Install necessary libraries for model training, data handling, and augmentation

- torch and torchvision: PyTorch framework and its vision-related functionalities
- timm: A library for working with pretrained models
- tqdm: A progress bar library for loops
- split-folders: A utility for splitting datasets into train/validation/test sets

```
!pip install --upgrade torch torchvision timm tqdm split-folders
albumentations

Requirement already satisfied: torch in
/usr/local/lib/python3.10/dist-packages (2.4.1+cu121)
Collecting torch
  Using cached torch-2.5.1-cp310-cp310-manylinux1_x86_64.whl.metadata
(28 kB)
Requirement already satisfied: torchvision in
/usr/local/lib/python3.10/dist-packages (0.19.1+cu121)
Collecting torchvision
  Using cached torchvision-0.20.1-cp310-cp310-
manylinux1_x86_64.whl.metadata (6.1 kB)
Collecting timm
  Using cached timm-1.0.11-py3-none-any.whl.metadata (48 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (4.66.5)
Collecting tqdm
  Using cached tqdm-4.67.1-py3-none-any.whl.metadata (57 kB)
Collecting split-folders
  Using cached split_folders-0.5.1-py3-none-any.whl.metadata (6.2 kB)
Requirement already satisfied: albumentations in
/usr/local/lib/python3.10/dist-packages (1.4.15)
Collecting albumentations
  Using cached albumentations-1.4.21-py3-none-any.whl.metadata (31 kB)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from torch) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in
/usr/local/lib/python3.10/dist-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.10/dist-packages (from torch) (2024.6.1)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch)
  Using cached nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch)
  Using cached nvidia_cuda_runtime_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
```

```
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch)
  Using cached nvidia_cuda_cupti_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch)
  Using cached nvidia_cudnn_cu12-9.1.0.70-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch)
  Using cached nvidia_cublas_cu12-12.4.5.8-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch)
  Using cached nvidia_cufft_cu12-11.2.1.3-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch)
  Using cached nvidia_curand_cu12-10.3.5.147-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch)
  Using cached nvidia_cusolver_cu12-11.6.1.9-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch)
  Using cached nvidia_cusparse_cu12-12.3.1.170-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-nccl-cu12==2.21.5 (from torch)
  Using cached nvidia_nccl_cu12-2.21.5-py3-none-
manylinux2014_x86_64.whl.metadata (1.8 kB)
Collecting nvidia-nvtx-cu12==12.4.127 (from torch)
  Using cached nvidia_nvtx_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.7 kB)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting triton==3.1.0 (from torch)
  Using cached triton-3.1.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (1.3 kB)
Collecting sympy==1.13.1 (from torch)
  Using cached sympy-1.13.1-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch)
(1.3.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in
/usr/local/lib/python3.10/dist-packages (from torchvision) (10.4.0)
Requirement already satisfied: pyyaml in
/usr/local/lib/python3.10/dist-packages (from timm) (6.0.2)
Requirement already satisfied: huggingface_hub in
/usr/local/lib/python3.10/dist-packages (from timm) (0.24.7)
Requirement already satisfied: safetensors in
/usr/local/lib/python3.10/dist-packages (from timm) (0.4.5)
Requirement already satisfied: scipy>=1.10.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from albumentations) (1.13.1)
Requirement already satisfied: pydantic>=2.7.0 in
/usr/local/lib/python3.10/dist-packages (from albumentations) (2.9.2)
Collecting albucore==0.0.20 (from albumentations)
  Using cached albucore-0.0.20-py3-none-any.whl.metadata (5.3 kB)
Requirement already satisfied: eval-type-backport in
/usr/local/lib/python3.10/dist-packages (from albumentations) (0.2.0)
Requirement already satisfied: opencv-python-headless>=4.9.0.80 in
/usr/local/lib/python3.10/dist-packages (from albumentations)
(4.10.0.84)
Collecting stringzilla>=3.10.4 (from albucore==0.0.20->albumentations)
  Using cached stringzilla-3.10.11-cp310-cp310-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_28_x86_64.whl.metad
ata (79 kB)
Collecting simsimd>=5.9.2 (from albucore==0.0.20->albumentations)
  Using cached simsimd-6.2.1-cp310-cp310-
manylinux_2_28_x86_64.whl.metadata (66 kB)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2.7.0-
>albumentations) (0.7.0)
Requirement already satisfied: pydantic-core==2.23.4 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2.7.0-
>albumentations) (2.23.4)
Requirement already satisfied: packaging>=20.9 in
/usr/local/lib/python3.10/dist-packages (from huggingface_hub->timm)
(24.1)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from huggingface_hub->timm)
(2.32.3)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests-
>huggingface_hub->timm) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests-
>huggingface_hub->timm) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests-
>huggingface_hub->timm) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests-
>huggingface_hub->timm) (2024.8.30)
Downloading torch-2.5.1-cp310-cp310-manylinux1_x86_64.whl (906.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 906.4/906.4 MB 1.0 MB/s eta
0:00:00
anylinux2014_x86_64.whl (363.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 917.4 kB/s eta
0:00:00
```

```
anylinux2014_x86_64.whl (13.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 942.5 kB/s eta
0:00:00
anylinux2014_x86_64.whl (24.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 1.0 MB/s eta
0:00:00
e_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 1.4 MB/s eta
0:00:00
anylinux2014_x86_64.whl (664.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 3.5 MB/s eta
0:00:00
anylinux2014_x86_64.whl (211.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 3.2 MB/s eta
0:00:00
anylinux2014_x86_64.whl (56.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 2.5 MB/s eta
0:00:00
anylinux2014_x86_64.whl (127.9 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 2.6 MB/s eta
0:00:00
anylinux2014_x86_64.whl (207.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 2.4 MB/s eta
0:00:00
anylinux2014_x86_64.whl (188.7 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 188.7/188.7 MB 3.6 MB/s eta
0:00:00
anylinux2014_x86_64.whl (21.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 3.1 MB/s eta
0:00:00
anylinux2014_x86_64.whl (99 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.1/99.1 kB 2.9 MB/s eta
0:00:00
py-1.13.1-py3-none-any.whl (6.2 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.2/6.2 MB 3.4 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (209.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 209.5/209.5 MB 4.8 MB/s eta
0:00:00
anylinux1_x86_64.whl (7.2 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.2/7.2 MB 5.4 MB/s eta
0:00:00
m-1.0.11-py3-none-any.whl (2.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 5.5 MB/s eta
0:00:00
-4.67.1-py3-none-any.whl (78 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.5/78.5 kB 3.3 MB/s eta
0:00:00
entations-1.4.21-py3-none-any.whl (227 kB)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 227.9/227.9 kB 5.0 MB/s eta
0:00:00
simd-6.2.1-cp310-cp310-manylinux_2_28_x86_64.whl (632 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 632.7/632.7 kB 5.5 MB/s eta
0:00:00
anylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_28_x86_64.whl (291
kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 291.7/291.7 kB 5.5 MB/s eta
0:00:00
simd, triton, tqdm, sympy, split-folders, nvidia-nvtx-cu12, nvidia-
nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-
cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-
cupti-cu12, nvidia-cublas-cu12, nvidia-cusparse-cu12, nvidia-cudnn-
cu12, albucore, nvidia-cusolver-cu12, albumentations, torch,
torchvision, timm
  Attempting uninstall: tqdm
    Found existing installation: tqdm 4.66.5
    Uninstalling tqdm-4.66.5:
      Successfully uninstalled tqdm-4.66.5
  Attempting uninstall: sympy
    Found existing installation: sympy 1.13.3
    Uninstalling sympy-1.13.3:
      Successfully uninstalled sympy-1.13.3
  Attempting uninstall: nvidia-nccl-cu12
    Found existing installation: nvidia-nccl-cu12 2.23.4
    Uninstalling nvidia-nccl-cu12-2.23.4:
      Successfully uninstalled nvidia-nccl-cu12-2.23.4
  Attempting uninstall: albucore
    Found existing installation: albucore 0.0.16
    Uninstalling albucore-0.0.16:
      Successfully uninstalled albucore-0.0.16
  Attempting uninstall: albumentations
    Found existing installation: albumentations 1.4.15
    Uninstalling albumentations-1.4.15:
      Successfully uninstalled albumentations-1.4.15
  Attempting uninstall: torch
    Found existing installation: torch 2.4.1+cu121
    Uninstalling torch-2.4.1+cu121:
      Successfully uninstalled torch-2.4.1+cu121
  Attempting uninstall: torchvision
    Found existing installation: torchvision 0.19.1+cu121
    Uninstalling torchvision-0.19.1+cu121:
      Successfully uninstalled torchvision-0.19.1+cu121
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
fastai 2.7.17 requires torch<2.5,>=1.10, but you have torch 2.5.1
which is incompatible.
torchaudio 2.4.1+cu121 requires torch==2.4.1, but you have torch 2.5.1
```

```
which is incompatible.
Successfully installed albucore-0.0.20 albumentations-1.4.21 nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-runtime-cu12-12.4.127 nvidia-cudnn-cu12-9.1.0.70 nvidia-cufft-cu12-11.2.1.3 nvidia-curand-cu12-10.3.5.147 nvidia-cusolver-cu12-11.6.1.9 nvidia-cusparse-cu12-12.3.1.170 nvidia-nccl-cu12-2.21.5 nvidia-nvjitlink-cu12-12.4.127 nvidia-nvtx-cu12-12.4.127 simsimd-6.2.1 split-folders-0.5.1 stringzilla-3.10.11 sympy-1.13.1 timm-1.0.11 torch-2.5.1 torchvision-0.20.1 tqdm-4.67.1 triton-3.1.0
```

Download a file from Google Drive using the gdown library.

The file is identified by its file ID provided as an argument. The command will download the file to the current working directory in the Colab environment.

```
!gdown 1Vs8FTlr5mtyFrd_g4SxbTT5ll0mTn9T-
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1Vs8FTlr5mtyFrd_g4SxbTT5ll0mTn9T-
From (redirected): https://drive.google.com/uc?id=1Vs8FTlr5mtyFrd_g4SxbTT5ll0mTn9T-&confirm=t&uuid=cc7ac345-db25-4833-920f-4ef5b8d61126
To: /content/BdSLW.zip
100% 5.28G/5.28G [08:35<00:00, 10.2MB/s]
```

Mount Google Drive and Unzip Dataset

In this section, we are preparing to unzip a dataset from a zip file and parallelize the extraction process using multiple threads for improved performance. The function unzip_file extracts all files from a given zip file to a specified output directory. The zip file path and the output directory are set in the variables zip_path and output_dir, respectively. We also use ThreadPoolExecutor to parallelize the extraction, allowing faster processing of large datasets.

```
from google.colab import drive
from PIL import ImageFile
import splitfolders
import zipfile
import os
from concurrent.futures import ThreadPoolExecutor
import shutil
import random
from tqdm import tqdm
import numpy as np
from PIL import Image, ImageFilter, ImageOps
from torch.utils.data import DataLoader, Dataset
import multiprocessing
```

```
from torchvision import datasets, transforms
import albumentations as A
import torchvision.transforms as T
from PIL import Image

# Mount Google Drive
# drive.mount('/content/drive')

# Unzip the dataset
def unzip_file(zip_path, output_dir):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(output_dir)

# Split the work into chunks if needed
zip_path = "/content/BdSLW.zip"
output_dir = "/content/BdSLW"

# Use ThreadPoolExecutor to parallelize the extraction
with ThreadPoolExecutor(max_workers=20) as executor:
    executor.submit(unzip_file, zip_path, output_dir)
```

Split Dataset into Training, Validation, and Test Sets

In this block, the dataset is split into three subsets: training, validation, and testing. The function `split_dataset` takes the directory of the original dataset and splits it based on the provided ratios (`train_ratio`, `val_ratio`, `test_ratio`, default values: 0.6, 0.2, and 0.2, respectively). The function does the following:

1. **Creates Directories**: It creates directories for training, validation, and testing within their respective folders for each category.
2. **Shuffles and Splits Data**: For each category in the dataset, it shuffles the files and calculates indices to split the data according to the specified ratios.
3. **Moves Files**: The images are copied into their corresponding directories (train, val, or test) for each category.

This ensures that the dataset is properly split for model training, evaluation, and testing, with no data leakage between the sets.

```
def split_dataset(dataset_dir, train_dir, val_dir, test1_dir,
test2_dir, train_ratio=0.6, val_ratio=0.2, test1_ratio=0.1,
test2_ratio=0.1):
    # Ensure input ratios sum to 1
    assert train_ratio + val_ratio + test1_ratio + test2_ratio == 1.0,
"Split ratios must sum to 1"

    categories = [category for category in os.listdir(dataset_dir) if
os.path.isdir(os.path.join(dataset_dir, category))]

    # Create directories for train, val, test1, and test2 sets
```

```
    for target_dir in [train_dir, val_dir, test1_dir, test2_dir]:
        os.makedirs(target_dir, exist_ok=True)
        for category in categories:
            os.makedirs(os.path.join(target_dir, category),
exist_ok=True)

    for category in categories:
        category_path = os.path.join(dataset_dir, category)
        files = [file for file in os.listdir(category_path) if
os.path.isfile(os.path.join(category_path, file))]
        random.shuffle(files)

        # Calculate split indices
        total_files = len(files)
        train_end = int(total_files * train_ratio)
        val_end = train_end + int(total_files * val_ratio)
        test1_end = val_end + int(total_files * test1_ratio)

        train_files = files[:train_end]
        val_files = files[train_end:val_end]
        test1_files = files[val_end:test1_end]
        test2_files = files[test1_end:]

        # Copy files to respective directories
        for file in train_files:
            shutil.copy(os.path.join(category_path, file),
os.path.join(train_dir, category, file))

        for file in val_files:
            shutil.copy(os.path.join(category_path, file),
os.path.join(val_dir, category, file))

        for file in test1_files:
            shutil.copy(os.path.join(category_path, file),
os.path.join(test1_dir, category, file))

        for file in test2_files:
            shutil.copy(os.path.join(category_path, file),
os.path.join(test2_dir, category, file))

    print(f"Dataset successfully split into:\nTrain: {train_dir}\
nValidation: {val_dir}\nTest1: {test1_dir}\nTest2: {test2_dir}")

!mkdir /content/BdSLW41WordDatasetSplittedAugmented/test
!cp -r /content/BdSLW41WordDatasetSplittedAugmented/test1/*
/content/BdSLW41WordDatasetSplittedAugmented/test/
!cp -r /content/BdSLW41WordDatasetSplittedAugmented/test2/*
/content/BdSLW41WordDatasetSplittedAugmented/test/
```

Define Dataset Directories and Perform Dataset Splitting

In this block, the paths for the dataset and the split directories are defined:

1. **Dataset Directories**: The `dataset_dir` points to the location of the original dataset, while `train_dir`, `val_dir`, and `test_dir` define the locations where the training, validation, and test data will be stored after splitting.
2. **Dataset Splitting**: The `split_dataset` function (previously defined) is intended to be called to split the dataset into 60% training data, 20% validation data, and 20% test data. This function will populate the directories defined above with the corresponding files.

```
# Define paths
dataset_dir = '/content/BdSLW36'
train_dir = '/content/BdSLW41WordDatasetSplitted/train'
val_dir = '/content/BdSLW41WordDatasetSplitted/val'
test1_dir = '/content/BdSLW41WordDatasetSplitted/test1'
test2_dir = '/content/BdSLW41WordDatasetSplitted/test2'

# Split dataset
split_dataset(dataset_dir, train_dir, val_dir, test1_dir, test2_dir)
```

```
Dataset successfully split into:
Train: /content/BdSLW41WordDatasetSplitted/train
Validation: /content/BdSLW41WordDatasetSplitted/val
Test1: /content/BdSLW41WordDatasetSplitted/test1
Test2: /content/BdSLW41WordDatasetSplitted/test2
```

Custom Dataset and Image Augmentation Pipeline

1. **ImageDataset Class**:
   - This class is used for loading images and applying transformations.
   - It takes `image_paths` as input, along with a random augmentation function (`random_augment`) and a resize target (`resize_to`).
   - The `__getitem__` method loads and resizes images based on the provided paths and returns the image along with its path.
2. **Possible Transformations**:
   - A list of possible transformations is defined, such as `RandomRotation`, `ColorJitter` (for brightness, hue, saturation, and contrast), and `GaussianBlur`.
   - These transformations help augment the data by introducing variety and making the model more robust to different variations in the images.
3. **RandomAugmentations Class**:
   - A class that selects a random number of transformations (between 2 and 4) from the list of possible transformations and applies them to an image.
   - The `__call__` method allows it to be used like a function, transforming the image each time it is called.
4. **save_augmented_images Function**:

- This function saves the original images and their augmented versions (8 augmentations for each image).
- It first organizes the images by class and split (train, val, test), and then saves the original image and its augmentations in a specific folder structure.
- The images are saved with unique names that include the class name and a counter to ensure each image has a distinct filename.

