

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

from google.colab import drive

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

Mounted at /content/drive

```
In [2]: # =====
# Step 2: Install Required Libraries
# This step installs all additional libraries required for
# Vision Transformer training and spelling correction.
# =====

!pip -q install timm
!pip -q install textblob
```

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

import os
import random
import shutil
import time
import warnings

import numpy as np
import matplotlib.pyplot as plt

from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim

from torch.utils.data import DataLoader

from torchvision import datasets
from torchvision import transforms

import timm

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

from textblob import TextBlob

warnings.filterwarnings("ignore")

print("=" * 60)
print("PyTorch Version :", torch.__version__)
print("TIMM Version    :", timm.__version__)
print("=" * 60)
```

```
=====
PyTorch Version : 2.11.0+cu128
TIMM Version    : 1.0.27
=====
```

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

SEED = 42

random.seed(SEED)
np.random.seed(SEED)

torch.manual_seed(SEED)

if torch.cuda.is_available():
    torch.cuda.manual_seed(SEED)
    torch.cuda.manual_seed_all(SEED)

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

print("=" * 60)
print("Project Configuration")
print("=" * 60)
print(f"Random Seed : {SEED}")
print(f"Device      : {DEVICE}")

if torch.cuda.is_available():
    print(
        "GPU Name    :",
        torch.cuda.get_device_name(0)
    )

print("=" * 60)
```

```
=====
Project Configuration
=====
Random Seed : 42
Device      : cuda
GPU Name    : NVIDIA A100-SXM4-40GB
=====
```

```
In [5]: # =====
# Step 5: Define Dataset Path
# This step locates the dataset directory inside Google Drive
# and verifies that all class folders are available.
# =====

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

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

if os.path.isdir(DATASET_DIR):

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

    print("Dataset Status : Found")
    print(f"Dataset Path   : {DATASET_DIR}")
    print(f"Total Classes  : {len(class_names)}")

else:

    raise FileNotFoundError(
        f"Dataset not found:\n{DATASET_DIR}"
    )

print("=" * 60)
```

```
=====
Dataset Configuration
=====
Dataset Status : Found
Dataset Path   : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Original/asl_dataset
Total Classes  : 36
=====
```

In [6]:

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

EXPECTED_CLASSES = [
    "0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
    "a", "b", "c", "d", "e", "f", "g", "h", "i", "j",
    "k", "l", "m", "n", "o", "p", "q", "r", "s", "t",
    "u", "v", "w", "x", "y", "z"
]

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

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

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

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

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

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

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

In [7]:

```
# =====
# Step 7: Load Final Dataset Split
# This step loads the same fixed Train, Validation, and Test
# split used by all models for a fair comparison.
# =====

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

TRAIN_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "train"
)

VALIDATION_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "val"
)

TEST_DIR = os.path.join(
    SPLIT_DATASET_DIR,
    "test"
)

required_paths = [
    TRAIN_DIR,
    VALIDATION_DIR,
    TEST_DIR
]

if not all(
    os.path.isdir(path)
    for path in required_paths
):
    raise FileNotFoundError(
        "The final dataset split was not found."
    )

print("=" * 60)
print("Final Dataset Split Loaded Successfully")
print("=" * 60)
print(f"Training Path      : {TRAIN_DIR}")
print(f"Validation Path    : {VALIDATION_DIR}")
print(f"Testing Path       : {TEST_DIR}")
print("=" * 60)
```

```
=====
Final Dataset Split Loaded Successfully
=====
Training Path      : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/train
Validation Path    : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/val
Testing Path       : /content/drive/MyDrive/ASL_Final_Project/01_Dataset/Dataset_Split_Final/test
=====
```

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

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

split_paths = {
    "Training": TRAIN_DIR,
    "Validation": VALIDATION_DIR,
    "Testing": TEST_DIR
}

grand_total = 0

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

for split_name, split_path in split_paths.items():

    total_images = 0

    for class_name in os.listdir(split_path):

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

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

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

    grand_total += total_images

    print(
        f"{split_name:<12}: "
        f"{total_images} images"
    )

