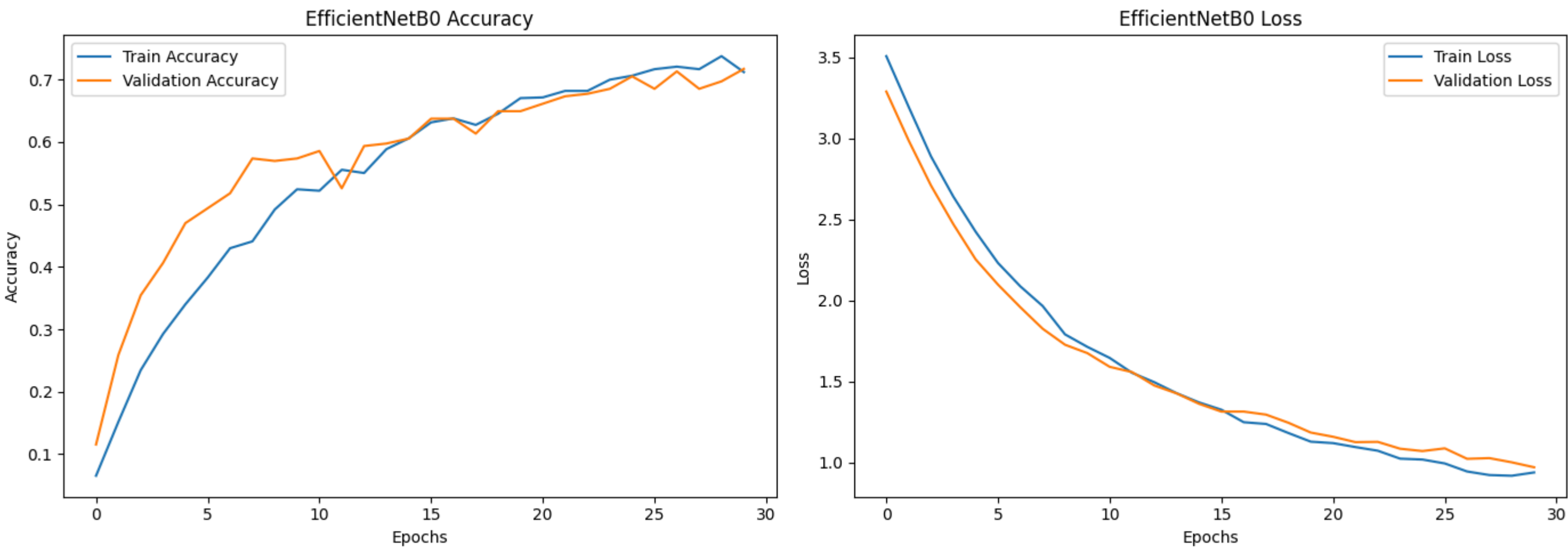
```
In [25]: # Step 20: Plot Training Performance
# This step plots the training and validation accuracy and loss curves.
```

```
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.title("EfficientNetB0 Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title("EfficientNetB0 Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

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



```
In [26]: # Step 21: Generate Predictions on the Test Set
# This step generates predictions and calculates the overall test accuracy.
```

```
test_generator.reset()

predictions = model.predict(test_generator)

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

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

[1m34/34][0m @32m [0m [37m[0m [1m44s[0m 1s/step
Test Accuracy: 0.7365491651205937

```
In [27]: # Step 22: Generate the Classification Report
# This step displays the precision, recall, F1-score, and accuracy for each class.
```

```
print("EfficientNetB0 Classification Report")

print(
    classification_report(
        y_true,
        y_pred,
        target_names=CLASS_NAMES,
        zero_division=0
    )
)
```

