

```
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%
```

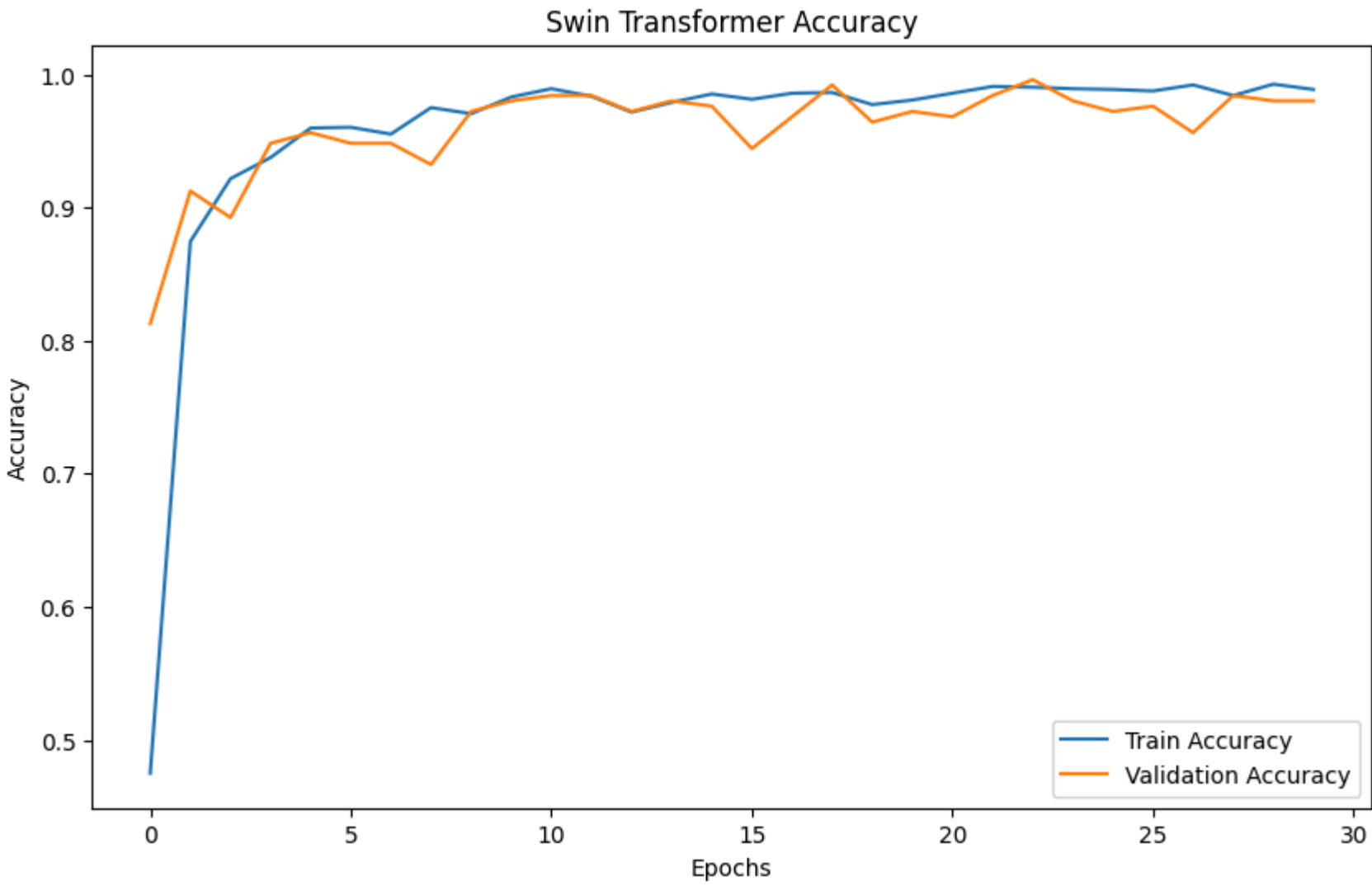
```
In [20]: # Step 20: Plot Accuracy Curves
# This step plots training and validation accuracy across epochs.

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()
```