5. **augment_and_save_images Function**:
   - This function orchestrates the entire process: it prepares the image paths, sets up the dataset and DataLoader, and then processes the images in batches.
   - It uses multiprocessing to speed up the augmentation process and a `tqdm` progress bar to show the status of the augmentation for each split (train, validation, and test).
   - After processing each batch, it saves the augmented images using the `save_augmented_images` function.
6. **Execution**:
   - The function `augment_and_save_images()` is called at the end of the code block to begin the image augmentation and saving process for the entire dataset, split into train, validation, and test sets.

```
# Custom Dataset
class ImageDataset(Dataset):
    def __init__(self, image_paths, random_augment, resize_to=(512,
512)):
        self.image_paths = image_paths
        self.random_augment = random_augment
        self.resize_to = resize_to

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        img = Image.open(img_path).convert("RGB")
        img = ImageOps.exif_transpose(img)
        img = img.resize(self.resize_to)
        return img, img_path


# Define possible transformations using Albumentations and torchvision
possible_transforms = [
    A.Rotate(limit=15, p=1.0),
    A.GaussNoise(var_limit=(50.0, 100.0), mean=10, p=0.95),
    A.RandomBrightnessContrast(p=0.95),
    A.HueSaturationValue(p=0.95),
    A.CLAHE(clip_limit=3.0, tile_grid_size=(8, 8), p=0.95),
    A.RandomGamma(p=0.95),
    A.ToGray(p=0.95),
    A.ChannelShuffle(p=0.95),
    A.RGBShift(r_shift_limit=30, g_shift_limit=30, b_shift_limit=30,
```

```
p=0.95)
]

# Random augmentation pipeline using Albumentations
class RandomAugmentations(object):
    def __init__(self, transforms, num_transforms_range=(2, 5)):
        self.transforms = transforms
        self.num_transforms_range = num_transforms_range

    def __call__(self, image):
        # Convert the Pillow Image to NumPy array before applying
Albumentations
        image = np.array(image)

        num_transforms = random.randint(*self.num_transforms_range)
        chosen_transforms = random.sample(self.transforms,
num_transforms)

        for transform in chosen_transforms:
            image = transform(image=image)["image"]

        # Convert back to Pillow Image if needed
        image = Image.fromarray(image)

        return image

random_augment = RandomAugmentations(possible_transforms)


# Save augmented images and original
def save_augmented_images(batch, output_dir, class_counters, split):
    for img, img_path in batch:
        class_name = os.path.basename(os.path.dirname(img_path))
        output_split_dir = os.path.join(output_dir, split, class_name)
        os.makedirs(output_split_dir, exist_ok=True)

        # Generate a unique ID for the class
        class_counter = class_counters.setdefault(split,
{}).setdefault(class_name, 1)

        # Save original image
        original_name = f"{class_name}_{class_counter}.jpg"
        original_path = os.path.join(output_split_dir, original_name)
        img.save(original_path)

        # Save augmentations (3 augmentations)
        for idx in range(3):  # 8 augmentations
            augmented_img = random_augment(img)
            augmented_name = f"{class_name}_{class_counter}_aug_{idx +
1}.jpg"
```

```
            augmented_path = os.path.join(output_split_dir,
augmented_name)
            augmented_img.save(augmented_path)

        # Increment class counter
        class_counters[split][class_name] += 1


# Main function for augmentation
def augment_and_save_images():
    input_dir = "/content/BdSLW41WordDatasetSplitted"
    output_dir = "/content/BdSLW41WordDatasetSplittedAugmented"

    # Collect all image paths grouped by splits
    image_paths_by_split = {'train': [], 'val': [], 'test1': [],
'test2': []}
    for split in image_paths_by_split.keys():
        split_dir = os.path.join(input_dir, split)
        for class_name in os.listdir(split_dir):
            class_dir = os.path.join(split_dir, class_name)
            if os.path.isdir(class_dir):
                image_paths_by_split[split].extend(
                    [os.path.join(class_dir, img) for img in
os.listdir(class_dir) if os.path.isfile(os.path.join(class_dir, img))]
                )

    # Track class counters for unique naming
    class_counters = {}

    # Process each split separately
    for split, image_paths in image_paths_by_split.items():
        # Dataset and DataLoader
        dataset = ImageDataset(image_paths, random_augment)
        cpu_count = multiprocessing.cpu_count()
        dataloader = DataLoader(dataset, batch_size=128,
shuffle=False, num_workers=cpu_count, collate_fn=lambda x: x)

        # Progress bar and processing
        total_images = len(image_paths)
        with tqdm(total=total_images, desc=f"Augmenting {split}
Images", unit="image") as pbar:
            for batch in dataloader:
                save_augmented_images(batch, output_dir,
class_counters, split)
                pbar.update(len(batch))

# Run
augment_and_save_images()
```

```
Augmenting train Images: 100%|██████████| 1282/1282 [00:37<00:00,
33.99image/s]
Augmenting val Images: 100%|██████████| 420/420 [00:20<00:00,
20.43image/s]
Augmenting test1 Images: 100%|██████████| 202/202 [00:15<00:00,
12.65image/s]
Augmenting test2 Images: 100%|██████████| 248/248 [00:16<00:00,
15.02image/s]
```

```
import torch
import torch.nn as nn
import torchvision
import torchvision.models as models
from torchvision.models import convnext_large, swin_v2_b,
regnet_y_32gf, resnext101_64x4d, densenet161, efficientnet_v2_s
from torchvision.models import ConvNeXt_Large_Weights,
Swin_V2_B_Weights, RegNet_Y_32GF_Weights, ResNeXt101_64X4D_Weights,
DenseNet161_Weights, EfficientNet_V2_S_Weights
from torchvision import datasets, transforms
from PIL import ImageFile
import splitfolders
import zipfile
import os
from concurrent.futures import ThreadPoolExecutor
import shutil
import random
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import torch.optim as optim
from tqdm import tqdm
from torch.cuda.amp import autocast
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.metrics import precision_score, recall_score, f1_score,
roc_auc_score, average_precision_score, balanced_accuracy_score,
top_k_accuracy_score
from sklearn.metrics import confusion_matrix,
precision_recall_fscore_support
import numpy as np
from PIL import Image, ImageFilter
from torch.utils.data import DataLoader, Dataset
from torch.utils.data import TensorDataset
import multiprocessing
import time
import gc
import torch.nn.functional as F
from torch.utils.data.sampler import SubsetRandomSampler
import itertools
from collections import Counter
from sklearn.metrics import root_mean_squared_error,
root_mean_squared_log_error
from dataVisualization import VisualizationTools as dv
import optuna
import json

model_configs = {
    "ConvNeXt_Large": {
        "model_fn": models.convnext_large,
        "weights_fn": models.ConvNeXt_Large_Weights.IMAGENET1K_V1,
```

```
        "classifier_layer": "classifier[-1]",
    },
    "DenseNet161": {
        "model_fn": models.densenet161,
        "weights_fn": models.DenseNet161_Weights.IMAGENET1K_V1,
        "classifier_layer": "classifier",
    },
    "EfficientNet_V2_S": {
        "model_fn": models.efficientnet_v2_s,
        "weights_fn": models.EfficientNet_V2_S_Weights.IMAGENET1K_V1,
        "classifier_layer": "classifier[-1]",
    },
    "RegNet_Y_32GF": {
        "model_fn": models.regnet_y_32gf,
        "weights_fn":
models.RegNet_Y_32GF_Weights.IMAGENET1K_SWAG_LINEAR_V1,
        "classifier_layer": "fc",
    },
    "ResNeXt101_64X4D": {
        "model_fn": models.resnext101_64x4d,
        "weights_fn": models.ResNeXt101_64X4D_Weights.IMAGENET1K_V1,
        "classifier_layer": "fc",
    },
    "Swin_V2_B": {
        "model_fn": models.swin_v2_b,
        "weights_fn": models.Swin_V2_B_Weights.IMAGENET1K_V1,
        "classifier_layer": "head",
    }
}

model_training_epoch = []
model_best_training_epoch = []

all_test_accuracies = []
all_test_losses = []
all_test_precisions = []
all_test_recalls = []
all_test_f1_scores = []
all_test_roc_auc_scores = []
all_test_pr_auc_scores = []
all_test_balanced_accuracies = []
all_test_top_5_accuracies = []

def calculate_metrics(all_labels, all_preds, all_probs):
    # Precision, Recall, F1 Score
    precision = precision_score(all_labels, all_preds,
average='weighted', zero_division=1)
    recall = recall_score(all_labels, all_preds, average='weighted',
zero_division=1)
    f1 = f1_score(all_labels, all_preds, average='weighted',
```

```
zero_division=1)

    # ROC AUC
    roc_auc = roc_auc_score(all_labels, all_probs, multi_class='ovr',
average='weighted')

    # PR AUC (Precision-Recall AUC)
    pr_auc = average_precision_score(all_labels, all_probs,
average='weighted')

    # Balanced Accuracy
    balanced_acc = balanced_accuracy_score(all_labels, all_preds)

    # Top-5 Accuracy
    top_5_accuracy = top_k_accuracy_score(all_labels, all_probs, k=5)

    return precision, recall, f1, roc_auc, pr_auc, balanced_acc,
top_5_accuracy
```

Train Test

```
label_smooth=0.01
l_r=0.0001
weight_d=1e-5
pat=1
fact=0.7
grad_clip=True
num_epochs = 100

folder_name=f"epoch{num_epochs}_lr{l_r}_pat{pat}_wd{weight_d}_ls{label
_smooth}_gc{grad_clip}_fact{fact}"

# Create the folder if it doesn't exist
if not os.path.exists(f"{folder_name}"):
    os.makedirs(f"{folder_name}")


def train_and_test_model(model, transform, num_classes, batch_size,
num_epochs):
    # Create training, validation, and test datasets
    train_dataset = datasets.ImageFolder(root=train_dir,
transform=transform)
    val_dataset = datasets.ImageFolder(root=val_dir,
transform=transform)
    test_dataset = datasets.ImageFolder(root=test_dir,
transform=transform)

    # Define batch size and create data loaders
    train_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=batch_size,
```

```
sampler=SubsetRandomSampler(torch.randperm(len(train_dataset))
[:int(0.7 * len(train_dataset))]))
    val_loader = torch.utils.data.DataLoader(val_dataset,
batch_size=batch_size, shuffle=False, num_workers=12)
    test_loader = torch.utils.data.DataLoader(test_dataset,
batch_size=batch_size, shuffle=False, num_workers=12)

    # Verify dataset loading and details
    # print(f"Number of training samples: {len(train_dataset)}")
    # print(f"Number of validation samples: {len(val_dataset)}")
    # print(f"Number of test samples: {len(test_dataset)}")
    # print(f"Classes: {train_dataset.classes}")
    # print(f"Class-to-Index Mapping: {train_dataset.class_to_idx}")

    # Use GPU if available
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    model = model.to(device)

    criterion = nn.CrossEntropyLoss(label_smoothing=label_smooth)
    optimizer = optim.AdamW(model.parameters(), lr=l_r,
weight_decay=weight_d)
    scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=pat,
factor=fact)

    if grad_clip:
      torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)


    # Initialize lists to store metrics
    train_losses = []
    train_accuracies = []
    train_precisions = []
    train_recalls = []
    train_f1_scores = []
    train_roc_auc = []
    train_pr_auc = []
    train_balanced_accuracies = []
    train_top_5_accuracies = []

    val_losses = []
    val_accuracies = []
    val_precisions = []
    val_recalls = []
    val_f1_scores = []
    val_roc_auc = []
    val_pr_auc = []
    val_balanced_accuracies = []
    val_top_5_accuracies = []
```

```
    test_loss=0
    test_accuracy=0
    test_precision=0
    test_recall=0
    test_f1_score=0
    test_roc_auc_score=0
    test_pr_auc_score=0
    test_balanced_accuracy=0

    start_time = time.time()

    early_stopping_threshold = 0.0001
    early_stopped = False
    best_accuracy = 0
    best_accuracy_epoch = 0

    for epoch in range(num_epochs):
      model.train()
      running_loss = 0.0
      correct_train = 0
      total_train = 0

      epoch_start_time = time.time()

      with tqdm(total=len(train_loader), desc=f"Epoch
{epoch+1}/{num_epochs}") as pbar:

          # Variables for accumulating predictions and labels
          all_train_labels = []
          all_train_preds = []
          all_train_probs = []

          for batch_idx, (images, labels) in enumerate(train_loader):
              images, labels = images.to(device), labels.to(device)

              # Forward pass
              outputs = model(images)
              loss = criterion(outputs, labels)

              # Backward pass and optimization
              optimizer.zero_grad()
              loss.backward()
              optimizer.step()

              probs = F.softmax(outputs, dim=1).type(torch.float32)
              assert
np.allclose(probs.sum(dim=1).cpu().detach().numpy(), 1.0, atol=1e-6),
"Probabilities do not sum to 1"

              # Update the running loss and calculate batch accuracy
```

```
            running_loss += loss.item()
            _, predicted = torch.max(probs, 1)
            total_train += labels.size(0)
            correct_train += (predicted == labels).sum().item()

            # Save predictions and labels for later metric
calculation
            all_train_labels.append(labels.cpu().numpy())
            all_train_preds.append(predicted.cpu().numpy())
            all_train_probs.append(probs.cpu().detach().numpy())

            # Calculate the average training loss and accuracy after
each batch
            avg_train_loss = running_loss / (batch_idx + 1)
            train_accuracy = 100 * correct_train / total_train

            # Update the progress bar with current loss and accuracy
less frequently
            if batch_idx % 10 == 0:  # Update every 10 batches
              elapsed_time = time.time() - start_time
              avg_epoch_time = elapsed_time / (epoch + 1)
              remaining_time = avg_epoch_time * (num_epochs - epoch
- 1)

              # Format times
              elapsed_str = time.strftime("%H:%M:%S",
time.gmtime(elapsed_time))
              remaining_str = time.strftime("%H:%M:%S",
time.gmtime(remaining_time))

              # Add timing info to progress bar
              pbar.set_postfix({
                  'Loss': f'{avg_train_loss:.4f}',
                  'Accuracy': f'{train_accuracy:.2f}%',
                  'Elapsed': elapsed_str,
                  'Remaining': remaining_str
              })
            pbar.update(1)

      # Compute average loss and accuracy for the whole epoch
      epoch_train_loss = running_loss / len(train_loader)
      train_losses.append(epoch_train_loss)
      epoch_train_accuracy = 100 * correct_train / total_train
      train_accuracies.append(train_accuracy)

      # Compute other metrics for the training set after the epoch
      all_train_labels = np.concatenate(all_train_labels)
      all_train_preds = np.concatenate(all_train_preds)
      all_train_probs = np.concatenate(all_train_probs)
```

```
        # Calculate all metrics
        train_precision, train_recall, train_f1, train_roc_auc_score,
train_pr_auc_score, train_balanced_accuracy, train_top_5_accuracy=
calculate_metrics(
            all_train_labels, all_train_preds, all_train_probs
        )

        # Append the metrics to the lists
        train_precisions.append(train_precision)
        train_recalls.append(train_recall)
        train_f1_scores.append(train_f1)
        train_roc_auc.append(train_roc_auc_score)
        train_pr_auc.append(train_pr_auc_score)
        train_balanced_accuracies.append(train_balanced_accuracy)
        train_top_5_accuracies.append(train_top_5_accuracy)

        # Print epoch results
        print(f"Epoch {epoch+1}/{num_epochs}, Training Loss:
{epoch_train_loss:.4f}, Accuracy: {train_accuracy:.2f}%")
        print(f"Training Precision: {train_precision:.4f}, Recall:
{train_recall:.4f}, F1 Score: {train_f1:.4f}")
        print(f"Training ROC AUC: {train_roc_auc_score:.4f}, PR AUC:
{train_pr_auc_score:.4f}, Balanced Accuracy:
{train_balanced_accuracy:.4f}")
        print(f"Training Top-5 Accuracy: {train_top_5_accuracy:.4f}")

        # Step the learning rate scheduler based on the training loss
        scheduler.step(epoch_train_loss)
        print(f"Learning Rate: {scheduler.get_last_lr()}")

        # Validation phase
        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0
        all_val_labels = []
        all_val_preds = []
        all_val_probs = []

        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)

                outputs = model(images)
                loss = criterion(outputs, labels)

                val_loss += loss.item()

                probs = F.softmax(outputs, dim=1).type(torch.float32)
                # print("Sum of probabilities:", probs.sum(dim=1))
```

```
            assert
np.allclose(probs.sum(dim=1).cpu().detach().numpy(), 1.0, atol=1e-6),
"Probabilities do not sum to 1"

            # Calculate predictions using the correct outputs
            _, predicted = torch.max(probs, 1)  # Update with the
validation outputs

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

            all_val_labels.append(labels.cpu().numpy())
            all_val_preds.append(predicted.cpu().numpy())
            all_val_probs.append(probs.cpu().detach().numpy())

      # Average validation loss and accuracy for the epoch
      epoch_val_loss = val_loss / len(val_loader)
      val_accuracy = 100 * correct / total
      val_losses.append(epoch_val_loss)
      val_accuracies.append(val_accuracy)

      # Compute other metrics for validation
      all_val_labels = np.concatenate(all_val_labels)
      all_val_preds = np.concatenate(all_val_preds)
      all_val_probs = np.concatenate(all_val_probs)

      # Calculate all metrics using the function
      val_precision, val_recall, val_f1, val_roc_auc_score,
val_pr_auc_score, val_balanced_accuracy, val_top_5_accuracy =
calculate_metrics(
      all_val_labels, all_val_preds, all_val_probs)

      #append metrics to the list
      val_precisions.append(val_precision)
      val_recalls.append(val_recall)
      val_f1_scores.append(val_f1)
      val_roc_auc.append(val_roc_auc_score)
      val_pr_auc.append(val_pr_auc_score)
      val_balanced_accuracies.append(val_balanced_accuracy)
      val_top_5_accuracies.append(val_top_5_accuracy)

      # Print validation results
      print(f"Validation Loss: {epoch_val_loss:.4f}, Accuracy:
{val_accuracy:.2f}%")
      print(f"Validation Precision: {val_precision:.4f}, Recall:
{val_recall:.4f}, F1 Score: {val_f1:.4f}")
      print(f"Validation ROC AUC: {val_roc_auc_score:.4f}, PR AUC:
{val_pr_auc_score:.4f}, Balanced Accuracy:
{val_balanced_accuracy:.4f}")
      print(f"Validation Top-5 Accuracy: {val_top_5_accuracy:.4f}")
```

```
      if val_accuracy > best_accuracy:
          filepath =
f"/content/{folder_name}/{type(model).__name__}_{best_accuracy_epoch}.
pth"

          if os.path.exists(filepath):
            os.remove(filepath)
          best_accuracy = val_accuracy
          best_accuracy_epoch = epoch + 1
          pthFilename =
f"/content/{folder_name}/{type(model).__name__}_{epoch+1}.pth"
          torch.save({
              'epoch': num_epochs,
              'model_state_dict': model.state_dict(),
              'optimizer_state_dict': optimizer.state_dict(),
              'train_loss': train_losses,
              'val_loss': val_losses,
              'train_accuracy': train_accuracies,
              'val_accuracy': val_accuracies,
              'train_precisions': train_precisions,
              'val_precisions': val_precisions,
              'train_recalls': train_recalls,
              'val_recalls': val_recalls,
              'train_f1_scores': train_f1_scores,
              'val_f1_scores': val_f1_scores,
              'train_roc_auc': train_roc_auc,
              'val_roc_auc': val_roc_auc,
              'train_roc_pr': train_pr_auc,
              'val_roc_pr': val_pr_auc,
              'train_balanced_accuracies': train_balanced_accuracies,
              'val_balanced_accuracies': val_balanced_accuracies,
              'train_top_5_accuracies': train_top_5_accuracies,
              'val_top_5_accuracies': val_top_5_accuracies,
          }, pthFilename)


      # Use robust_average for loss and accuracy tracking
      if len(val_losses) >= 10 and len(val_losses) % 5 == 0:
          avg_loss_change = robust_average(val_losses[-10:]) -
robust_average(val_losses[-5:])
          avg_accuracy_change = robust_average(val_accuracies[-5:]) -
robust_average(val_accuracies[-10:])
          print(f"Average Loss Change: {avg_loss_change:.6f}, Average
Accuracy Change: {avg_accuracy_change:.6f}")

          if avg_loss_change < early_stopping_threshold and
avg_accuracy_change < early_stopping_threshold:
              print(f"Early stopping at epoch {epoch+1}")
              model_training_epoch.append(epoch+1)
```

```
                early_stopped = True
                break

    if early_stopped==False:
      model_training_epoch.append(num_epochs)

    # Load the saved model state
    model_path =
f"/content/{folder_name}/{type(model).__name__}_{best_accuracy_epoch}.
pth"
    saved_state = torch.load(model_path, map_location=device)
    model.load_state_dict(saved_state['model_state_dict'])

    model.eval()
    model_best_training_epoch.append(best_accuracy_epoch)
    test_running_loss = 0.0
    correct_test = 0
    total_test = 0

    # Variables for accumulating test predictions and labels
    all_test_labels = []
    all_test_preds = []
    all_test_probs = []

    with torch.no_grad():
        for batch_idx, (images, labels) in enumerate(test_loader):
            images, labels = images.to(device), labels.to(device)

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            probs = F.softmax(outputs, dim=1).type(torch.float32)

            # print("Sum of probabilities:", probs.sum(dim=1))
            assert
np.allclose(probs.sum(dim=1).cpu().detach().numpy(), 1.0, atol=1e-6),
"Probabilities do not sum to 1"

            # Update the running loss and calculate batch accuracy
            test_running_loss += loss.item()
            _, predicted = torch.max(probs, 1)
            total_test += labels.size(0)
            correct_test += (predicted == labels).sum().item()

            # Save predictions and labels for metric calculation
            all_test_labels.append(labels.cpu().numpy())
            all_test_preds.append(predicted.cpu().numpy())
            all_test_probs.append(probs.cpu().detach().numpy())
```

```
    # Compute average loss and accuracy for the test set
    test_loss = test_running_loss / len(test_loader)
    test_accuracy = 100 * correct_test / total_test

    # Concatenate all predictions and labels for metric calculation
    all_test_labels = np.concatenate(all_test_labels)
    all_test_preds = np.concatenate(all_test_preds)
    all_test_probs = np.concatenate(all_test_probs)

    # Calculate metrics using the function
    test_precision, test_recall, test_f1, test_roc_auc_score,
test_pr_auc_score, test_balanced_accuracy, test_top_5_accuracy =
calculate_metrics(
        all_test_labels, all_test_preds, all_test_probs
    )

    # Print test results
    print(f"Test Loss: {test_loss:.4f}, Test Accuracy:
{test_accuracy:.2f}%")
    print(f"Test Precision: {test_precision:.4f}, Recall:
{test_recall:.4f}, F1 Score: {test_f1:.4f}")
    print(f"Test ROC AUC: {test_roc_auc_score:.4f}, PR AUC:
{test_pr_auc_score:.4f} Balanced Accuracy:
{test_balanced_accuracy:.4f}")
    print(f"Test Top-5 Accuracy: {test_top_5_accuracy:.4f}")

    all_test_accuracies.append(test_accuracy)
    all_test_losses.append(test_loss)
    all_test_precisions.append(test_precision)
    all_test_recalls.append(test_recall)
    all_test_f1_scores.append(test_f1)
    all_test_roc_auc_scores.append(test_roc_auc_score)
    all_test_pr_auc_scores.append(test_pr_auc_score)
    all_test_balanced_accuracies.append(test_balanced_accuracy*100)
    all_test_top_5_accuracies.append(test_top_5_accuracy*100)

    df = pd.DataFrame(all_test_probs)
    csvFilename =
f"/content/{folder_name}/{type(model).__name__}_{best_accuracy_epoch}.
csv"
    df.to_csv(csvFilename, index=False)

    lengths = {
        'train_losses': len(train_losses),
        'train_accuracies': len(train_accuracies),
        'train_precisions': len(train_precisions),
        'train_recalls': len(train_recalls),
        'train_f1_scores': len(train_f1_scores),
        'train_roc_auc': len(train_roc_auc),
        'train_roc_pr': len(train_pr_auc),
```