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

```
=====
Dataset Split Summary
=====
Training    : 1760 images
Validation  : 251 images
Testing     : 504 images
-----
Total Images : 2515
=====
```

```
In [9]: # =====
# Step 9: Create Image Transformations
# This step applies Vision Transformer preprocessing and data
# augmentation to the Training, Validation, and Testing images.
# =====

IMAGE_SIZE = 224
BATCH_SIZE = 16

train_transform = transforms.Compose([
    transforms.Resize(
        (IMAGE_SIZE, IMAGE_SIZE)
    ),

    transforms.RandomRotation(15),

    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]
    )
])

validation_test_transform = transforms.Compose([
    transforms.Resize(
        (IMAGE_SIZE, IMAGE_SIZE)
    ),

    transforms.ToTensor(),

    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

print("=" * 60)
print("Image Transformations Created Successfully")
print("=" * 60)
print(f"Image Size : {IMAGE_SIZE} x {IMAGE_SIZE}")
print(f"Batch Size : {BATCH_SIZE}")
print("=" * 60)
```

```
=====
Image Transformations Created Successfully
=====
Image Size : 224 x 224
Batch Size : 16
=====
```

```
In [10]: # =====
# Step 10: Load Datasets and Create DataLoaders
# This step loads the Training, Validation, and Testing
# datasets and creates DataLoaders for model training.
# =====

train_dataset = datasets.ImageFolder(
    root=TRAIN_DIR,
    transform=train_transform
)

validation_dataset = datasets.ImageFolder(
    root=VALIDATION_DIR,
    transform=validation_test_transform
)

test_dataset = datasets.ImageFolder(
    root=TEST_DIR,
    transform=validation_test_transform
)

CLASS_NAMES = train_dataset.classes
NUM_CLASSES = len(CLASS_NAMES)

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

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

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

print("=" * 60)
print("Datasets and DataLoaders Created Successfully")
print("=" * 60)
print(f"Training Samples   : {len(train_dataset)}")
print(f"Validation Samples : {len(validation_dataset)}")
print(f"Testing Samples    : {len(test_dataset)}")
print(f"Number of Classes   : {NUM_CLASSES}")
print(f"Training Batches    : {len(train_loader)}")
print(f"Validation Batches  : {len(validation_loader)}")
print(f"Testing Batches     : {len(test_loader)}")
print("=" * 60)
print("Class Names")
print("=" * 60)

for class_name in CLASS_NAMES:
    print(class_name)

print("=" * 60)
```

```
=====
Datasets and DataLoaders Created Successfully
=====
Training Samples   : 1760
Validation Samples : 251
Testing Samples    : 504
Number of Classes  : 36
Training Batches   : 110
Validation Batches : 16
Testing Batches    : 32
=====
Class Names
=====
0
1
2
3
4
5
6
7
8
9
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
=====
```

```
In [11]: # =====
# Step 11: Build the Vision Transformer Model
# This step loads the pretrained Vision Transformer model
# and replaces its classification head for 36 ASL classes.
# =====

MODEL_NAME = "vit_base_patch16_224"

model = timm.create_model(
    MODEL_NAME,
    pretrained=True,
    num_classes=NUM_CLASSES
)

model = model.to(DEVICE)

print("=" * 60)
print("Vision Transformer Model Created Successfully")
print("=" * 60)
print(f"Model Name       : {MODEL_NAME}")
print(f"Number of Classes : {NUM_CLASSES}")
print(f"Total Parameters  : {sum(p.numel() for p in model.parameters()):,}")
print(f"Device           : {DEVICE}")
print("=" * 60)
```

Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
WARNING:huggingface_hub.utils._http:Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.

```
model.safetensors:   0% |          | 0.00/346M [00:00<?, ?B/s]

=====
Vision Transformer Model Created Successfully
=====
Model Name       : vit_base_patch16_224
Number of Classes : 36
Total Parameters  : 85,826,340
Device           : cuda
=====
```

In [12]:

```
# =====
# Step 12: Configure the Vision Transformer Model
# This step defines the loss function, optimizer, learning
# rate scheduler, and model saving path.
# =====

LEARNING_RATE = 0.0001

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

os.makedirs(
    MODEL_DIR,
    exist_ok=True
)

BEST_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "best_vit_asl_model.pth"
)

criterion = nn.CrossEntropyLoss()

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

scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode="min",
    factor=0.3,
    patience=4,
    min_lr=1e-6
)

print("=" * 60)
print("Training Setup Configured Successfully")
print("=" * 60)
print("Optimizer      : AdamW")
print(f"Learning Rate   : {LEARNING_RATE}")
print("Loss Function   : Cross Entropy Loss")
print("Weight Decay    : 0.01")
print(f"Best Model Path: {BEST_MODEL_PATH}")
print("=" * 60)
```

```
=====
Training Setup Configured Successfully
=====
Optimizer      : AdamW
Learning Rate   : 0.0001
Loss Function   : Cross Entropy Loss
Weight Decay    : 0.01
Best Model Path: /content/drive/MyDrive/ASL_Final_Project/02_Models/VisionTransformer/best_vit_asl_model.pth
=====
```

```
In [13]: # =====
# Step 13: Create Training and Evaluation Functions
# This step creates the functions used to train and evaluate
# the Vision Transformer model during each epoch.
# =====

def train_one_epoch(
    model,
    data_loader,
    criterion,
    optimizer,
    device
):
    model.train()

    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    for images, labels in data_loader:

        images = images.to(
            device,
            non_blocking=True
        )

        labels = labels.to(
            device,
            non_blocking=True
        )

        optimizer.zero_grad()

        outputs = model(images)

        loss = criterion(
            outputs,
            labels
        )

        loss.backward()

        optimizer.step()

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

        predicted_labels = outputs.argmax(
            dim=1
        )

        correct_predictions += (
            predicted_labels == labels
        ).sum().item()

        total_samples += labels.size(0)

    epoch_loss = (
        running_loss / total_samples
    )

    epoch_accuracy = (
        correct_predictions / total_samples
    )

    return epoch_loss, epoch_accuracy

def evaluate_model(
    model,
    data_loader,
    criterion,
    device
):
    model.eval()

    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    true_labels = []
    predicted_labels = []

    with torch.no_grad():

        for images, labels in data_loader:

            images = images.to(
                device,
                non_blocking=True
            )

            labels = labels.to(
                device,
                non_blocking=True
            )

            outputs = model(images)

            loss = criterion(
                outputs,
                labels
            )

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

            predictions = outputs.argmax(
                dim=1
            )

            correct_predictions += (
                predictions == labels
            ).sum().item()

            total_samples += labels.size(0)

            true_labels.extend(
                labels.cpu().numpy()
            )

            predicted_labels.extend(
                predictions.cpu().numpy()
            )

    epoch_loss = (
        running_loss / total_samples
    )

    epoch_accuracy = (
        correct_predictions / total_samples
    )

    return (
        epoch_loss,
        epoch_accuracy,
        true_labels,
        predicted_labels
    )

print("=" * 60)
print("Training and Evaluation Functions Created Successfully")
print("=" * 60)
```

```
=====
Training and Evaluation Functions Created Successfully
=====
```