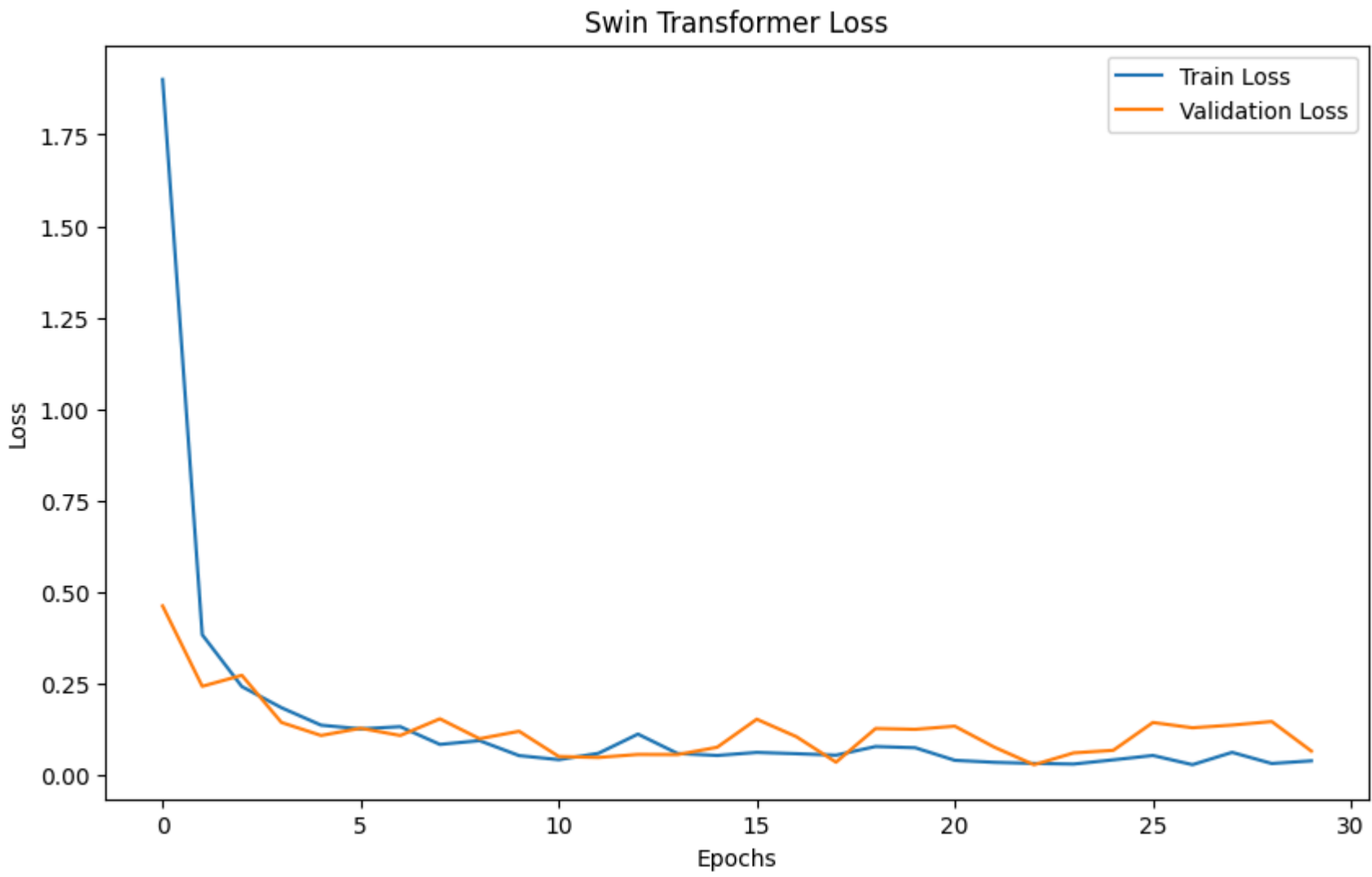
```
In [21]: # Step 21: Plot Loss Curves
# This step plots training and validation loss across epochs.