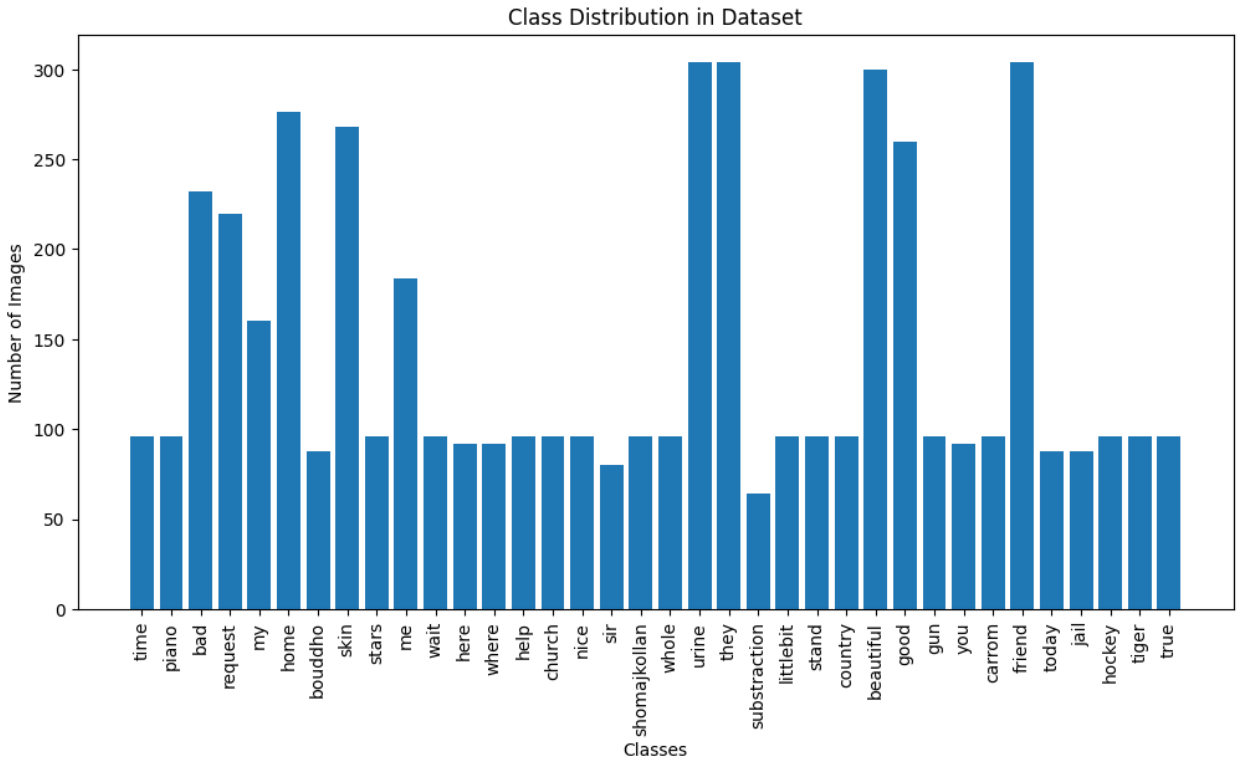
```
        'train_balanced_accuracies': len(train_balanced_accuracies),
        'train_top_5_accuracies': len(train_top_5_accuracies),
        'val_losses': len(val_losses),
        'val_accuracies': len(val_accuracies),
        'val_precisions': len(val_precisions),
        'val_recalls': len(val_recalls),
        'val_f1_scores': len(val_f1_scores),
        'val_roc_auc': len(val_roc_auc),
        'val_roc_pr': len(val_pr_auc),
        'val_balanced_accuracies': len(val_balanced_accuracies),
        'val_top_5_accuracies': len(val_top_5_accuracies)
    }

    epochs = range(1, epoch + 2)

    metrics_df = pd.DataFrame({
        'Epoch': epochs,
        'Train Loss': train_losses,
        'Val Loss': val_losses,
        'Train Accuracy': train_accuracies,
        'Val Accuracy': val_accuracies,
        'Train Precision': train_precisions,
        'Val Precision': val_precisions,
        'Train Recall': train_recalls,
        'Val Recall': val_recalls,
        'Train F1 Score': train_f1_scores,
        'Val F1 Score': val_f1_scores,
        'Train AUC-ROC': train_roc_auc,
        'Val AUC-ROC': val_roc_auc,
        'Train AUC-PR': train_pr_auc,
        'Val AUC-PR': val_pr_auc,
        'Train Balanced Accuracy': train_balanced_accuracies,
        'Val Balanced Accuracy': val_balanced_accuracies,
        'Train Top-5 Accuracy': train_top_5_accuracies,
        'Val Top-5 Accuracy': val_top_5_accuracies
    })

    # Plotting the metrics using Seaborn

    sns.set(style="whitegrid")

    fig, axes = plt.subplots(3, 3, figsize=(20, 20))
    fig.tight_layout(pad=5.0)

    # Plot the Train and Validation Loss
    sns.lineplot(x='Epoch', y='Train Loss', data=metrics_df,
ax=axes[0, 0], label='Train Loss', color='blue')
    sns.lineplot(x='Epoch', y='Val Loss', data=metrics_df, ax=axes[0,
0], label='Val Loss', color='orange')
    axes[0, 0].set_title('Loss Curves')
```

```
    # Plot the Train and Validation Accuracy
    sns.lineplot(x='Epoch', y='Train Accuracy', data=metrics_df,
ax=axes[0, 1], label='Train Accuracy', color='blue')
    sns.lineplot(x='Epoch', y='Val Accuracy', data=metrics_df,
ax=axes[0, 1], label='Val Accuracy', color='orange')
    axes[0, 1].set_title('Accuracy Curves')

    # Plot Precision for Train and Validation
    sns.lineplot(x='Epoch', y='Train Precision', data=metrics_df,
ax=axes[0, 2], label='Train Precision', color='blue')
    sns.lineplot(x='Epoch', y='Val Precision', data=metrics_df,
ax=axes[0, 2], label='Val Precision', color='orange')
    axes[0, 2].set_title('Precision Curves')

    # Plot Recall for Train and Validation
    sns.lineplot(x='Epoch', y='Train Recall', data=metrics_df,
ax=axes[1, 0], label='Train Recall', color='blue')
    sns.lineplot(x='Epoch', y='Val Recall', data=metrics_df,
ax=axes[1, 0], label='Val Recall', color='orange')
    axes[1, 0].set_title('Recall Curves')

    # Plot F1 Score for Train and Validation
    sns.lineplot(x='Epoch', y='Train F1 Score', data=metrics_df,
ax=axes[1, 1], label='Train F1', color='blue')
    sns.lineplot(x='Epoch', y='Val F1 Score', data=metrics_df,
ax=axes[1, 1], label='Val F1', color='orange')
    axes[1, 1].set_title('F1 Score Curves')

    # Plot AUC-ROC for Train and Validation
    sns.lineplot(x='Epoch', y='Train AUC-ROC', data=metrics_df,
ax=axes[1, 2], label='Train AUC-ROC', color='blue')
    sns.lineplot(x='Epoch', y='Val AUC-ROC', data=metrics_df,
ax=axes[1, 2], label='Val AUC-ROC', color='orange')
    axes[1, 2].set_title('AUC-ROC Curves')

    # Plot AUC-PR for Train and Validation (added AUC-PR)
    sns.lineplot(x='Epoch', y='Train AUC-PR', data=metrics_df,
ax=axes[2, 0], label='Train AUC-PR', color='blue')
    sns.lineplot(x='Epoch', y='Val AUC-PR', data=metrics_df,
ax=axes[2, 0], label='Val AUC-PR', color='orange')
    axes[2, 0].set_title('AUC-PR Curves')

    # Plot Balanced Accuracy for Train and Validation
    sns.lineplot(x='Epoch', y='Train Balanced Accuracy',
data=metrics_df, ax=axes[2, 1], label='Train Balanced Accuracy',
color='blue')
    sns.lineplot(x='Epoch', y='Val Balanced Accuracy',
data=metrics_df, ax=axes[2, 1], label='Val Balanced Accuracy',
color='orange')
```

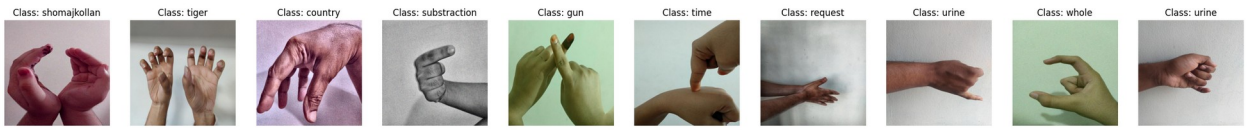
```
    axes[2, 1].set_title('Balanced Accuracy Curves')

    # Plot Top-5 Accuracy for Train and Validation
    sns.lineplot(x='Epoch', y='Train Top-5 Accuracy', data=metrics_df,
ax=axes[2, 2], label='Train Top-5 Accuracy', color='blue')
    sns.lineplot(x='Epoch', y='Val Top-5 Accuracy', data=metrics_df,
ax=axes[2, 2], label='Val Top-5 Accuracy', color='orange')
    axes[2, 2].set_title('Top-5 Accuracy Curves')

    # Adjust layout to prevent overlap
    plt.tight_layout()

    # Show the plot
    plt.show()


    print("\nConfusion Matrix:")
    dv.plot_confusion_matrix(all_test_labels,all_test_preds,
test_dataset.classes)
    print("\nError Analysis:")
    dv.plot_error_analysis(all_test_labels, all_test_preds,
test_dataset.classes)
```

```
# Dataset directories
train_dir = '/content/BdSLW41WordDatasetSplittedAugmented/train'
val_dir = '/content/BdSLW41WordDatasetSplittedAugmented/val'
test_dir = '/content/BdSLW41WordDatasetSplittedAugmented/test1'

# Handle truncated images
ImageFile.LOAD_TRUNCATED_IMAGES = True

num_classes = 36
batch_sizes = {
    "EfficientNet_V2_S": 16,
    "ConvNeXt_Large": 16,
    "Swin_V2_B": 16,
    "RegNet_Y_32GF": 16,
    "ResNeXt101_64X4D": 32,
    "DenseNet161": 32,
}

def GPU_unload():
  model = None
  del model
  torch.cuda.empty_cache()
  gc.collect()

def robust_average(values):
    if len(values) < 2:
        return sum(values) / len(values)
```