In [14]:

```
# =====
# Step 14: Train the Vision Transformer Model
# This step trains the model, monitors validation performance,
# applies Early Stopping, and saves the best model.
# =====

EPOCHS = 30
EARLY_STOPPING_PATIENCE = 7

train_losses = []
validation_losses = []

train_accuracies = []
validation_accuracies = []

best_validation_accuracy = 0.0
best_validation_loss = float("inf")

early_stopping_counter = 0

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
    )

    scheduler.step(
        validation_loss
    )

    train_losses.append(
        train_loss
    )

    validation_losses.append(
        validation_loss
    )

    train_accuracies.append(
        train_accuracy
    )

    validation_accuracies.append(
        validation_accuracy
    )

    current_learning_rate = (
        optimizer.param_groups[0]["lr"]
    )

    print("=" * 60)
    print(f"Epoch [{epoch + 1}/{EPOCHS}]")
    print("=" * 60)
    print(f"Training Loss      : {train_loss:.4f}")
    print(f"Training Accuracy    : {train_accuracy:.4f}")
    print(f"Validation Loss       : {validation_loss:.4f}")
    print(f"Validation Accuracy  : {validation_accuracy:.4f}")
    print(f"Learning Rate        : {current_learning_rate:.8f}")

    if validation_accuracy > best_validation_accuracy:

        best_validation_accuracy = (
            validation_accuracy
        )

        torch.save(
            model.state_dict(),
            BEST_MODEL_PATH
        )

        print("Best model saved successfully.")

    if validation_loss < best_validation_loss:

        best_validation_loss = (
            validation_loss
        )

        early_stopping_counter = 0

    else:

        early_stopping_counter += 1

        print(
            "Early Stopping Counter : "
            f"{early_stopping_counter}/"
            f"{EARLY_STOPPING_PATIENCE}"
        )

    if (
        early_stopping_counter
        >= EARLY_STOPPING_PATIENCE
    ):

        print("=" * 60)
        print("Early Stopping Activated")
        print("=" * 60)

        break

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

model = model.to(DEVICE)

print("=" * 60)
print("Model Training Completed Successfully")
print("=" * 60)
print(
    f"Best Validation Accuracy : "
    f"{best_validation_accuracy:.4f}"
)
print("Best model weights restored successfully.")
print("=" * 60)
```

```
=====
Epoch [1/30]
=====
Training Loss      : 3.4832
Training Accuracy  : 0.0716
Validation Loss    : 2.6513
Validation Accuracy : 0.2829
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [2/30]
=====
Training Loss      : 1.2265
Training Accuracy  : 0.5881
Validation Loss    : 0.5569
Validation Accuracy : 0.7968
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [3/30]
=====
Training Loss      : 0.4696
Training Accuracy  : 0.8426
Validation Loss    : 0.3135
Validation Accuracy : 0.9044
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [4/30]
=====
Training Loss      : 0.2991
Training Accuracy  : 0.8989
Validation Loss    : 0.2440
Validation Accuracy : 0.9163
```

```
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [5/30]
=====
Training Loss      : 0.1811
Training Accuracy   : 0.9364
Validation Loss     : 0.1838
Validation Accuracy : 0.9203
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [6/30]
=====
Training Loss      : 0.1534
Training Accuracy   : 0.9517
Validation Loss     : 0.1842
Validation Accuracy : 0.9442
Learning Rate      : 0.00010000
Best model saved successfully.
Early Stopping Counter : 1/7
=====
Epoch [7/30]
=====
Training Loss      : 0.1719
Training Accuracy   : 0.9443
Validation Loss     : 0.3328
Validation Accuracy : 0.9203
Learning Rate      : 0.00010000
Early Stopping Counter : 2/7
=====
Epoch [8/30]
=====
Training Loss      : 0.1679
Training Accuracy   : 0.9443
Validation Loss     : 0.1623
Validation Accuracy : 0.9522
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [9/30]
=====
Training Loss      : 0.1065
Training Accuracy   : 0.9693
Validation Loss     : 0.1709
Validation Accuracy : 0.9442
Learning Rate      : 0.00010000