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")

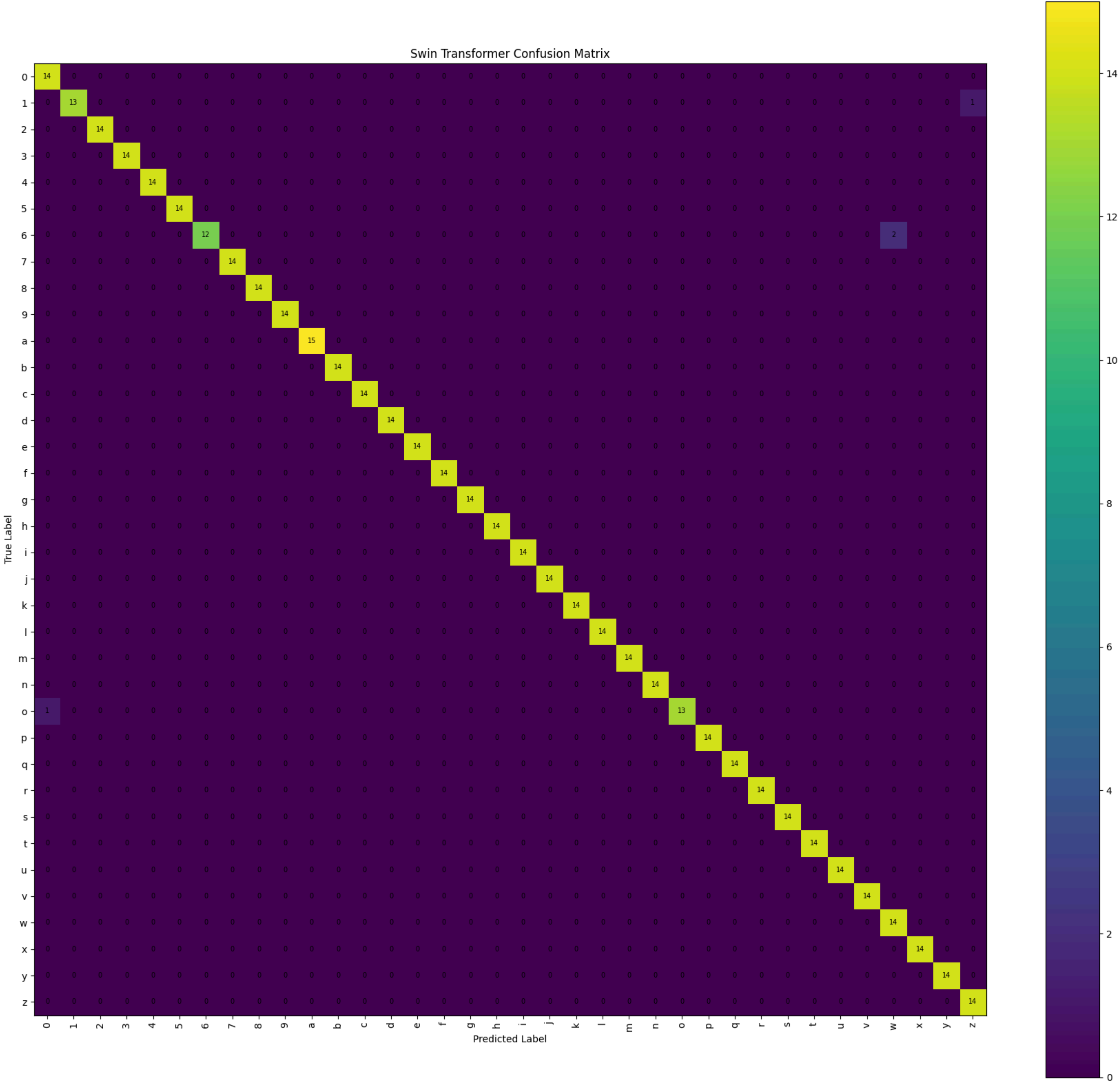
plt.xticks(
    ticks=np.arange(len(CLASS_NAMES)),
    labels=CLASS_NAMES,
    rotation=90
)