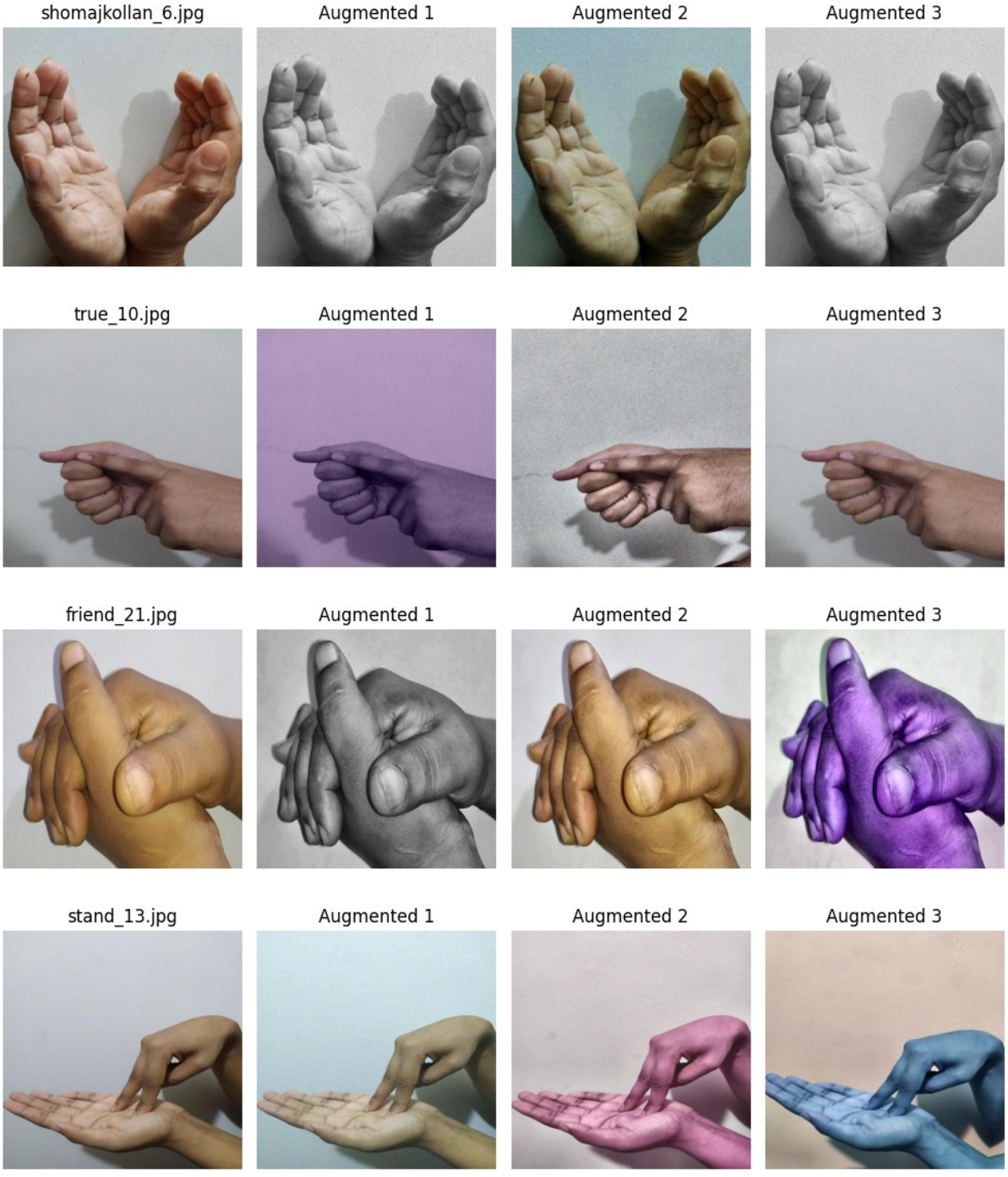
```
    diffs = np.abs(np.diff(values))
    median_diff = np.median(diffs)
    threshold = median_diff

    filtered_values = [values[0]]
    for i in range(1, len(values)):
        if np.abs(values[i] - values[i - 1]) <= threshold:
            filtered_values.append(values[i])

    if len(filtered_values) == 0:
        filtered_values = values

    return sum(filtered_values) / len(filtered_values)

import os
import matplotlib.pyplot as plt

def plot_class_distribution(dataset_dir):
    categories = os.listdir(dataset_dir)
    class_counts = [len(os.listdir(os.path.join(dataset_dir,
category))) for category in categories]

    plt.figure(figsize=(12, 6))
    plt.bar(categories, class_counts)
    plt.xticks(rotation=90)
    plt.xlabel('Classes')
    plt.ylabel('Number of Images')
    plt.title('Class Distribution in Dataset')
    plt.show()

# Plot class distribution
dataset_dir = '/content/BdSLW41WordDatasetSplittedAugmented/train'
plot_class_distribution(dataset_dir)
```

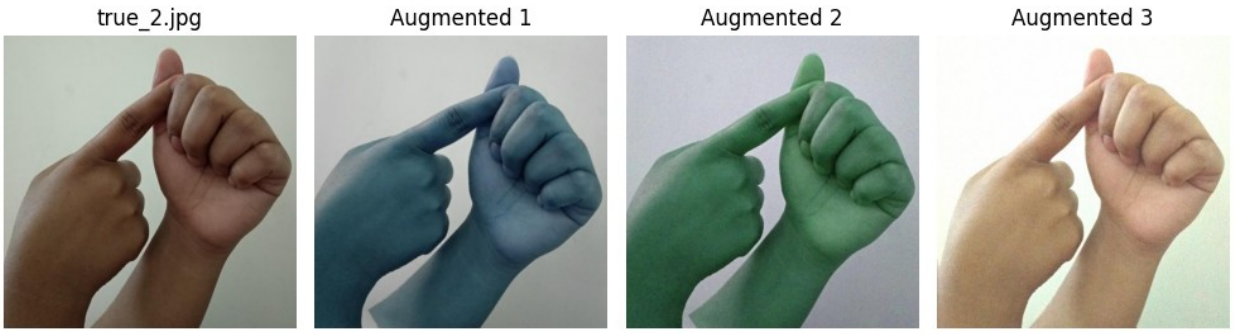



```python
def visualize_sample_images(dataset_dir, num_samples=10):
    categories = os.listdir(dataset_dir)
    fig, axs = plt.subplots(1, num_samples, figsize=(30, 20))
    axs = axs.ravel()

    for i in range(num_samples):
        category = random.choice(categories)
        class_dir = os.path.join(dataset_dir, category)
        image_files = os.listdir(class_dir)
        img_path = os.path.join(class_dir, random.choice(image_files))

        img = Image.open(img_path)
        axs[i].imshow(img)
        axs[i].set_title(f"Class: {category}")
        axs[i].axis('off')

    plt.show()

# Visualize sample images
visualize_sample_images(dataset_dir)
```



```
def visualize_saved_augmented_images(dataset_dir, num_augmentations):
    for i in range(1, 6):
      categories = os.listdir(dataset_dir)
      category = random.choice(categories)
      class_dir = os.path.join(dataset_dir, category)
      image_files = [f for f in os.listdir(class_dir) if '_aug' not in
f]
      if not image_files:
          print(f"No base images found in {class_dir}.")
          return

      base_image_name = random.choice(image_files)
      base_image_path = os.path.join(class_dir, base_image_name)

      # Load the original image
      original_img = Image.open(base_image_path)

      # Load augmented images based on naming pattern
      augmented_images = []
      for i in range(1, num_augmentations + 1):
          aug_image_name = f"{os.path.splitext(base_image_name)
[0]}_aug_{i}{os.path.splitext(base_image_name)[1]}"
          aug_image_path = os.path.join(class_dir, aug_image_name)
          if os.path.exists(aug_image_path):
              augmented_images.append(Image.open(aug_image_path))
          else:
              print(f"Augmented image not found: {aug_image_path}")

      # Plot original and augmented images
      total_images = len(augmented_images) + 1
      fig, axs = plt.subplots(1, total_images, figsize=(10, 3))
      axs[0].imshow(original_img)
      axs[0].set_title(base_image_name)
      axs[0].axis('off')

      for i, aug_img in enumerate(augmented_images):
          axs[i + 1].imshow(aug_img)
          axs[i + 1].set_title(f"Augmented {i + 1}")
          axs[i + 1].axis('off')

      plt.tight_layout()
      plt.show()

visualize_saved_augmented_images('/content/BdSLW41WordDatasetSplittedA
ugmented/train', num_augmentations=3)
```


shomajkollan_6.jpg
Augmented 1
Augmented 2
Augmented 3
true_10.jpg
Augmented 1
Augmented 2
Augmented 3
friend_21.jpg
Augmented 1
Augmented 2
Augmented 3
stand_13.jpg
Augmented 1
Augmented 2
Augmented 3



Results

```
# List of model names from your configurations
model_names = list(model_configs.keys())

# selected_indices = list(range(0, 7))

selected_indices = [0,1,2,3,4,5]
# selected_indices = [4]

# Loop through the hardcoded list of selected model indices
for idx in selected_indices:
    model_name = model_names[idx]
    print(f"Training and testing {model_name} model...")

    config = model_configs[model_name]
    print(f"Training and testing {model_name} model...")

    model = config["model_fn"](weights=config["weights_fn"])
    transform = config["weights_fn"].transforms()

    # Modify the classifier layer based on the model's weight set
    if config["weights_fn"] ==
models.EfficientNet_V2_S_Weights.IMAGENET1K_V1:
        model.classifier[-1] = nn.Linear(model.classifier[-
1].in_features, num_classes)

    elif config["weights_fn"] ==
models.ConvNeXt_Large_Weights.IMAGENET1K_V1:
        model.classifier[-1] = nn.Linear(model.classifier[-
1].in_features, num_classes)

    elif config["weights_fn"] ==
models.Swin_V2_B_Weights.IMAGENET1K_V1:
        model.head = nn.Linear(model.head.in_features, num_classes)

    elif config["weights_fn"] ==
models.RegNet_Y_32GF_Weights.IMAGENET1K_SWAG_LINEAR_V1:
        model.fc = nn.Linear(model.fc.in_features, num_classes)
```

```
    elif config["weights_fn"] ==
models.ResNeXt101_64X4D_Weights.IMAGENET1K_V1:
        model.fc = nn.Linear(model.fc.in_features, num_classes)

    elif config["weights_fn"] ==
models.DenseNet161_Weights.IMAGENET1K_V1:
        model.classifier = nn.Linear(model.classifier.in_features,
num_classes)

    elif config["weights_fn"] ==
models.Inception_V3_Weights.IMAGENET1K_V1:
        model.fc = nn.Linear(model.fc.in_features, num_classes)

    # Fetch batch size for this model and start training/testing
    batch_size = batch_sizes.get(model_name)
    train_and_test_model(model, transform, num_classes, batch_size,
num_epochs)

    GPU_unload()

Training and testing ConvNeXt_Large model...
Training and testing ConvNeXt_Large model...

Epoch 1/100: 100%|██████████| 225/225 [00:53<00:00,  4.18it/s,
Loss=1.2931, Accuracy=76.05%, Elapsed=00:00:53, Remaining=01:27:27]

Epoch 1/100, Training Loss: 1.2743, Accuracy: 76.37%
Training Precision: 0.8141, Recall: 0.7637, F1 Score: 0.7656
Training ROC AUC: 0.9770, PR AUC: 0.8396, Balanced Accuracy: 0.7077
Training Top-5 Accuracy: 0.8933
Learning Rate: [0.0001]
Validation Loss: 0.2004, Accuracy: 98.39%
Validation Precision: 0.9854, Recall: 0.9839, F1 Score: 0.9838
Validation ROC AUC: 1.0000, PR AUC: 0.9989, Balanced Accuracy: 0.9779
Validation Top-5 Accuracy: 0.9982

Epoch 2/100: 100%|██████████| 225/225 [00:52<00:00,  4.29it/s,
Loss=0.1339, Accuracy=99.83%, Elapsed=00:01:51, Remaining=01:31:25]

Epoch 2/100, Training Loss: 0.1332, Accuracy: 99.83%
Training Precision: 0.9983, Recall: 0.9983, F1 Score: 0.9983
Training ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9983
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1295, Accuracy: 98.99%
Validation Precision: 0.9904, Recall: 0.9899, F1 Score: 0.9898
Validation ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9867
Validation Top-5 Accuracy: 1.0000
```

```
Epoch 3/100: 100%|██████████| 225/225 [00:52<00:00,  4.29it/s,
Loss=0.0995, Accuracy=99.97%, Elapsed=00:02:50, Remaining=01:31:52]

Epoch 3/100, Training Loss: 0.0994, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9998
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1279, Accuracy: 98.45%
Validation Precision: 0.9856, Recall: 0.9845, F1 Score: 0.9844
Validation ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9797
Validation Top-5 Accuracy: 1.0000

Epoch 4/100: 100%|██████████| 225/225 [00:50<00:00,  4.48it/s,
Loss=0.0937, Accuracy=100.00%, Elapsed=00:03:45, Remaining=01:30:06]

Epoch 4/100, Training Loss: 0.0937, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1322, Accuracy: 98.57%
Validation Precision: 0.9867, Recall: 0.9857, F1 Score: 0.9856
Validation ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9825
Validation Top-5 Accuracy: 1.0000

Epoch 5/100: 100%|██████████| 225/225 [00:49<00:00,  4.50it/s,
Loss=0.0923, Accuracy=100.00%, Elapsed=00:04:39, Remaining=01:28:36]

Epoch 5/100, Training Loss: 0.0923, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1306, Accuracy: 98.45%
Validation Precision: 0.9856, Recall: 0.9845, F1 Score: 0.9844
Validation ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9804
Validation Top-5 Accuracy: 1.0000

Epoch 6/100: 100%|██████████| 225/225 [00:49<00:00,  4.58it/s,
Loss=0.0962, Accuracy=99.86%, Elapsed=00:05:33, Remaining=01:27:02]
```

```
Epoch 6/100, Training Loss: 0.0962, Accuracy: 99.86%
Training Precision: 0.9986, Recall: 0.9986, F1 Score: 0.9986
Training ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9983
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1497, Accuracy: 98.21%
Validation Precision: 0.9832, Recall: 0.9821, F1 Score: 0.9823
Validation ROC AUC: 0.9999, PR AUC: 0.9971, Balanced Accuracy: 0.9824
Validation Top-5 Accuracy: 1.0000

Epoch 7/100: 100%|██████████| 225/225 [00:51<00:00,  4.40it/s,
Loss=0.0939, Accuracy=100.00%, Elapsed=00:06:28, Remaining=01:26:05]

Epoch 7/100, Training Loss: 0.0938, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1212, Accuracy: 99.17%
Validation Precision: 0.9922, Recall: 0.9917, F1 Score: 0.9917
Validation ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9923
Validation Top-5 Accuracy: 0.9994

Epoch 8/100: 100%|██████████| 225/225 [00:51<00:00,  4.38it/s,
Loss=0.0911, Accuracy=100.00%, Elapsed=00:07:26, Remaining=01:25:35]

Epoch 8/100, Training Loss: 0.0911, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1160, Accuracy: 99.05%
Validation Precision: 0.9910, Recall: 0.9905, F1 Score: 0.9905
Validation ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9896
Validation Top-5 Accuracy: 1.0000

Epoch 9/100: 100%|██████████| 225/225 [00:50<00:00,  4.46it/s,
Loss=0.0906, Accuracy=100.00%, Elapsed=00:08:21, Remaining=01:24:30]

Epoch 9/100, Training Loss: 0.0906, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
```

```
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1180, Accuracy: 99.17%
Validation Precision: 0.9922, Recall: 0.9917, F1 Score: 0.9917
Validation ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9917
Validation Top-5 Accuracy: 1.0000

Epoch 10/100: 100%|██████████| 225/225 [00:51<00:00,  4.38it/s,
Loss=0.0905, Accuracy=100.00%, Elapsed=00:09:17, Remaining=01:23:35]

Epoch 10/100, Training Loss: 0.0905, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1205, Accuracy: 99.05%
Validation Precision: 0.9910, Recall: 0.9905, F1 Score: 0.9904
Validation ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9895
Validation Top-5 Accuracy: 1.0000
Average Loss Change: 0.008878, Average Accuracy Change: 0.089286

Epoch 11/100: 100%|██████████| 225/225 [00:49<00:00,  4.52it/s,
Loss=0.0902, Accuracy=100.00%, Elapsed=00:10:11, Remaining=01:22:28]

Epoch 11/100, Training Loss: 0.0903, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1332, Accuracy: 98.87%
Validation Precision: 0.9894, Recall: 0.9887, F1 Score: 0.9886
Validation ROC AUC: 0.9999, PR AUC: 0.9983, Balanced Accuracy: 0.9875
Validation Top-5 Accuracy: 0.9994

Epoch 12/100: 100%|██████████| 225/225 [00:51<00:00,  4.36it/s,
Loss=0.1147, Accuracy=99.43%, Elapsed=00:11:07, Remaining=01:21:36]

Epoch 12/100, Training Loss: 0.1163, Accuracy: 99.41%
Training Precision: 0.9942, Recall: 0.9941, F1 Score: 0.9941
Training ROC AUC: 0.9997, PR AUC: 0.9987, Balanced Accuracy: 0.9964
Training Top-5 Accuracy: 0.9994
Learning Rate: [7e-05]
```

```
Validation Loss: 0.1637, Accuracy: 97.38%
Validation Precision: 0.9759, Recall: 0.9738, F1 Score: 0.9735
Validation ROC AUC: 0.9999, PR AUC: 0.9988, Balanced Accuracy: 0.9657
Validation Top-5 Accuracy: 0.9988

Epoch 13/100: 100%|██████████| 225/225 [00:50<00:00,  4.44it/s,
Loss=0.1115, Accuracy=99.52%, Elapsed=00:12:02, Remaining=01:20:36]

Epoch 13/100, Training Loss: 0.1112, Accuracy: 99.53%
Training Precision: 0.9953, Recall: 0.9953, F1 Score: 0.9953
Training ROC AUC: 0.9999, PR AUC: 0.9985, Balanced Accuracy: 0.9957
Training Top-5 Accuracy: 0.9997
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1199, Accuracy: 99.40%
Validation Precision: 0.9945, Recall: 0.9940, F1 Score: 0.9941
Validation ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9948
Validation Top-5 Accuracy: 1.0000

Epoch 14/100: 100%|██████████| 225/225 [00:51<00:00,  4.36it/s,
Loss=0.0913, Accuracy=100.00%, Elapsed=00:13:00, Remaining=01:19:54]

Epoch 14/100, Training Loss: 0.0913, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1207, Accuracy: 99.17%
Validation Precision: 0.9923, Recall: 0.9917, F1 Score: 0.9916
Validation ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9914
Validation Top-5 Accuracy: 1.0000

Epoch 15/100: 100%|██████████| 225/225 [00:49<00:00,  4.51it/s,
Loss=0.0909, Accuracy=100.00%, Elapsed=00:13:54, Remaining=01:18:50]

Epoch 15/100, Training Loss: 0.0909, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [3.4299999999999993e-05]



Validation Loss: 0.1155, Accuracy: 99.52%
Validation Precision: 0.9956, Recall: 0.9952, F1 Score: 0.9953
Validation ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9960
```

```
Validation Top-5 Accuracy: 1.0000
Average Loss Change: 0.000274, Average Accuracy Change: 0.267857

Epoch 16/100: 100%|██████████| 225/225 [00:50<00:00,  4.47it/s,
Loss=0.0906, Accuracy=100.00%, Elapsed=00:14:50, Remaining=01:17:57]

Epoch 16/100, Training Loss: 0.0906, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [3.4299999999999993e-05]



Validation Loss: 0.1167, Accuracy: 99.52%
Validation Precision: 0.9956, Recall: 0.9952, F1 Score: 0.9953
Validation ROC AUC: 1.0000, PR AUC: 0.9995, Balanced Accuracy: 0.9960
Validation Top-5 Accuracy: 1.0000

Epoch 17/100: 100%|██████████| 225/225 [00:49<00:00,  4.54it/s,
Loss=0.0905, Accuracy=100.00%, Elapsed=00:15:44, Remaining=01:16:53]

Epoch 17/100, Training Loss: 0.0905, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1162, Accuracy: 99.46%
Validation Precision: 0.9950, Recall: 0.9946, F1 Score: 0.9947
Validation ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9958
Validation Top-5 Accuracy: 1.0000

Epoch 18/100: 100%|██████████| 225/225 [00:49<00:00,  4.53it/s,
Loss=0.0906, Accuracy=100.00%, Elapsed=00:16:38, Remaining=01:15:50]

Epoch 18/100, Training Loss: 0.0905, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1155, Accuracy: 99.40%
Validation Precision: 0.9943, Recall: 0.9940, F1 Score: 0.9941
Validation ROC AUC: 1.0000, PR AUC: 0.9995, Balanced Accuracy: 0.9954
Validation Top-5 Accuracy: 1.0000
```