Early Stopping Counter : 1/7
=====
Epoch [10/30]
=====
Training Loss      : 0.1067
Training Accuracy   : 0.9608
Validation Loss     : 0.2084
Validation Accuracy : 0.9402
Learning Rate      : 0.00010000
Early Stopping Counter : 2/7
=====
Epoch [11/30]
=====
Training Loss      : 0.0861
Training Accuracy   : 0.9699
Validation Loss     : 0.1340
Validation Accuracy : 0.9522
Learning Rate      : 0.00010000
=====
Epoch [12/30]
=====
Training Loss      : 0.0894
Training Accuracy   : 0.9744
Validation Loss     : 0.0986
Validation Accuracy : 0.9602
Learning Rate      : 0.00010000
Best model saved successfully.
=====
Epoch [13/30]
=====
Training Loss      : 0.0357
Training Accuracy   : 0.9864
Validation Loss     : 0.1351
Validation Accuracy : 0.9522
Learning Rate      : 0.00010000
Early Stopping Counter : 1/7
=====
Epoch [14/30]
=====
Training Loss      : 0.1299
Training Accuracy   : 0.9563
Validation Loss     : 0.1437
Validation Accuracy : 0.9442
Learning Rate      : 0.00010000
Early Stopping Counter : 2/7
=====
Epoch [15/30]
=====
Training Loss      : 0.1426
Training Accuracy   : 0.9545
Validation Loss     : 0.1770
Validation Accuracy : 0.9562
Learning Rate      : 0.00010000
Early Stopping Counter : 3/7
=====
Epoch [16/30]
=====
Training Loss      : 0.0861
Training Accuracy   : 0.9790
Validation Loss     : 0.1600
Validation Accuracy : 0.9562
Learning Rate      : 0.00010000
Early Stopping Counter : 4/7
=====
Epoch [17/30]
=====
Training Loss      : 0.0571
Training Accuracy   : 0.9824
Validation Loss     : 0.2132
Validation Accuracy : 0.9323
Learning Rate      : 0.00003000
Early Stopping Counter : 5/7
=====
Epoch [18/30]
=====
Training Loss      : 0.0158
Training Accuracy   : 0.9949
Validation Loss     : 0.1022
Validation Accuracy : 0.9681
Learning Rate      : 0.00003000
Best model saved successfully.
Early Stopping Counter : 6/7
=====
Epoch [19/30]
=====
Training Loss      : 0.0118
Training Accuracy   : 0.9955
Validation Loss     : 0.0721
Validation Accuracy : 0.9801
Learning Rate      : 0.00003000
Best model saved successfully.
=====
Epoch [20/30]
=====
Training Loss      : 0.0060
Training Accuracy   : 0.9989
Validation Loss     : 0.0596
Validation Accuracy : 0.9841
Learning Rate      : 0.00003000
Best model saved successfully.
=====
Epoch [21/30]
=====
Training Loss      : 0.0011
Training Accuracy   : 1.0000
Validation Loss     : 0.0548
Validation Accuracy : 0.9801
Learning Rate      : 0.00003000
=====
Epoch [22/30]
=====
Training Loss      : 0.0021
Training Accuracy   : 0.9994
Validation Loss     : 0.0783
Validation Accuracy : 0.9841
Learning Rate      : 0.00003000
Early Stopping Counter : 1/7
=====
Epoch [23/30]
=====
Training Loss      : 0.0032
Training Accuracy   : 0.9989
Validation Loss     : 0.0878
```

```
Validation Accuracy : 0.9841
Learning Rate       : 0.00003000
Early Stopping Counter : 2/7
=====
Epoch [24/30]
=====
Training Loss       : 0.0066
Training Accuracy   : 0.9977
Validation Loss     : 0.0570
Validation Accuracy : 0.9880
Learning Rate       : 0.00003000
Best model saved successfully.
Early Stopping Counter : 3/7
=====
Epoch [25/30]
=====
Training Loss       : 0.0028
Training Accuracy   : 0.9994
Validation Loss     : 0.0661
Validation Accuracy : 0.9880
Learning Rate       : 0.00003000
Early Stopping Counter : 4/7
=====
Epoch [26/30]
=====
Training Loss       : 0.0075
Training Accuracy   : 0.9977
Validation Loss     : 0.0942
Validation Accuracy : 0.9681
Learning Rate       : 0.00000900
Early Stopping Counter : 5/7
=====
Epoch [27/30]
=====
Training Loss       : 0.0024
Training Accuracy   : 0.9994
Validation Loss     : 0.0620
Validation Accuracy : 0.9841
Learning Rate       : 0.00000900
Early Stopping Counter : 6/7
=====
Epoch [28/30]
=====
Training Loss       : 0.0006
Training Accuracy   : 1.0000
Validation Loss     : 0.0601
Validation Accuracy : 0.9841
Learning Rate       : 0.00000900
Early Stopping Counter : 7/7
=====
Early Stopping Activated
=====
Model Training Completed Successfully
=====
Best Validation Accuracy : 0.9880
Best model weights restored successfully.
=====
```