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
)

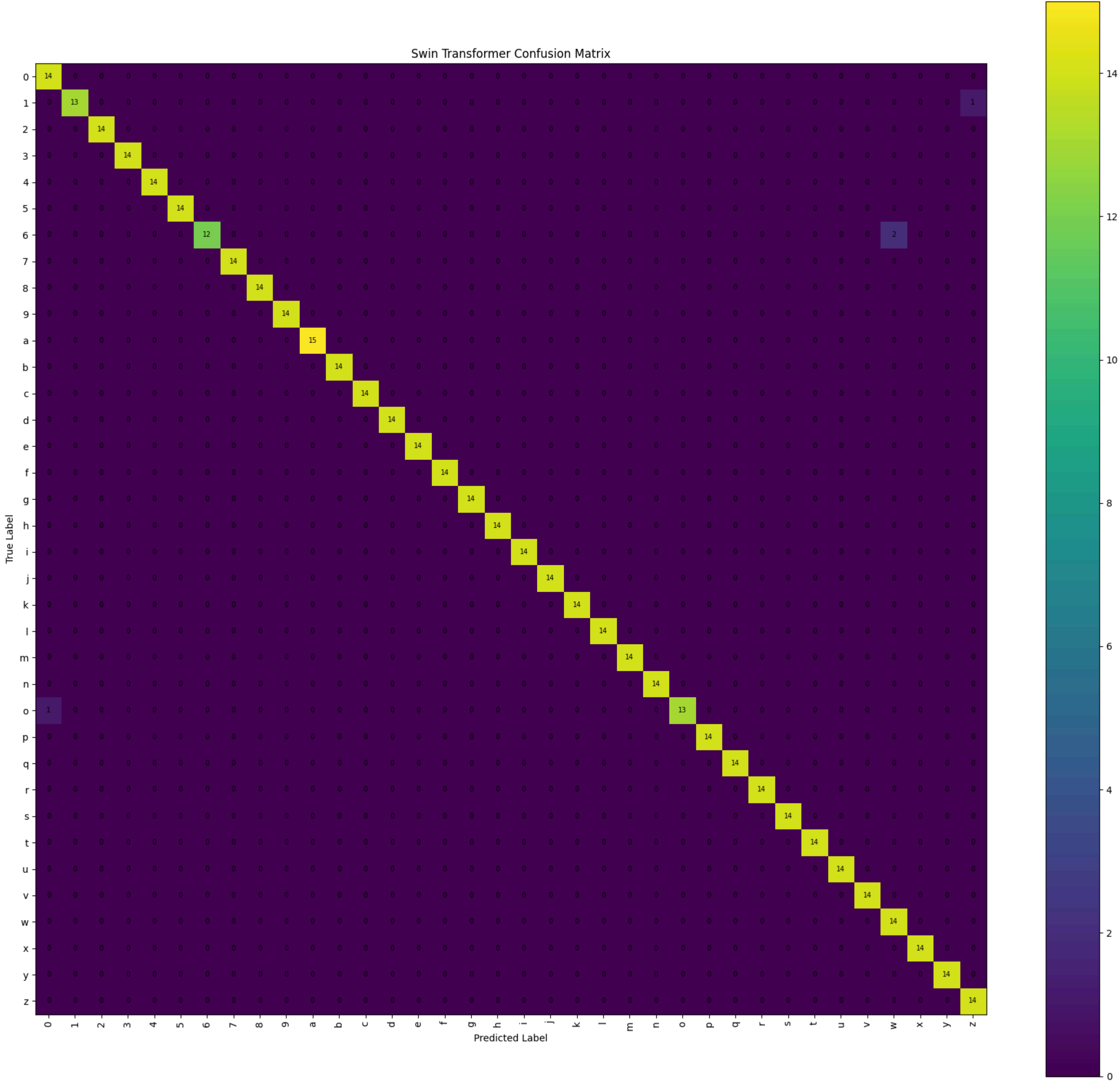
plt.colorbar()

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