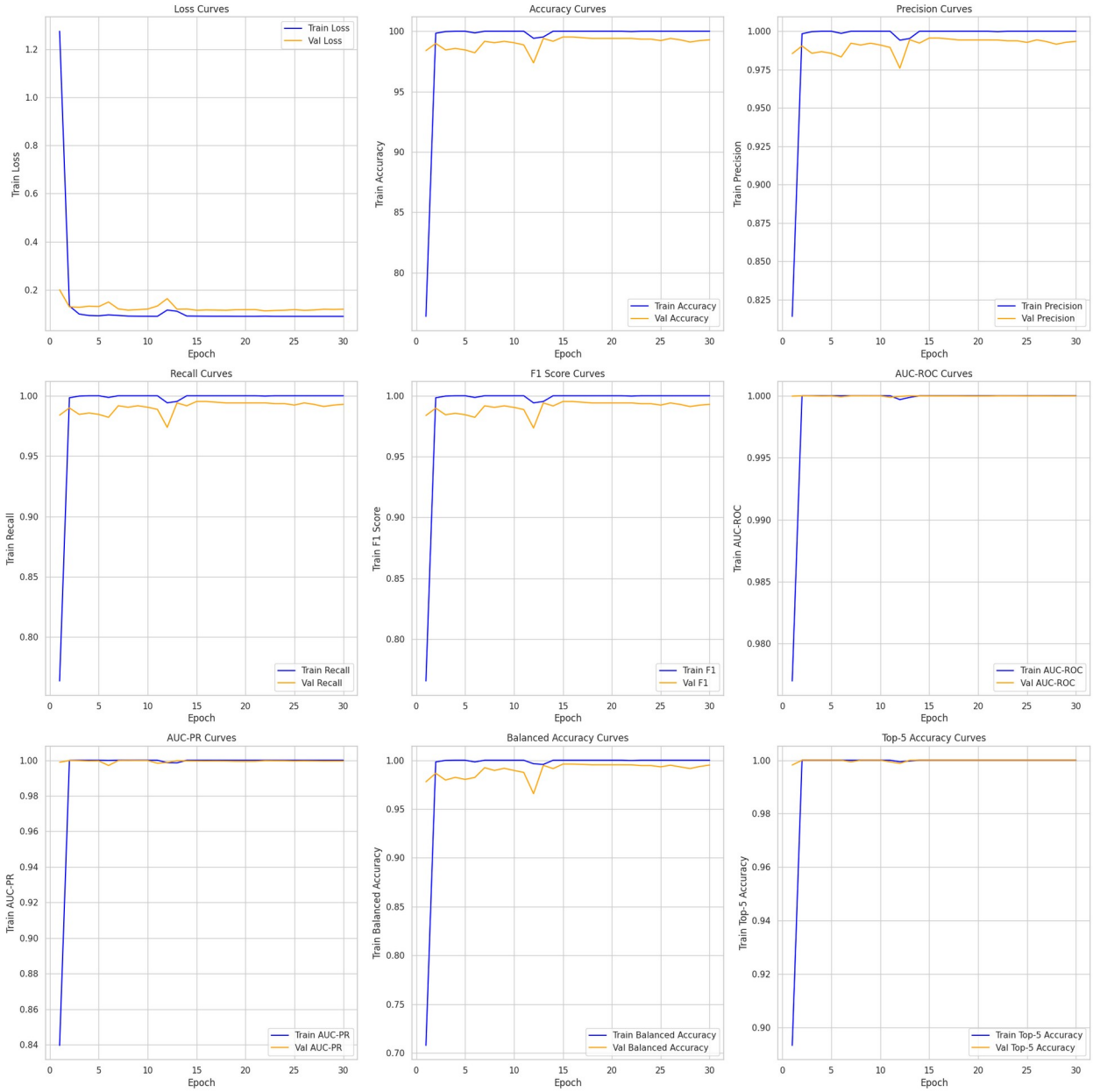
```
Epoch 19/100: 100%|██████████| 225/225 [00:49<00:00,  4.51it/s,
Loss=0.0902, Accuracy=100.00%, Elapsed=00:17:33, Remaining=01:14:49]

Epoch 19/100, Training Loss: 0.0902, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1176, Accuracy: 99.40%
Validation Precision: 0.9943, Recall: 0.9940, F1 Score: 0.9941
Validation ROC AUC: 1.0000, PR AUC: 0.9994, Balanced Accuracy: 0.9954
Validation Top-5 Accuracy: 1.0000

Epoch 20/100: 100%|██████████| 225/225 [00:50<00:00,  4.49it/s,
Loss=0.0902, Accuracy=100.00%, Elapsed=00:18:27, Remaining=01:13:51]

Epoch 20/100, Training Loss: 0.0902, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1180, Accuracy: 99.40%
Validation Precision: 0.9943, Recall: 0.9940, F1 Score: 0.9941
Validation ROC AUC: 1.0000, PR AUC: 0.9994, Balanced Accuracy: 0.9954
Validation Top-5 Accuracy: 1.0000
Average Loss Change: 0.003074, Average Accuracy Change: 0.099206

Epoch 21/100: 100%|██████████| 225/225 [00:49<00:00,  4.52it/s,
Loss=0.0902, Accuracy=100.00%, Elapsed=00:19:22, Remaining=01:12:51]

Epoch 21/100, Training Loss: 0.0902, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1182, Accuracy: 99.40%
Validation Precision: 0.9943, Recall: 0.9940, F1 Score: 0.9941
Validation ROC AUC: 1.0000, PR AUC: 0.9994, Balanced Accuracy: 0.9954
Validation Top-5 Accuracy: 1.0000

Epoch 22/100: 100%|██████████| 225/225 [00:51<00:00,  4.40it/s,
Loss=0.0908, Accuracy=99.97%, Elapsed=00:20:17, Remaining=01:11:57]
```

```
Epoch 22/100, Training Loss: 0.0907, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9997
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1126, Accuracy: 99.40%
Validation Precision: 0.9943, Recall: 0.9940, F1 Score: 0.9941
Validation ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9953
Validation Top-5 Accuracy: 1.0000

Epoch 23/100: 100%|██████████| 225/225 [00:52<00:00,  4.25it/s,
Loss=0.0901, Accuracy=100.00%, Elapsed=00:21:15, Remaining=01:11:09]

Epoch 23/100, Training Loss: 0.0901, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1145, Accuracy: 99.35%
Validation Precision: 0.9937, Recall: 0.9935, F1 Score: 0.9935
Validation ROC AUC: 1.0000, PR AUC: 0.9997, Balanced Accuracy: 0.9945
Validation Top-5 Accuracy: 1.0000

Epoch 24/100: 100%|██████████| 225/225 [00:52<00:00,  4.32it/s,
Loss=0.0900, Accuracy=100.00%, Elapsed=00:22:12, Remaining=01:10:18]

Epoch 24/100, Training Loss: 0.0900, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1156, Accuracy: 99.35%
Validation Precision: 0.9937, Recall: 0.9935, F1 Score: 0.9935
Validation ROC AUC: 1.0000, PR AUC: 0.9997, Balanced Accuracy: 0.9945
Validation Top-5 Accuracy: 1.0000

Epoch 25/100: 100%|██████████| 225/225 [00:50<00:00,  4.47it/s,
Loss=0.0900, Accuracy=100.00%, Elapsed=00:23:06, Remaining=01:09:20]

Epoch 25/100, Training Loss: 0.0900, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
```

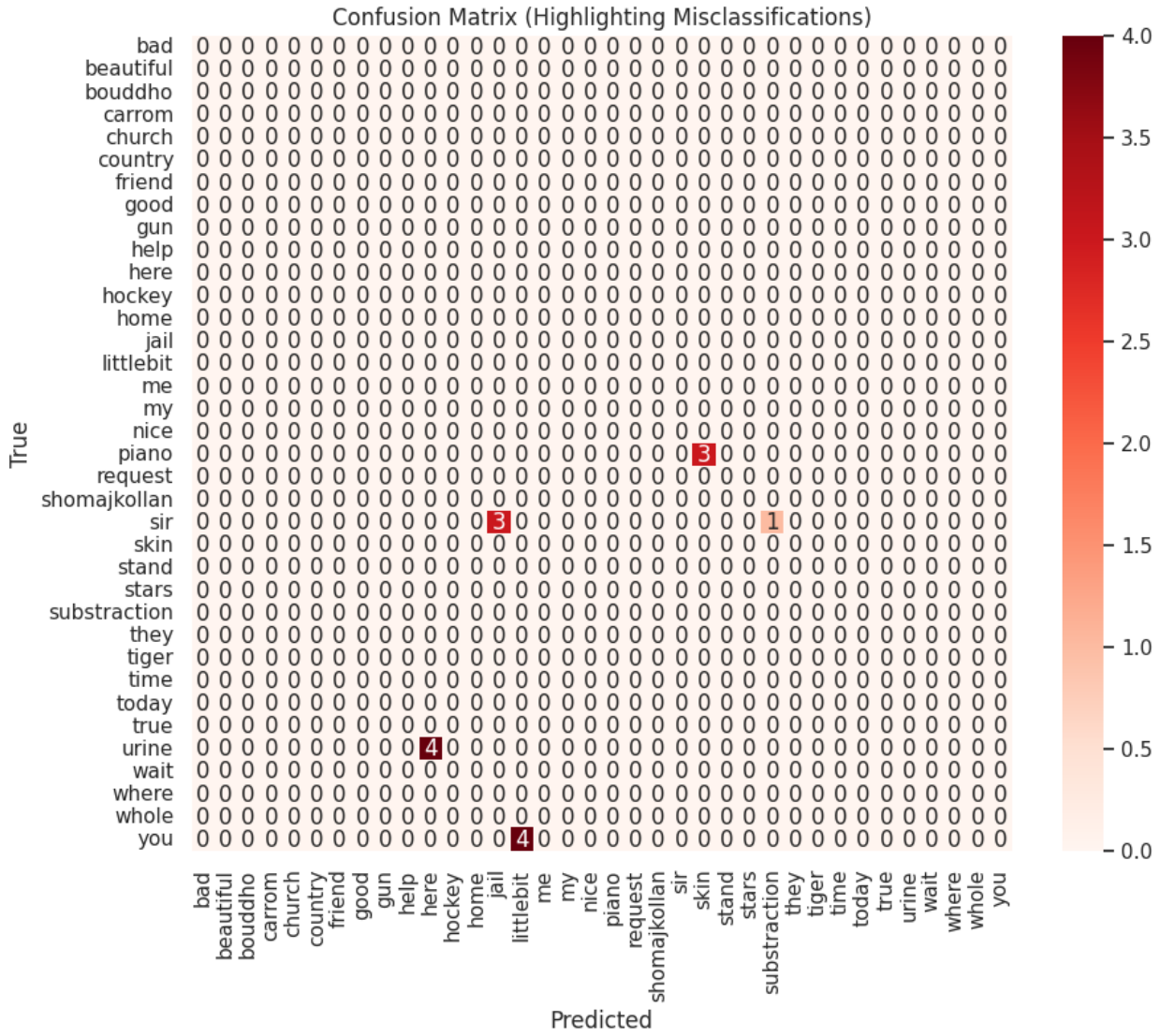
```
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1181, Accuracy: 99.23%
Validation Precision: 0.9927, Recall: 0.9923, F1 Score: 0.9923
Validation ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9932
Validation Top-5 Accuracy: 1.0000
Average Loss Change: 0.000618, Average Accuracy Change: -0.029762

Epoch 26/100: 100%|██████████| 225/225 [00:50<00:00,  4.49it/s,
Loss=0.0900, Accuracy=100.00%, Elapsed=00:24:01, Remaining=01:08:23]

Epoch 26/100, Training Loss: 0.0900, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1150, Accuracy: 99.40%
Validation Precision: 0.9944, Recall: 0.9940, F1 Score: 0.9941
Validation ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9949
Validation Top-5 Accuracy: 1.0000

Epoch 27/100: 100%|██████████| 225/225 [00:50<00:00,  4.46it/s,
Loss=0.0899, Accuracy=100.00%, Elapsed=00:24:56, Remaining=01:07:26]

Epoch 27/100, Training Loss: 0.0899, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1167, Accuracy: 99.29%
Validation Precision: 0.9932, Recall: 0.9929, F1 Score: 0.9928
Validation ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9932
Validation Top-5 Accuracy: 1.0000

Epoch 28/100: 100%|██████████| 225/225 [00:51<00:00,  4.34it/s,
Loss=0.0899, Accuracy=100.00%, Elapsed=00:25:53, Remaining=01:06:33]

Epoch 28/100, Training Loss: 0.0899, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]
```

```
Validation Loss: 0.1195, Accuracy: 99.11%
Validation Precision: 0.9915, Recall: 0.9911, F1 Score: 0.9911
Validation ROC AUC: 1.0000, PR AUC: 0.9995, Balanced Accuracy: 0.9915
Validation Top-5 Accuracy: 1.0000

Epoch 29/100: 100%|██████████| 225/225 [00:52<00:00,  4.25it/s,
Loss=0.0899, Accuracy=100.00%, Elapsed=00:26:50, Remaining=01:05:43]

Epoch 29/100, Training Loss: 0.0899, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.1764899999999996e-05]



Validation Loss: 0.1188, Accuracy: 99.23%
Validation Precision: 0.9928, Recall: 0.9923, F1 Score: 0.9923
Validation ROC AUC: 1.0000, PR AUC: 0.9995, Balanced Accuracy: 0.9936
Validation Top-5 Accuracy: 1.0000

Epoch 30/100: 100%|██████████| 225/225 [00:52<00:00,  4.26it/s,
Loss=0.0899, Accuracy=100.00%, Elapsed=00:27:48, Remaining=01:04:52]

Epoch 30/100, Training Loss: 0.0899, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.1764899999999996e-05]



Validation Loss: 0.1196, Accuracy: 99.29%
Validation Precision: 0.9934, Recall: 0.9929, F1 Score: 0.9929
Validation ROC AUC: 1.0000, PR AUC: 0.9995, Balanced Accuracy: 0.9951
Validation Top-5 Accuracy: 1.0000
Average Loss Change: -0.000596, Average Accuracy Change: -0.019841
Early stopping at epoch 30

<ipython-input-4-ffb78043919f>:294: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
```