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

(
    test_loss,
    test_accuracy,
    y_true,
    y_pred
) = evaluate_model(
    model,
    test_loader,
    criterion,
    DEVICE
)

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

```
=====
Model Evaluation
=====
Test Loss      : 0.0427
Test Accuracy : 0.9881
=====
```

```
In [16]: # =====
# Step 16: Generate Predictions and Classification Report
# This step displays the precision, recall, F1-score,
# and support values for all ASL classes.
# =====

report = classification_report(
    y_true,
    y_pred,
    target_names=CLASS_NAMES,
    digits=4,
    zero_division=0
)

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

```
=====
Vision Transformer Classification Report
=====
```

	precision	recall	f1-score	support
0	0.8462	0.7857	0.8148	14
1	1.0000	1.0000	1.0000	14
2	1.0000	1.0000	1.0000	14
3	1.0000	1.0000	1.0000	14
4	1.0000	1.0000	1.0000	14
5	1.0000	1.0000	1.0000	14
6	1.0000	0.9286	0.9630	14
7	1.0000	1.0000	1.0000	14
8	1.0000	1.0000	1.0000	14
9	1.0000	1.0000	1.0000	14
a	1.0000	1.0000	1.0000	14
b	1.0000	1.0000	1.0000	14
c	1.0000	1.0000	1.0000	14
d	1.0000	1.0000	1.0000	14
e	1.0000	1.0000	1.0000	14
f	1.0000	1.0000	1.0000	14
g	1.0000	1.0000	1.0000	14
h	1.0000	1.0000	1.0000	14
i	1.0000	1.0000	1.0000	14
j	1.0000	1.0000	1.0000	14
k	1.0000	1.0000	1.0000	14
l	1.0000	1.0000	1.0000	14
m	1.0000	1.0000	1.0000	14
n	1.0000	1.0000	1.0000	14
o	0.8000	0.8571	0.8276	14
p	1.0000	1.0000	1.0000	14
q	1.0000	1.0000	1.0000	14
r	1.0000	1.0000	1.0000	14
s	1.0000	1.0000	1.0000	14
t	1.0000	1.0000	1.0000	14
u	1.0000	1.0000	1.0000	14
v	1.0000	1.0000	1.0000	14
w	0.9333	1.0000	0.9655	14
x	1.0000	1.0000	1.0000	14
y	1.0000	1.0000	1.0000	14
z	1.0000	1.0000	1.0000	14
accuracy			0.9881	504
macro avg	0.9883	0.9881	0.9881	504
weighted avg	0.9883	0.9881	0.9881	504