EfficientNetB0 Classification Report				
	precision	recall	f1-score	support
0	1.00	0.33	0.50	15
1	1.00	0.67	0.80	15
2	0.80	0.53	0.64	15
3	0.75	1.00	0.86	15
4	0.71	0.80	0.75	15
5	1.00	0.60	0.75	15
6	0.60	0.20	0.30	15
7	0.37	0.47	0.41	15
8	0.38	1.00	0.55	15
9	0.65	0.87	0.74	15
a	0.83	1.00	0.91	15
b	1.00	1.00	1.00	15
c	1.00	1.00	1.00	15
d	0.70	0.47	0.56	15
e	0.54	1.00	0.70	15
f	1.00	1.00	1.00	15
g	1.00	0.87	0.93	15
h	0.88	1.00	0.94	15
i	0.35	1.00	0.52	15
j	1.00	0.80	0.89	15
k	1.00	0.53	0.70	15
l	1.00	0.80	0.89	15
m	0.52	0.73	0.61	15
n	0.62	0.53	0.57	15
o	0.60	1.00	0.75	15
p	1.00	1.00	1.00	15
q	1.00	1.00	1.00	15
r	1.00	0.33	0.50	15
s	1.00	0.60	0.75	15
t	0.82	0.64	0.72	14
u	0.92	0.80	0.86	15
v	0.57	0.27	0.36	15
w	1.00	0.20	0.33	15
x	0.93	0.87	0.90	15
y	0.83	1.00	0.91	15
z	1.00	0.60	0.75	15
accuracy			0.74	539
macro avg	0.82	0.74	0.73	539
weighted avg	0.82	0.74	0.73	539

```
In [28]: # Step 23: Generate the Confusion Matrix
# This step visualizes the classification errors between the predicted and true labels.

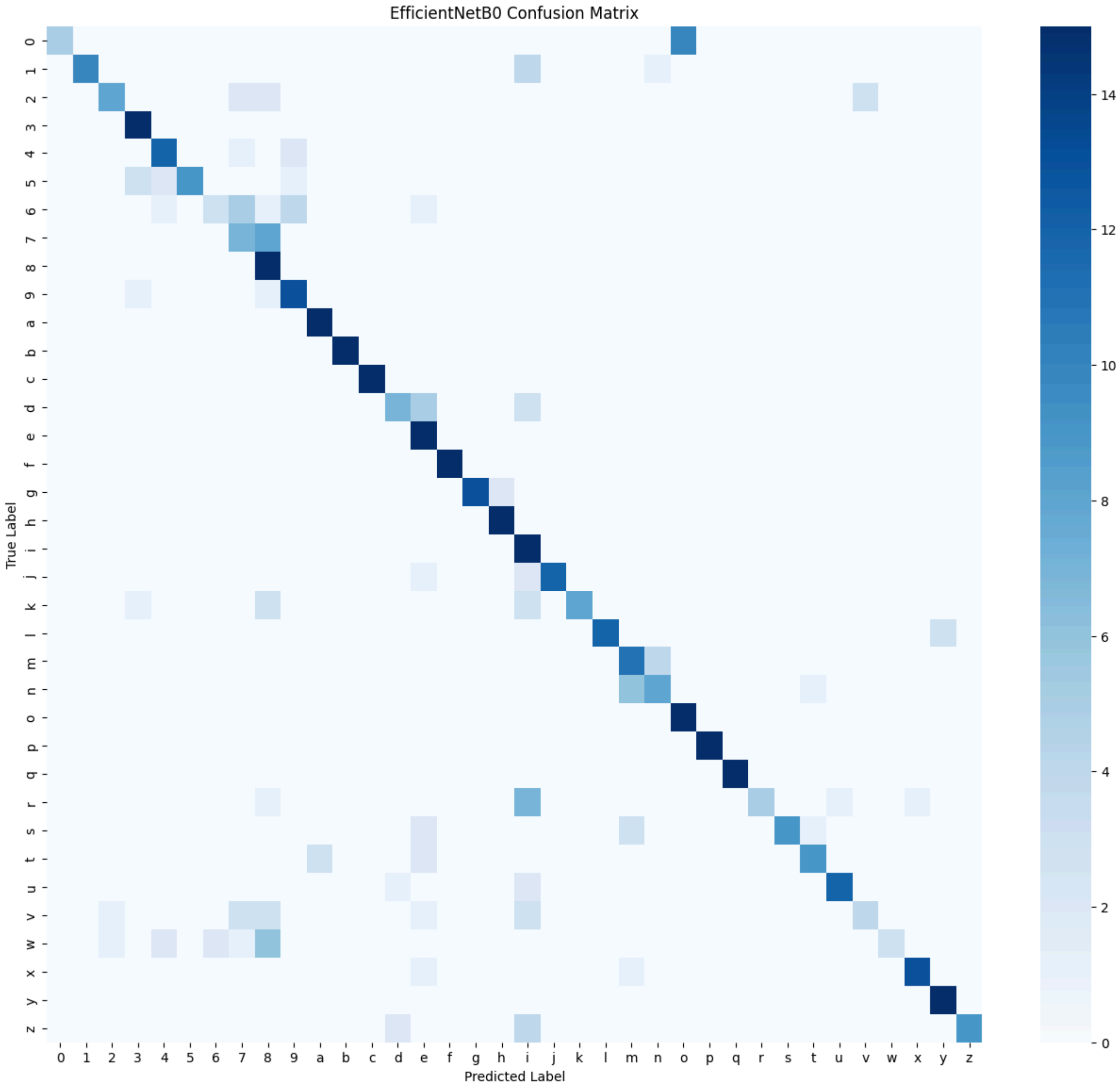
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(16, 14))

sns.heatmap(
    cm,
    annot=False,
    xticklabels=CLASS_NAMES,
    yticklabels=CLASS_NAMES,
    cmap="Blues",
    fmt="d"
)

plt.title("EfficientNetB0 Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

plt.show()
```



```
In [29]: # Step 24: Save the Trained Model and Class Names to Google Drive
# This step saves the final EfficientNetB0 model and class labels for future use without retraining.

import os

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

model.save(os.path.join(SAVE_PATH, "final_efficientnet_asl_model.keras"))

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("EfficientNetB0 model saved successfully.")
print("Class names saved successfully.")

EfficientNetB0 model saved successfully.
Class names saved successfully.
```

```
In [30]: # Step 26: Install Gradio
# This step installs Gradio to create a web interface for image prediction.

!pip install gradio -q
```

```