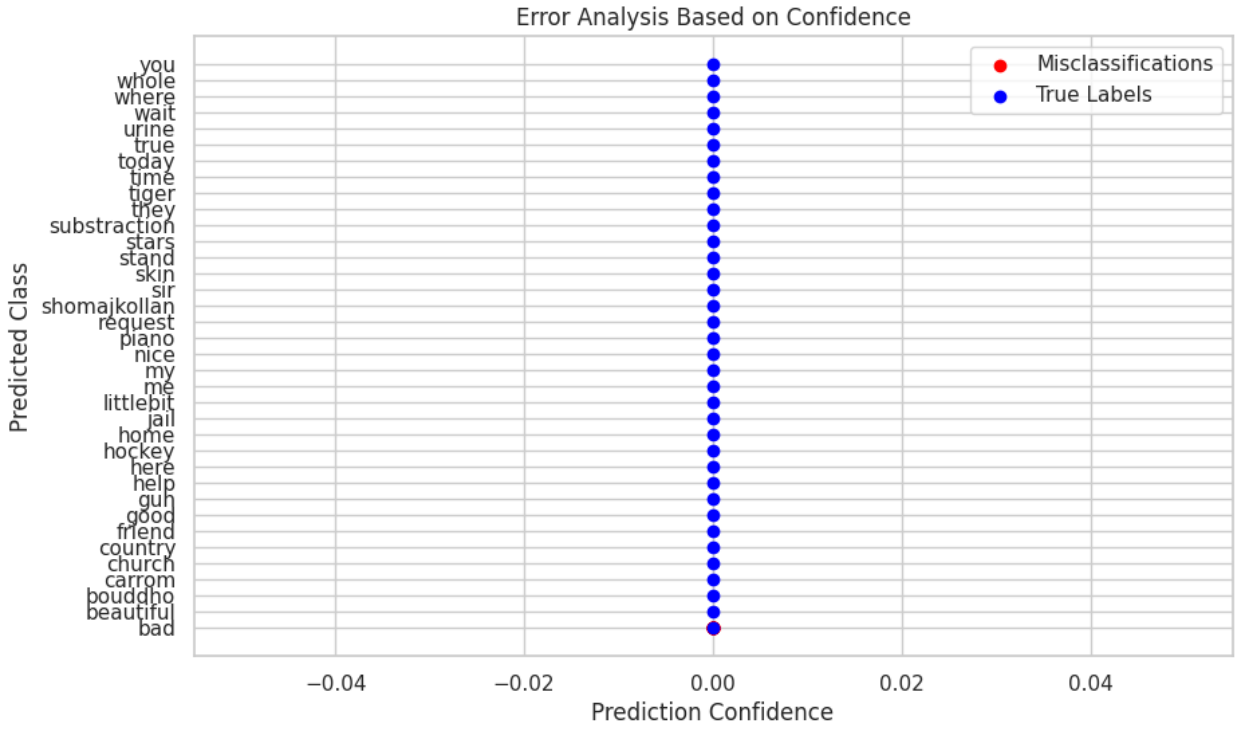
```
`weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  saved_state = torch.load(model_path, map_location=device)

Test Loss: 0.1734, Test Accuracy: 98.14%
Test Precision: 0.9848, Recall: 0.9814, F1 Score: 0.9811
Test ROC AUC: 0.9998, PR AUC:0.9966 Balanced Accuracy: 0.9740
Test Top-5 Accuracy: 1.0000
```




```
Confusion Matrix:
```




```
Error Analysis:
```




```
Training and testing DenseNet161 model...
Training and testing DenseNet161 model...

Epoch 1/100: 100%|██████████| 113/113 [00:22<00:00,  5.04it/s,
Loss=1.4031, Accuracy=75.06%, Elapsed=00:00:22, Remaining=00:36:20]

Epoch 1/100, Training Loss: 1.3849, Accuracy: 75.31%
Training Precision: 0.8101, Recall: 0.7531, F1 Score: 0.7525
Training ROC AUC: 0.9770, PR AUC: 0.8388, Balanced Accuracy: 0.6895
Training Top-5 Accuracy: 0.8924
Learning Rate: [0.0001]



Validation Loss: 0.3797, Accuracy: 96.49%
Validation Precision: 0.9674, Recall: 0.9649, F1 Score: 0.9639
Validation ROC AUC: 0.9998, PR AUC: 0.9953, Balanced Accuracy: 0.9602
Validation Top-5 Accuracy: 0.9976

Epoch 2/100: 100%|██████████| 113/113 [00:21<00:00,  5.18it/s,
Loss=0.1708, Accuracy=99.66%, Elapsed=00:00:46, Remaining=00:38:09]

Epoch 2/100, Training Loss: 0.1704, Accuracy: 99.67%
Training Precision: 0.9967, Recall: 0.9967, F1 Score: 0.9967
Training ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9958
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]
```

```
Validation Loss: 0.2033, Accuracy: 97.68%
Validation Precision: 0.9792, Recall: 0.9768, F1 Score: 0.9767
Validation ROC AUC: 0.9998, PR AUC: 0.9965, Balanced Accuracy: 0.9766
Validation Top-5 Accuracy: 0.9964

Epoch 3/100: 100%|██████████| 113/113 [00:21<00:00,  5.18it/s,
Loss=0.1064, Accuracy=100.00%, Elapsed=00:01:11, Remaining=00:38:27]

Epoch 3/100, Training Loss: 0.1081, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1727, Accuracy: 98.33%
Validation Precision: 0.9844, Recall: 0.9833, F1 Score: 0.9833
Validation ROC AUC: 0.9999, PR AUC: 0.9968, Balanced Accuracy: 0.9847
Validation Top-5 Accuracy: 0.9970

Epoch 4/100: 100%|██████████| 113/113 [00:21<00:00,  5.18it/s,
Loss=0.1004, Accuracy=100.00%, Elapsed=00:01:35, Remaining=00:38:22]

Epoch 4/100, Training Loss: 0.1012, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1685, Accuracy: 98.75%
Validation Precision: 0.9884, Recall: 0.9875, F1 Score: 0.9875
Validation ROC AUC: 0.9997, PR AUC: 0.9963, Balanced Accuracy: 0.9872
Validation Top-5 Accuracy: 0.9970

Epoch 5/100: 100%|██████████| 113/113 [00:21<00:00,  5.20it/s,
Loss=0.0950, Accuracy=100.00%, Elapsed=00:02:00, Remaining=00:38:06]

Epoch 5/100, Training Loss: 0.0968, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1671, Accuracy: 98.27%
Validation Precision: 0.9838, Recall: 0.9827, F1 Score: 0.9826
Validation ROC AUC: 0.9999, PR AUC: 0.9964, Balanced Accuracy: 0.9812
Validation Top-5 Accuracy: 0.9970
```

```
Epoch 6/100: 100%|██████████| 113/113 [00:21<00:00,  5.22it/s,
Loss=0.1022, Accuracy=99.89%, Elapsed=00:02:24, Remaining=00:37:41]

Epoch 6/100, Training Loss: 0.1027, Accuracy: 99.89%
Training Precision: 0.9989, Recall: 0.9989, F1 Score: 0.9989
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9991
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1971, Accuracy: 98.27%
Validation Precision: 0.9846, Recall: 0.9827, F1 Score: 0.9827
Validation ROC AUC: 0.9992, PR AUC: 0.9925, Balanced Accuracy: 0.9798
Validation Top-5 Accuracy: 0.9946

Epoch 7/100: 100%|██████████| 113/113 [00:21<00:00,  5.22it/s,
Loss=0.0945, Accuracy=100.00%, Elapsed=00:02:48, Remaining=00:37:16]

Epoch 7/100, Training Loss: 0.0957, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1880, Accuracy: 98.15%
Validation Precision: 0.9830, Recall: 0.9815, F1 Score: 0.9815
Validation ROC AUC: 0.9995, PR AUC: 0.9923, Balanced Accuracy: 0.9806
Validation Top-5 Accuracy: 0.9952

Epoch 8/100: 100%|██████████| 113/113 [00:21<00:00,  5.25it/s,
Loss=0.1008, Accuracy=99.89%, Elapsed=00:03:12, Remaining=00:36:50]

Epoch 8/100, Training Loss: 0.1034, Accuracy: 99.89%
Training Precision: 0.9989, Recall: 0.9989, F1 Score: 0.9989
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9992
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.2305, Accuracy: 96.61%
Validation Precision: 0.9677, Recall: 0.9661, F1 Score: 0.9652
Validation ROC AUC: 0.9990, PR AUC: 0.9891, Balanced Accuracy: 0.9570
Validation Top-5 Accuracy: 0.9946

Epoch 9/100: 100%|██████████| 113/113 [00:21<00:00,  5.18it/s,
Loss=0.1446, Accuracy=98.73%, Elapsed=00:03:36, Remaining=00:36:27]
```

```
Epoch 9/100, Training Loss: 0.1455, Accuracy: 98.75%
Training Precision: 0.9875, Recall: 0.9875, F1 Score: 0.9875
Training ROC AUC: 0.9999, PR AUC: 0.9981, Balanced Accuracy: 0.9883
Training Top-5 Accuracy: 0.9997
Learning Rate: [7e-05]



Validation Loss: 0.2565, Accuracy: 96.25%
Validation Precision: 0.9665, Recall: 0.9625, F1 Score: 0.9628
Validation ROC AUC: 0.9985, PR AUC: 0.9874, Balanced Accuracy: 0.9656
Validation Top-5 Accuracy: 0.9940

Epoch 10/100: 100%|██████████| 113/113 [00:21<00:00,  5.28it/s,
Loss=0.1057, Accuracy=99.69%, Elapsed=00:04:00, Remaining=00:36:01]

Epoch 10/100, Training Loss: 0.1062, Accuracy: 99.67%
Training Precision: 0.9967, Recall: 0.9967, F1 Score: 0.9967
Training ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9958
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1959, Accuracy: 97.20%
Validation Precision: 0.9744, Recall: 0.9720, F1 Score: 0.9717
Validation ROC AUC: 0.9995, PR AUC: 0.9928, Balanced Accuracy: 0.9643
Validation Top-5 Accuracy: 0.9958
Average Loss Change: 0.012298, Average Accuracy Change: -0.138889

Epoch 11/100: 100%|██████████| 113/113 [00:21<00:00,  5.18it/s,
Loss=0.0950, Accuracy=100.00%, Elapsed=00:04:24, Remaining=00:35:38]

Epoch 11/100, Training Loss: 0.0997, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1875, Accuracy: 97.74%
Validation Precision: 0.9794, Recall: 0.9774, F1 Score: 0.9773
Validation ROC AUC: 0.9991, PR AUC: 0.9931, Balanced Accuracy: 0.9756
Validation Top-5 Accuracy: 0.9946

Epoch 12/100: 100%|██████████| 113/113 [00:21<00:00,  5.21it/s,
Loss=0.0952, Accuracy=99.97%, Elapsed=00:04:48, Remaining=00:35:15]

Epoch 12/100, Training Loss: 0.0960, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9996
```

```
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1888, Accuracy: 97.98%
Validation Precision: 0.9807, Recall: 0.9798, F1 Score: 0.9796
Validation ROC AUC: 0.9992, PR AUC: 0.9911, Balanced Accuracy: 0.9764
Validation Top-5 Accuracy: 0.9958

Epoch 13/100: 100%|██████████| 113/113 [00:21<00:00,  5.23it/s,
Loss=0.0933, Accuracy=100.00%, Elapsed=00:05:12, Remaining=00:34:50]

Epoch 13/100, Training Loss: 0.0990, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9996
Training Top-5 Accuracy: 1.0000
Learning Rate: [3.4299999999999993e-05]



Validation Loss: 0.1870, Accuracy: 98.15%
Validation Precision: 0.9827, Recall: 0.9815, F1 Score: 0.9815
Validation ROC AUC: 0.9991, PR AUC: 0.9904, Balanced Accuracy: 0.9813
Validation Top-5 Accuracy: 0.9935

Epoch 14/100: 100%|██████████| 113/113 [00:21<00:00,  5.19it/s,
Loss=0.0937, Accuracy=99.97%, Elapsed=00:05:36, Remaining=00:34:27]

Epoch 14/100, Training Loss: 0.0944, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9999
Training Top-5 Accuracy: 1.0000
Learning Rate: [3.4299999999999993e-05]



Validation Loss: 0.1805, Accuracy: 98.51%
Validation Precision: 0.9858, Recall: 0.9851, F1 Score: 0.9851
Validation ROC AUC: 0.9988, PR AUC: 0.9898, Balanced Accuracy: 0.9833
Validation Top-5 Accuracy: 0.9958

Epoch 15/100: 100%|██████████| 113/113 [00:21<00:00,  5.15it/s,
Loss=0.0926, Accuracy=100.00%, Elapsed=00:06:00, Remaining=00:34:04]

Epoch 15/100, Training Loss: 0.0929, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [3.4299999999999993e-05]
```

```
Validation Loss: 0.1782, Accuracy: 98.33%
Validation Precision: 0.9841, Recall: 0.9833, F1 Score: 0.9833
Validation ROC AUC: 0.9988, PR AUC: 0.9904, Balanced Accuracy: 0.9819
Validation Top-5 Accuracy: 0.9946
Average Loss Change: -0.001246, Average Accuracy Change: 0.124717

Epoch 16/100: 100%|██████████| 113/113 [00:21<00:00,  5.19it/s,
Loss=0.0922, Accuracy=100.00%, Elapsed=00:06:24, Remaining=00:33:41]

Epoch 16/100, Training Loss: 0.0925, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [3.4299999999999993e-05]



Validation Loss: 0.1783, Accuracy: 98.21%
Validation Precision: 0.9832, Recall: 0.9821, F1 Score: 0.9820
Validation ROC AUC: 0.9988, PR AUC: 0.9919, Balanced Accuracy: 0.9788
Validation Top-5 Accuracy: 0.9952

Epoch 17/100: 100%|██████████| 113/113 [00:21<00:00,  5.24it/s,
Loss=0.0919, Accuracy=100.00%, Elapsed=00:06:48, Remaining=00:33:16]

Epoch 17/100, Training Loss: 0.0937, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [3.4299999999999993e-05]



Validation Loss: 0.1822, Accuracy: 98.27%
Validation Precision: 0.9842, Recall: 0.9827, F1 Score: 0.9827
Validation ROC AUC: 0.9982, PR AUC: 0.9920, Balanced Accuracy: 0.9810
Validation Top-5 Accuracy: 0.9935

Epoch 18/100: 100%|██████████| 113/113 [00:21<00:00,  5.17it/s,
Loss=0.0921, Accuracy=100.00%, Elapsed=00:07:13, Remaining=00:32:53]

Epoch 18/100, Training Loss: 0.0941, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1822, Accuracy: 98.21%
Validation Precision: 0.9834, Recall: 0.9821, F1 Score: 0.9822
```

```
Validation ROC AUC: 0.9987, PR AUC: 0.9917, Balanced Accuracy: 0.9809
Validation Top-5 Accuracy: 0.9940

Epoch 19/100: 100%|██████████| 113/113 [00:21<00:00,  5.25it/s,
Loss=0.0925, Accuracy=100.00%, Elapsed=00:07:37, Remaining=00:32:28]

Epoch 19/100, Training Loss: 0.0930, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1794, Accuracy: 98.21%
Validation Precision: 0.9836, Recall: 0.9821, F1 Score: 0.9822
Validation ROC AUC: 0.9985, PR AUC: 0.9908, Balanced Accuracy: 0.9800
Validation Top-5 Accuracy: 0.9958

Epoch 20/100: 100%|██████████| 113/113 [00:21<00:00,  5.23it/s,
Loss=0.0919, Accuracy=100.00%, Elapsed=00:08:00, Remaining=00:32:03]

Epoch 20/100, Training Loss: 0.0928, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1769, Accuracy: 98.21%
Validation Precision: 0.9833, Recall: 0.9821, F1 Score: 0.9821
Validation ROC AUC: 0.9984, PR AUC: 0.9917, Balanced Accuracy: 0.9820
Validation Top-5 Accuracy: 0.9946
Average Loss Change: 0.004510, Average Accuracy Change: 0.069444

Epoch 21/100: 100%|██████████| 113/113 [00:21<00:00,  5.18it/s,
Loss=0.0918, Accuracy=100.00%, Elapsed=00:08:25, Remaining=00:31:40]

Epoch 21/100, Training Loss: 0.0921, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1700, Accuracy: 98.63%
Validation Precision: 0.9874, Recall: 0.9863, F1 Score: 0.9863
Validation ROC AUC: 0.9985, PR AUC: 0.9915, Balanced Accuracy: 0.9854
Validation Top-5 Accuracy: 0.9958
```

```
Epoch 22/100: 100%|██████████| 113/113 [00:21<00:00,  5.19it/s,
Loss=0.0916, Accuracy=100.00%, Elapsed=00:08:49, Remaining=00:31:16]

Epoch 22/100, Training Loss: 0.0918, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1780, Accuracy: 98.27%
Validation Precision: 0.9840, Recall: 0.9827, F1 Score: 0.9827
Validation ROC AUC: 0.9984, PR AUC: 0.9906, Balanced Accuracy: 0.9836
Validation Top-5 Accuracy: 0.9958

Epoch 23/100: 100%|██████████| 113/113 [00:21<00:00,  5.18it/s,
Loss=0.0914, Accuracy=100.00%, Elapsed=00:09:13, Remaining=00:30:53]

Epoch 23/100, Training Loss: 0.0918, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1774, Accuracy: 98.15%
Validation Precision: 0.9830, Recall: 0.9815, F1 Score: 0.9816
Validation ROC AUC: 0.9988, PR AUC: 0.9923, Balanced Accuracy: 0.9818
Validation Top-5 Accuracy: 0.9958

Epoch 24/100: 100%|██████████| 113/113 [00:21<00:00,  5.27it/s,
Loss=0.0912, Accuracy=100.00%, Elapsed=00:09:37, Remaining=00:30:28]

Epoch 24/100, Training Loss: 0.0937, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.1764899999999996e-05]



Validation Loss: 0.1875, Accuracy: 97.92%
Validation Precision: 0.9811, Recall: 0.9792, F1 Score: 0.9792
Validation ROC AUC: 0.9986, PR AUC: 0.9922, Balanced Accuracy: 0.9789
Validation Top-5 Accuracy: 0.9952

Epoch 25/100: 100%|██████████| 113/113 [00:21<00:00,  5.20it/s,
Loss=0.0915, Accuracy=100.00%, Elapsed=00:10:01, Remaining=00:30:04]
```

```
Epoch 25/100, Training Loss: 0.0919, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.1764899999999996e-05]



Validation Loss: 0.1813, Accuracy: 98.10%
Validation Precision: 0.9827, Recall: 0.9810, F1 Score: 0.9810
Validation ROC AUC: 0.9985, PR AUC: 0.9911, Balanced Accuracy: 0.9810
Validation Top-5 Accuracy: 0.9964
Average Loss Change: 0.003176, Average Accuracy Change: 0.079365

Epoch 26/100: 100%|██████████| 113/113 [00:21<00:00,  5.26it/s,
Loss=0.0913, Accuracy=100.00%, Elapsed=00:10:25, Remaining=00:29:39]

Epoch 26/100, Training Loss: 0.0926, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [8.235429999999996e-06]



Validation Loss: 0.1740, Accuracy: 98.51%
Validation Precision: 0.9861, Recall: 0.9851, F1 Score: 0.9851
Validation ROC AUC: 0.9983, PR AUC: 0.9912, Balanced Accuracy: 0.9853
Validation Top-5 Accuracy: 0.9958

Epoch 27/100: 100%|██████████| 113/113 [00:21<00:00,  5.21it/s,
Loss=0.0913, Accuracy=100.00%, Elapsed=00:10:49, Remaining=00:29:15]

Epoch 27/100, Training Loss: 0.0921, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [8.235429999999996e-06]



Validation Loss: 0.1879, Accuracy: 98.10%
Validation Precision: 0.9827, Recall: 0.9810, F1 Score: 0.9811
Validation ROC AUC: 0.9981, PR AUC: 0.9901, Balanced Accuracy: 0.9816
Validation Top-5 Accuracy: 0.9964

Epoch 28/100: 100%|██████████| 113/113 [00:21<00:00,  5.15it/s,
Loss=0.0913, Accuracy=100.00%, Elapsed=00:11:13, Remaining=00:28:52]

Epoch 28/100, Training Loss: 0.0917, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
```

```
Training Top-5 Accuracy: 1.0000
Learning Rate: [8.235429999999996e-06]



Validation Loss: 0.1898, Accuracy: 97.92%
Validation Precision: 0.9811, Recall: 0.9792, F1 Score: 0.9792
Validation ROC AUC: 0.9983, PR AUC: 0.9909, Balanced Accuracy: 0.9784
Validation Top-5 Accuracy: 0.9964

Epoch 29/100: 100%|██████████| 113/113 [00:21<00:00,  5.19it/s,
Loss=0.0911, Accuracy=100.00%, Elapsed=00:11:37, Remaining=00:28:28]

Epoch 29/100, Training Loss: 0.0914, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [8.235429999999996e-06]



Validation Loss: 0.1837, Accuracy: 98.04%
Validation Precision: 0.9821, Recall: 0.9804, F1 Score: 0.9804
Validation ROC AUC: 0.9984, PR AUC: 0.9911, Balanced Accuracy: 0.9807
Validation Top-5 Accuracy: 0.9964

Epoch 30/100: 100%|██████████| 113/113 [00:21<00:00,  5.17it/s,
Loss=0.0911, Accuracy=100.00%, Elapsed=00:12:02, Remaining=00:28:04]

Epoch 30/100, Training Loss: 0.0919, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [8.235429999999996e-06]



Validation Loss: 0.1796, Accuracy: 98.10%
Validation Precision: 0.9826, Recall: 0.9810, F1 Score: 0.9810
Validation ROC AUC: 0.9982, PR AUC: 0.9907, Balanced Accuracy: 0.9816
Validation Top-5 Accuracy: 0.9958
Average Loss Change: -0.000825, Average Accuracy Change: 0.059524

Epoch 31/100: 100%|██████████| 113/113 [00:21<00:00,  5.16it/s,
Loss=0.0914, Accuracy=100.00%, Elapsed=00:12:26, Remaining=00:27:40]

Epoch 31/100, Training Loss: 0.0916, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [5.764800999999997e-06]
```