```
=====
```

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

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

cm = confusion_matrix(
    y_true,
    y_pred
)

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

plt.imshow(
    cm,
    cmap="Blues"
)

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

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

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

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

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

plt.colorbar()

threshold = cm.max() / 2

for i in range(len(CLASS_NAMES)):
    for j in range(len(CLASS_NAMES)):
        text_color = (
            "white"
            if cm[i, j] > threshold
            else "#0B3D91"
        )

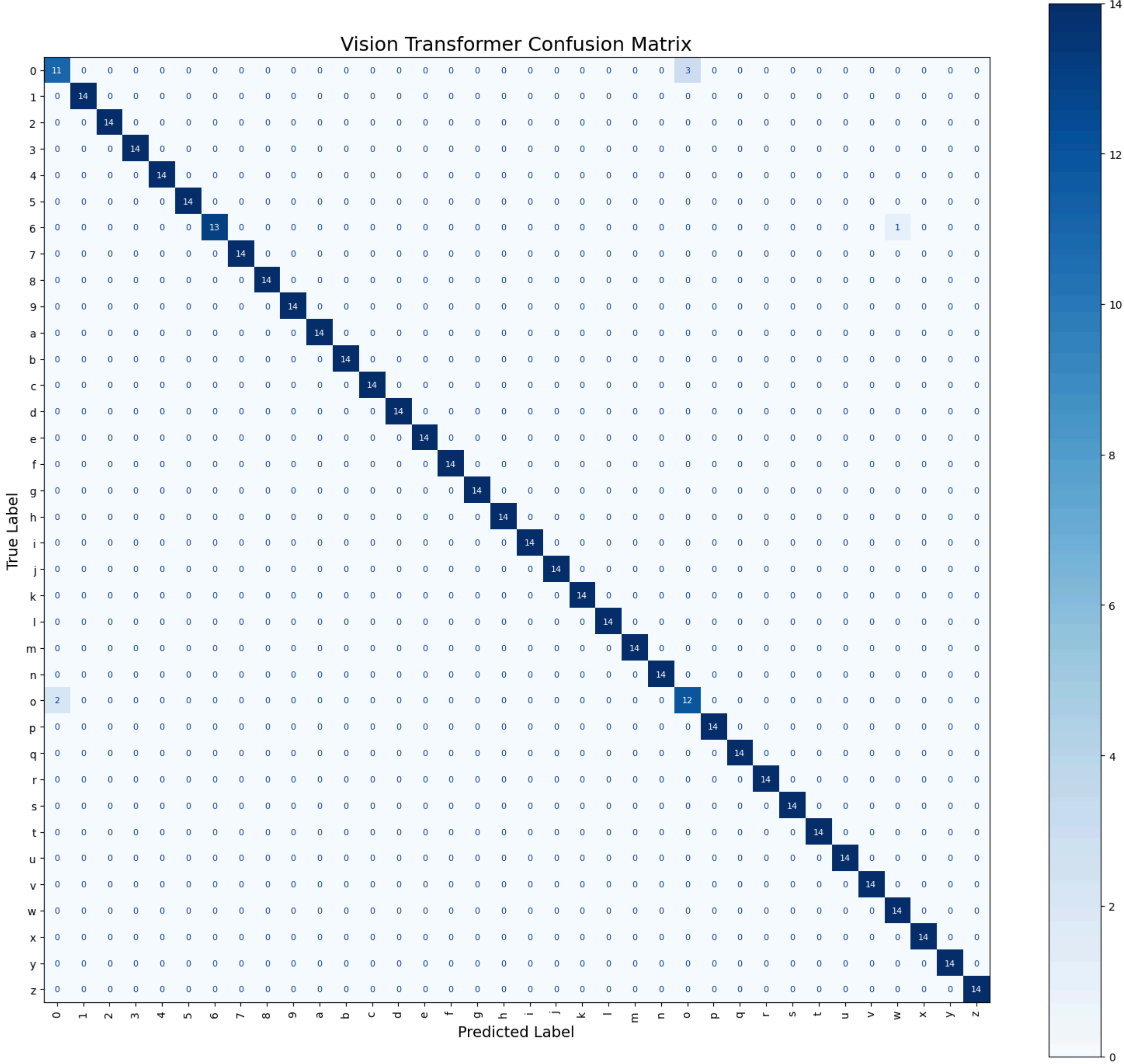
        plt.text(
            j,
            i,
            str(cm[i, j]),
            ha="center",
            va="center",
            fontsize=8,
            color=text_color
        )

plt.tight_layout()

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

plt.show()

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



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

In [18]:

```
# =====
# Step 18: Plot Training History
# This step plots the Training and Validation Accuracy and
# Loss curves and saves them as high-quality images.
# =====

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

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

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

plt.plot(
    train_accuracies,
    label="Training Accuracy"
)

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

plt.title(
    "Vision Transformer Accuracy Curve"
)

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

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

plt.show()

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

plt.plot(
    train_losses,
    label="Training Loss"
)