In [31]: # Step 27: Import Libraries for the Web Interface
# This step imports the required libraries for image prediction and the Gradio interface.

import gradio as gr
import numpy as np
import tensorflow as tf

from PIL import Image
import matplotlib.pyplot as plt
```

```
In [32]: # Step 28: Load the Saved Model and Class Names from Google Drive
# This step loads the trained EfficientNetB0 model and class labels without retraining.

import os
import tensorflow as tf

LOAD_PATH = "/content/drive/MyDrive/ASL_EfficientNet_Project"

model = tf.keras.models.load_model(
    os.path.join(LOAD_PATH, "final_efficientnet_asl_model.keras")
)

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

print("EfficientNetB0 model loaded successfully.")
print("Number of Classes:", len(CLASS_NAMES))
print("Class Names:", CLASS_NAMES)

EfficientNetB0 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 [33]: # Step 30: Create the Prediction Function
# This step prepares the uploaded image and returns the predicted class with confidence.

IMG_SIZE = 224

def predict_asl(image):

    if image is None:
        return "No image uploaded"

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

    image_array = np.array(image)
    image_array = np.expand_dims(image_array, axis=0)

    prediction = model.predict(image_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 [34]: # Step 31: Launch the Gradio Web Interface
# This step creates the web interface for ASL image prediction.
```

```
interface = gr.Interface(
    fn=predict_asl,

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

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

    title="ASL Recognition using EfficientNetB0",

    description="Upload an ASL hand sign image to predict the corresponding class.",

    live=False
)
```

```
interface.launch(share=True)
```

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: <https://357da62032791dfdce.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 EfficientNetB0

Upload an ASL hand sign image to 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 ⚙️