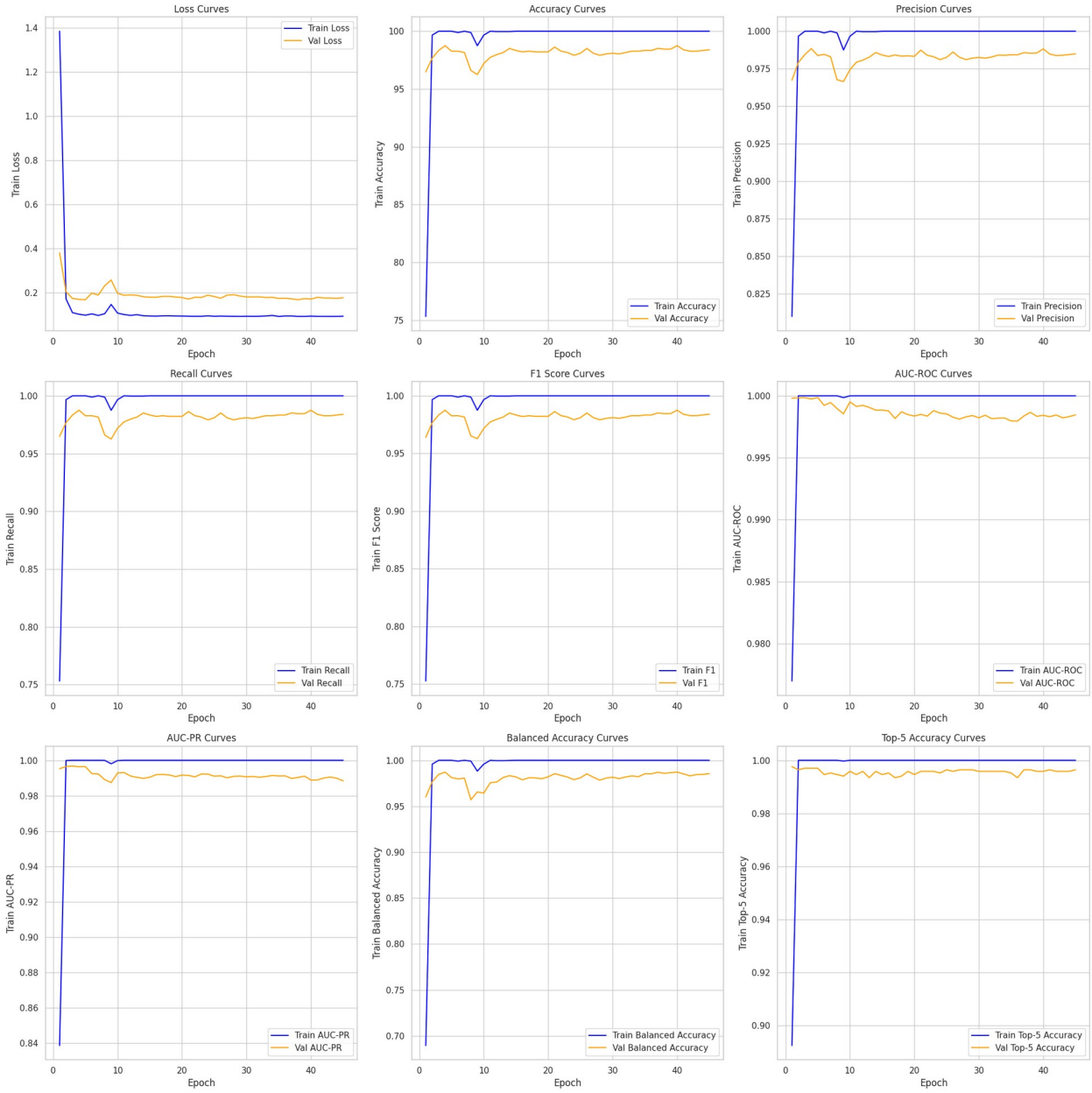
```
Validation Loss: 0.1797, Accuracy: 98.04%
Validation Precision: 0.9821, Recall: 0.9804, F1 Score: 0.9805
Validation ROC AUC: 0.9984, PR AUC: 0.9909, Balanced Accuracy: 0.9802
Validation Top-5 Accuracy: 0.9958

Epoch 32/100: 100%|██████████| 113/113 [00:21<00:00,  5.18it/s,
Loss=0.0911, Accuracy=100.00%, Elapsed=00:12:50, Remaining=00:27:17]

Epoch 32/100, Training Loss: 0.0919, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [5.764800999999997e-06]



Validation Loss: 0.1800, Accuracy: 98.15%
Validation Precision: 0.9828, Recall: 0.9815, F1 Score: 0.9815
Validation ROC AUC: 0.9982, PR AUC: 0.9905, Balanced Accuracy: 0.9819
Validation Top-5 Accuracy: 0.9958

Epoch 33/100: 100%|██████████| 113/113 [00:21<00:00,  5.22it/s,
Loss=0.0910, Accuracy=100.00%, Elapsed=00:13:14, Remaining=00:26:52]

Epoch 33/100, Training Loss: 0.0932, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.035360699999998e-06]



Validation Loss: 0.1770, Accuracy: 98.27%
Validation Precision: 0.9841, Recall: 0.9827, F1 Score: 0.9828
Validation ROC AUC: 0.9982, PR AUC: 0.9909, Balanced Accuracy: 0.9830
Validation Top-5 Accuracy: 0.9958

Epoch 34/100: 100%|██████████| 113/113 [00:21<00:00,  5.19it/s,
Loss=0.0911, Accuracy=100.00%, Elapsed=00:13:38, Remaining=00:26:28]

Epoch 34/100, Training Loss: 0.0957, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.035360699999998e-06]



Validation Loss: 0.1778, Accuracy: 98.27%
Validation Precision: 0.9840, Recall: 0.9827, F1 Score: 0.9827
Validation ROC AUC: 0.9982, PR AUC: 0.9914, Balanced Accuracy: 0.9822
Validation Top-5 Accuracy: 0.9958
```

```
Epoch 35/100: 100%|██████████| 113/113 [00:21<00:00,  5.18it/s,
Loss=0.0910, Accuracy=100.00%, Elapsed=00:14:02, Remaining=00:26:04]

Epoch 35/100, Training Loss: 0.0912, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.035360699999998e-06]



Validation Loss: 0.1731, Accuracy: 98.33%
Validation Precision: 0.9843, Recall: 0.9833, F1 Score: 0.9833
Validation ROC AUC: 0.9980, PR AUC: 0.9912, Balanced Accuracy: 0.9854
Validation Top-5 Accuracy: 0.9952
Average Loss Change: 0.000553, Average Accuracy Change: -0.019841

Epoch 36/100: 100%|██████████| 113/113 [00:21<00:00,  5.22it/s,
Loss=0.0909, Accuracy=100.00%, Elapsed=00:14:26, Remaining=00:25:40]

Epoch 36/100, Training Loss: 0.0931, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.035360699999998e-06]



Validation Loss: 0.1739, Accuracy: 98.33%
Validation Precision: 0.9843, Recall: 0.9833, F1 Score: 0.9833
Validation ROC AUC: 0.9980, PR AUC: 0.9912, Balanced Accuracy: 0.9854
Validation Top-5 Accuracy: 0.9935

Epoch 37/100: 100%|██████████| 113/113 [00:21<00:00,  5.15it/s,
Loss=0.0909, Accuracy=100.00%, Elapsed=00:14:50, Remaining=00:25:17]

Epoch 37/100, Training Loss: 0.0931, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.8247524899999986e-06]



Validation Loss: 0.1708, Accuracy: 98.51%
Validation Precision: 0.9858, Recall: 0.9851, F1 Score: 0.9851
Validation ROC AUC: 0.9984, PR AUC: 0.9898, Balanced Accuracy: 0.9870
Validation Top-5 Accuracy: 0.9964

Epoch 38/100: 100%|██████████| 113/113 [00:21<00:00,  5.14it/s,
Loss=0.0909, Accuracy=100.00%, Elapsed=00:15:15, Remaining=00:24:53]
```

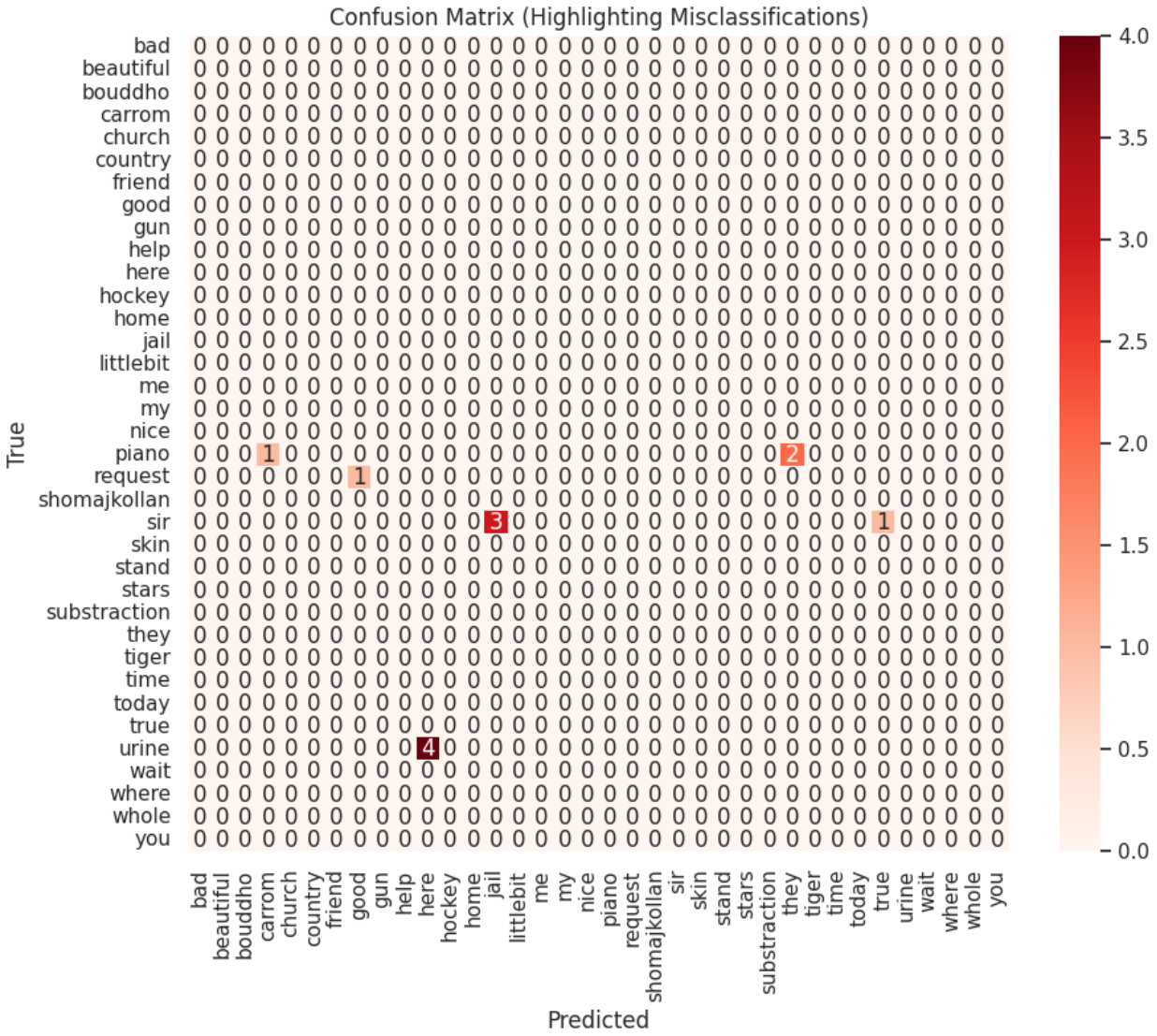
```
Epoch 38/100, Training Loss: 0.0912, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.8247524899999986e-06]



Validation Loss: 0.1670, Accuracy: 98.45%
Validation Precision: 0.9854, Recall: 0.9845, F1 Score: 0.9845
Validation ROC AUC: 0.9987, PR AUC: 0.9903, Balanced Accuracy: 0.9858
Validation Top-5 Accuracy: 0.9964

Epoch 39/100: 100%|██████████| 113/113 [00:21<00:00,  5.25it/s,
Loss=0.0909, Accuracy=100.00%, Elapsed=00:15:39, Remaining=00:24:28]

Epoch 39/100, Training Loss: 0.0913, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.977326742999999e-06]



Validation Loss: 0.1720, Accuracy: 98.45%
Validation Precision: 0.9855, Recall: 0.9845, F1 Score: 0.9845
Validation ROC AUC: 0.9983, PR AUC: 0.9911, Balanced Accuracy: 0.9866
Validation Top-5 Accuracy: 0.9958

Epoch 40/100: 100%|██████████| 113/113 [00:21<00:00,  5.21it/s,
Loss=0.0909, Accuracy=100.00%, Elapsed=00:16:03, Remaining=00:24:04]

Epoch 40/100, Training Loss: 0.0925, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.977326742999999e-06]



Validation Loss: 0.1700, Accuracy: 98.75%
Validation Precision: 0.9883, Recall: 0.9875, F1 Score: 0.9875
Validation ROC AUC: 0.9984, PR AUC: 0.9888, Balanced Accuracy: 0.9872
Validation Top-5 Accuracy: 0.9958
Average Loss Change: 0.004823, Average Accuracy Change: 0.099206

Epoch 41/100: 100%|██████████| 113/113 [00:21<00:00,  5.16it/s,
Loss=0.0911, Accuracy=100.00%, Elapsed=00:16:27, Remaining=00:23:41]

Epoch 41/100, Training Loss: 0.0914, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
```

```
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.3841287200999992e-06]



Validation Loss: 0.1777, Accuracy: 98.39%
Validation Precision: 0.9848, Recall: 0.9839, F1 Score: 0.9839
Validation ROC AUC: 0.9983, PR AUC: 0.9889, Balanced Accuracy: 0.9852
Validation Top-5 Accuracy: 0.9964

Epoch 42/100: 100%|██████████| 113/113 [00:22<00:00,  5.13it/s,
Loss=0.0909, Accuracy=100.00%, Elapsed=00:16:51, Remaining=00:23:17]

Epoch 42/100, Training Loss: 0.0914, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.3841287200999992e-06]



Validation Loss: 0.1750, Accuracy: 98.27%
Validation Precision: 0.9838, Recall: 0.9827, F1 Score: 0.9828
Validation ROC AUC: 0.9985, PR AUC: 0.9900, Balanced Accuracy: 0.9831
Validation Top-5 Accuracy: 0.9958

Epoch 43/100: 100%|██████████| 113/113 [00:21<00:00,  5.16it/s,
Loss=0.0909, Accuracy=100.00%, Elapsed=00:17:16, Remaining=00:22:53]

Epoch 43/100, Training Loss: 0.0911, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.3841287200999992e-06]



Validation Loss: 0.1748, Accuracy: 98.27%
Validation Precision: 0.9840, Recall: 0.9827, F1 Score: 0.9828
Validation ROC AUC: 0.9982, PR AUC: 0.9906, Balanced Accuracy: 0.9845
Validation Top-5 Accuracy: 0.9958

Epoch 44/100: 100%|██████████| 113/113 [00:22<00:00,  5.13it/s,
Loss=0.0909, Accuracy=100.00%, Elapsed=00:17:40, Remaining=00:22:29]

Epoch 44/100, Training Loss: 0.0913, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.3841287200999992e-06]
```