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

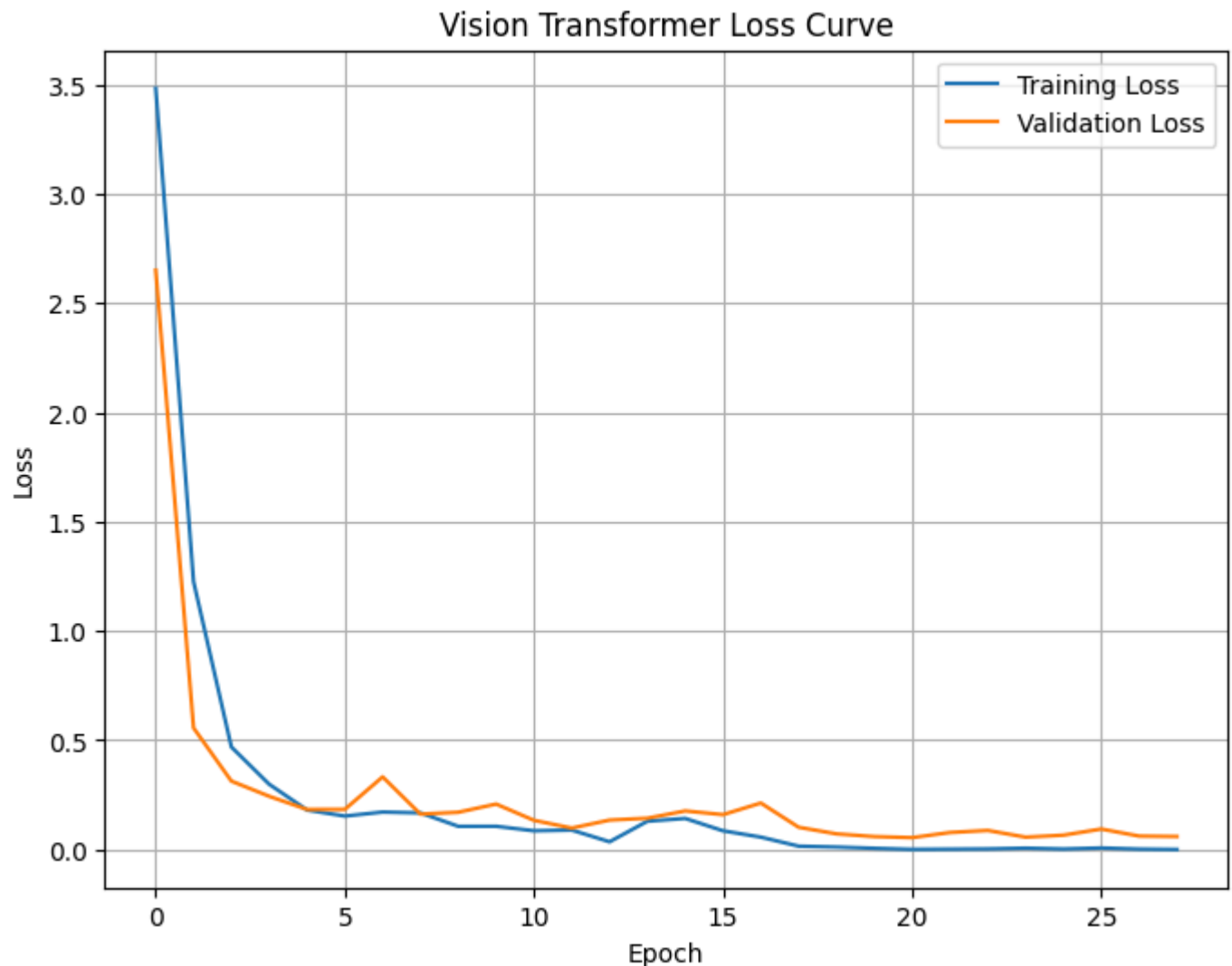
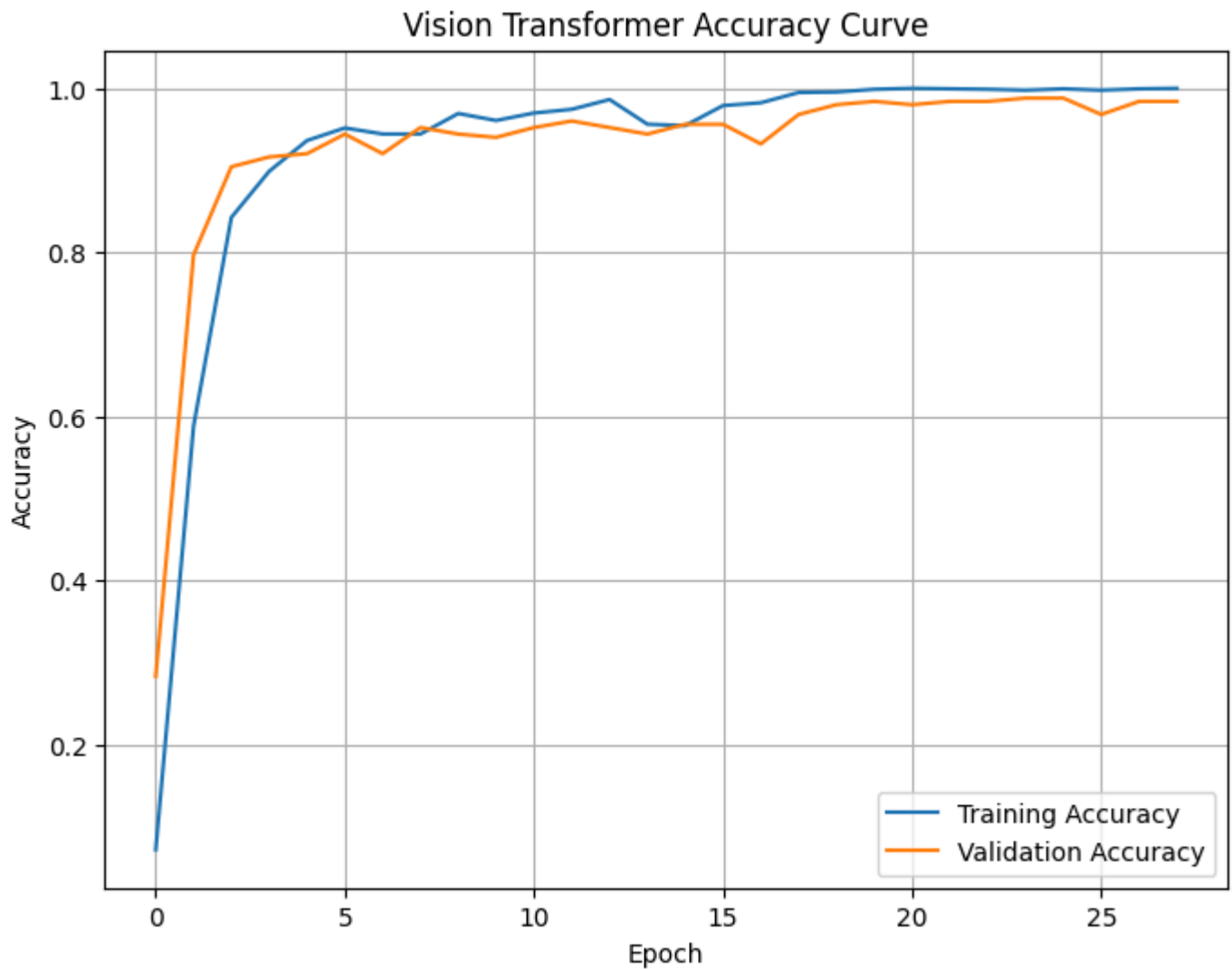
plt.title(
    "Vision Transformer Loss Curve"
)

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

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

plt.show()

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



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

In [19]:

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

FINAL_MODEL_PATH = os.path.join(
    MODEL_DIR,
    "final_vit_asl_model.pth"
)

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

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

with open(
    CLASS_NAMES_PATH,
    "w"
) as file:

    for class_name in CLASS_NAMES:
        file.write(
            f"{class_name}\n"
        )

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

=====
Model Saved Successfully
=====

Final Model : /content/drive/MyDrive/ASL_Final_Project/02_Models/VisionTransformer/final_vit_asl_model.pth
Class Names : /content/drive/MyDrive/ASL_Final_Project/02_Models/VisionTransformer/vit_class_names.txt
=====
```

In [20]:

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

INFERENCE_REPEATS = 3

model.eval()

# Warm-up run
with torch.no_grad():

    for images, _ in test_loader:

        images = images.to(
            DEVICE,
            non_blocking=True
        )

        _ = model(images)

    break

if torch.cuda.is_available():
    torch.cuda.synchronize()

inference_times = []

for repeat in range(INFERENCE_REPEATS):

    start_time = time.perf_counter()

    with torch.no_grad():

        for images, _ in test_loader:

            images = images.to(
                DEVICE,
                non_blocking=True
            )

            _ = model(images)

        if torch.cuda.is_available():
            torch.cuda.synchronize()

        end_time = time.perf_counter()

        inference_times.append(
            end_time - start_time
        )

average_inference_time = np.mean(
    inference_times
)

std_inference_time = np.std(
    inference_times
)

average_time_per_image = (
    average_inference_time
    / len(test_dataset)
)

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

=====
Standardized Inference Time
=====
Model           : Vision Transformer
Test Images     : 504
Repeated Measurements : 3
Average Total Time   : 3.8516 seconds
Standard Deviation   : 3.1241 seconds
Average Time per Image : 0.007642 seconds
=====
```

In [21]:

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

import os
import json

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

MODEL_KEY = "vit"
MODEL_DISPLAY_NAME = "Vision Transformer"

FINAL_SAVED_MODEL_PATH = (
    "/content/drive/MyDrive/ASL_Final_Project/"
    "02_Models/VisionTransformer/final_vit_asl_model.pth"
)

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

os.makedirs(METRICS_DIR, exist_ok=True)

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

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

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

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

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

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

gpu_name = (
    torch.cuda.get_device_name(0)
    if torch.cuda.is_available()
    else "CPU"
)

total_parameters = sum(
    parameter.numel()
    for parameter in model.parameters()
)

metrics_data = {
    "model": MODEL_DISPLAY_NAME,
    "framework": "PyTorch",
    "gpu": gpu_name,
    "test_images": int(len(test_dataset)),
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "weighted_precision": float(weighted_precision),
    "weighted_recall": float(weighted_recall),
    "weighted_f1_score": float(weighted_f1),
    "macro_f1_score": float(macro_f1),
    "average_inference_time_seconds": float(
        average_inference_time
    ),
    "inference_standard_deviation_seconds": float(
        std_inference_time
    ),
    "average_time_per_image_seconds": float(
        average_time_per_image
    ),
    "parameters": int(total_parameters),
    "model_size_mb": float(model_size_mb),
    "model_path": FINAL_SAVED_MODEL_PATH
}

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

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

=====
Vision Transformer Metrics Saved Successfully
=====
Accuracy      : 0.9881
Precision     : 0.9883
Recall        : 0.9881
F1-Score      : 0.9881
Model Size    : 327.46 MB
Parameters    : 85,826,340
Metrics File  : /content/drive/MyDrive/ASL_Final_Project/03_Results/Metrics/vit_metrics.json
=====
```