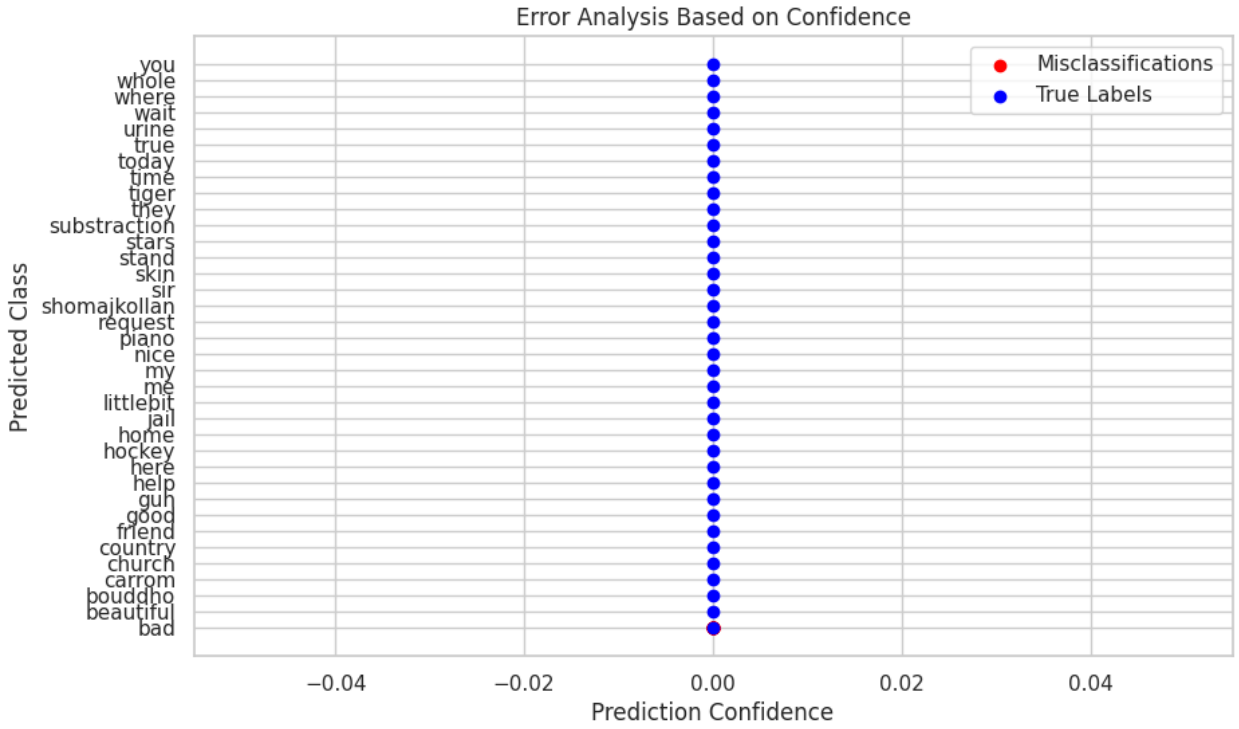
```
Validation Loss: 0.1736, Accuracy: 98.33%
Validation Precision: 0.9845, Recall: 0.9833, F1 Score: 0.9834
Validation ROC AUC: 0.9983, PR AUC: 0.9899, Balanced Accuracy: 0.9847
Validation Top-5 Accuracy: 0.9958

Epoch 45/100: 100%|██████████| 113/113 [00:21<00:00,  5.18it/s,
Loss=0.0909, Accuracy=100.00%, Elapsed=00:18:04, Remaining=00:22:05]

Epoch 45/100, Training Loss: 0.0920, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [9.688901040699995e-07]



Validation Loss: 0.1758, Accuracy: 98.39%
Validation Precision: 0.9850, Recall: 0.9839, F1 Score: 0.9840
Validation ROC AUC: 0.9985, PR AUC: 0.9884, Balanced Accuracy: 0.9855
Validation Top-5 Accuracy: 0.9964
Average Loss Change: -0.001509, Average Accuracy Change: -0.024802
Early stopping at epoch 45

<ipython-input-4-ffb78043919f>:294: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  saved_state = torch.load(model_path, map_location=device)

Test Loss: 0.1616, Test Accuracy: 98.51%
Test Precision: 0.9874, Recall: 0.9851, F1 Score: 0.9850
Test ROC AUC: 0.9999, PR AUC:0.9960 Balanced Accuracy: 0.9824
Test Top-5 Accuracy: 1.0000
```



Confusion Matrix:




```
Error Analysis:
```




```
Training and testing EfficientNet_V2_S model...
Training and testing EfficientNet_V2_S model...

Epoch 1/100: 100%|██████████| 225/225 [00:30<00:00,  7.34it/s,
Loss=1.7597, Accuracy=64.23%, Elapsed=00:00:30, Remaining=00:49:39]

Epoch 1/100, Training Loss: 1.7350, Accuracy: 64.73%
Training Precision: 0.6817, Recall: 0.6473, F1 Score: 0.6415
Training ROC AUC: 0.9495, PR AUC: 0.7037, Balanced Accuracy: 0.5743
Training Top-5 Accuracy: 0.8253
Learning Rate: [0.0001]



Validation Loss: 378.1934, Accuracy: 95.71%
Validation Precision: 0.9602, Recall: 0.9571, F1 Score: 0.9566
Validation ROC AUC: 0.9902, PR AUC: 0.9651, Balanced Accuracy: 0.9494
Validation Top-5 Accuracy: 0.9881

Epoch 2/100: 100%|██████████| 225/225 [00:30<00:00,  7.41it/s,
Loss=0.2022, Accuracy=98.98%, Elapsed=00:01:03, Remaining=00:52:01]

Epoch 2/100, Training Loss: 0.2012, Accuracy: 98.97%
Training Precision: 0.9898, Recall: 0.9897, F1 Score: 0.9897
Training ROC AUC: 0.9999, PR AUC: 0.9986, Balanced Accuracy: 0.9883
Training Top-5 Accuracy: 0.9994
Learning Rate: [0.0001]
```

```
Validation Loss: 385.1457, Accuracy: 97.80%
Validation Precision: 0.9793, Recall: 0.9780, F1 Score: 0.9777
Validation ROC AUC: 0.9962, PR AUC: 0.9840, Balanced Accuracy: 0.9750
Validation Top-5 Accuracy: 0.9958

Epoch 3/100: 100%|██████████| 225/225 [00:30<00:00,  7.41it/s,
Loss=0.1249, Accuracy=99.55%, Elapsed=00:01:37, Remaining=00:52:20]

Epoch 3/100, Training Loss: 0.1252, Accuracy: 99.55%
Training Precision: 0.9956, Recall: 0.9955, F1 Score: 0.9955
Training ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9947
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1696, Accuracy: 98.10%
Validation Precision: 0.9819, Recall: 0.9810, F1 Score: 0.9807
Validation ROC AUC: 0.9998, PR AUC: 0.9964, Balanced Accuracy: 0.9791
Validation Top-5 Accuracy: 0.9976

Epoch 4/100: 100%|██████████| 225/225 [00:30<00:00,  7.45it/s,
Loss=0.1353, Accuracy=99.26%, Elapsed=00:02:10, Remaining=00:52:08]

Epoch 4/100, Training Loss: 0.1358, Accuracy: 99.28%
Training Precision: 0.9928, Recall: 0.9928, F1 Score: 0.9928
Training ROC AUC: 1.0000, PR AUC: 0.9991, Balanced Accuracy: 0.9934
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 6316.6117, Accuracy: 97.68%
Validation Precision: 0.9784, Recall: 0.9768, F1 Score: 0.9767
Validation ROC AUC: 0.9954, PR AUC: 0.9817, Balanced Accuracy: 0.9778
Validation Top-5 Accuracy: 0.9952

Epoch 5/100: 100%|██████████| 225/225 [00:29<00:00,  7.51it/s,
Loss=0.1142, Accuracy=99.66%, Elapsed=00:02:43, Remaining=00:51:38]

Epoch 5/100, Training Loss: 0.1149, Accuracy: 99.67%
Training Precision: 0.9967, Recall: 0.9967, F1 Score: 0.9967
Training ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9967
Training Top-5 Accuracy: 0.9997
Learning Rate: [0.0001]
Validation Loss: 1787.8988, Accuracy: 98.39%
Validation Precision: 0.9850, Recall: 0.9839, F1 Score: 0.9840
Validation ROC AUC: 0.9955, PR AUC: 0.9817, Balanced Accuracy: 0.9843
Validation Top-5 Accuracy: 0.9958

Epoch 6/100: 100%|██████████| 225/225 [00:29<00:00,  7.56it/s,
Loss=0.1088, Accuracy=99.75%, Elapsed=00:03:15, Remaining=00:51:08]
```

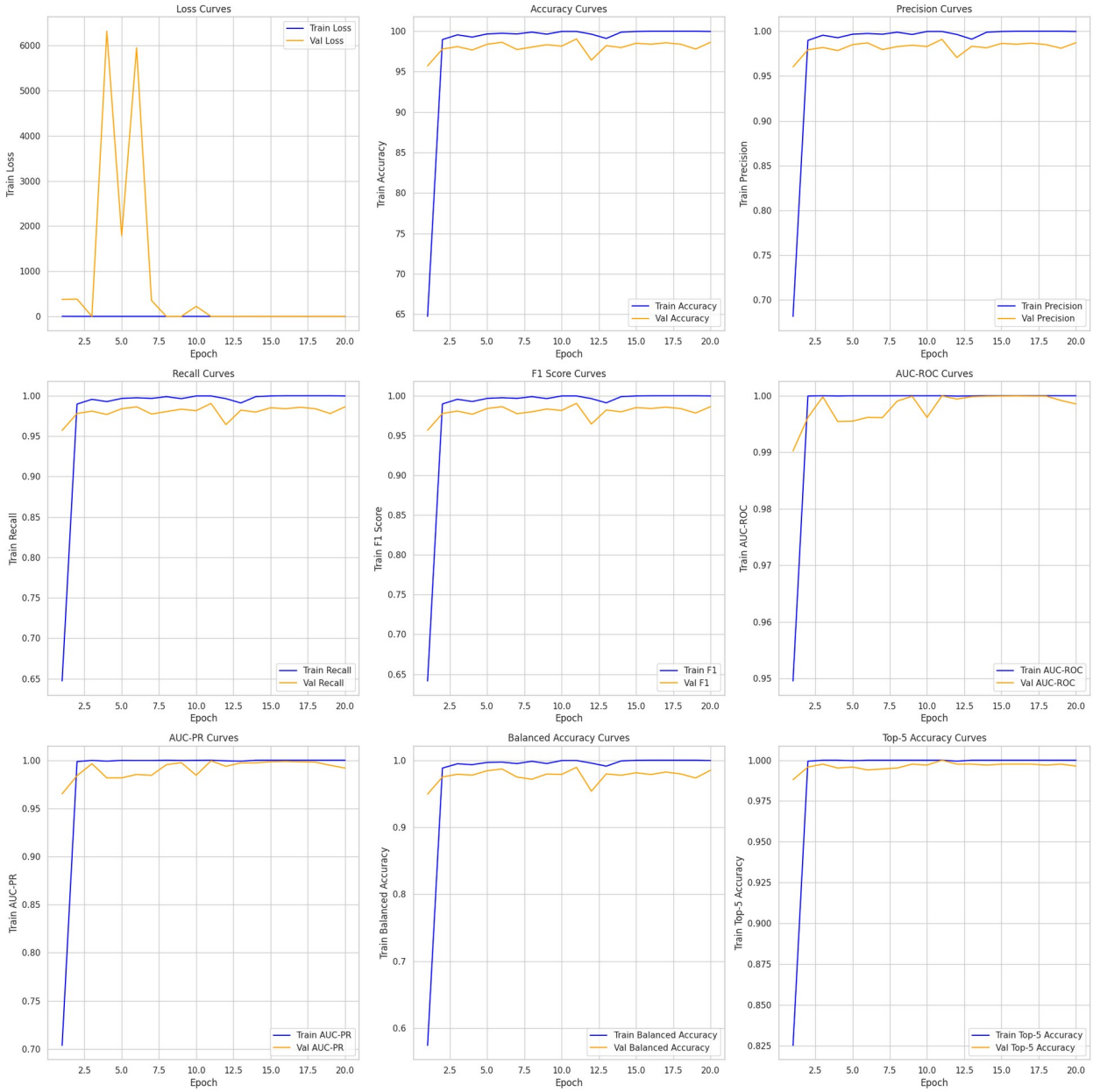
```
Epoch 6/100, Training Loss: 0.1090, Accuracy: 99.75%
Training Precision: 0.9975, Recall: 0.9975, F1 Score: 0.9975
Training ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9972
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 5949.9014, Accuracy: 98.63%
Validation Precision: 0.9870, Recall: 0.9863, F1 Score: 0.9863
Validation ROC AUC: 0.9962, PR AUC: 0.9853, Balanced Accuracy: 0.9870
Validation Top-5 Accuracy: 0.9940

Epoch 7/100: 100%|██████████| 225/225 [00:29<00:00,  7.53it/s,
Loss=0.1096, Accuracy=99.69%, Elapsed=00:03:48, Remaining=00:50:39]

Epoch 7/100, Training Loss: 0.1099, Accuracy: 99.67%
Training Precision: 0.9967, Recall: 0.9967, F1 Score: 0.9967
Training ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9950
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]
Validation Loss: 353.4436, Accuracy: 97.74%
Validation Precision: 0.9795, Recall: 0.9774, F1 Score: 0.9775
Validation ROC AUC: 0.9961, PR AUC: 0.9843, Balanced Accuracy: 0.9750
Validation Top-5 Accuracy: 0.9946

Epoch 8/100: 100%|██████████| 225/225 [00:30<00:00,  7.40it/s,
Loss=0.1013, Accuracy=99.89%, Elapsed=00:04:21, Remaining=00:50:11]

Epoch 8/100, Training Loss: 0.1013, Accuracy: 99.89%
Training Precision: 0.9989, Recall: 0.9989, F1 Score: 0.9989
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9985
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1755, Accuracy: 98.04%
Validation Precision: 0.9829, Recall: 0.9804, F1 Score: 0.9798
Validation ROC AUC: 0.9991, PR AUC: 0.9954, Balanced Accuracy: 0.9717
Validation Top-5 Accuracy: 0.9952

Epoch 9/100: 100%|██████████| 225/225 [00:30<00:00,  7.49it/s,
Loss=0.1039, Accuracy=99.69%, Elapsed=00:04:54, Remaining=00:49:39]

Epoch 9/100, Training Loss: 0.1043, Accuracy: 99.64%
Training Precision: 0.9964, Recall: 0.9964, F1 Score: 0.9964
Training ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9951
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]
Validation Loss: 0.1587, Accuracy: 98.33%
Validation Precision: 0.9844, Recall: 0.9833, F1 Score: 0.9833
```

```
Validation ROC AUC: 0.9999, PR AUC: 0.9975, Balanced Accuracy: 0.9795
Validation Top-5 Accuracy: 0.9976

Epoch 10/100: 100%|██████████| 225/225 [00:29<00:00,  7.52it/s,
Loss=0.0994, Accuracy=99.97%, Elapsed=00:05:27, Remaining=00:49:05]

Epoch 10/100, Training Loss: 0.1000, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9995
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 222.7781, Accuracy: 98.15%
Validation Precision: 0.9829, Recall: 0.9815, F1 Score: 0.9815
Validation ROC AUC: 0.9962, PR AUC: 0.9844, Balanced Accuracy: 0.9788
Validation Top-5 Accuracy: 0.9970
Average Loss Change: -1893.175891, Average Accuracy Change: 0.545635

Epoch 11/100: 100%|██████████| 225/225 [00:30<00:00,  7.40it/s,
Loss=0.0957, Accuracy=99.97%, Elapsed=00:06:00, Remaining=00:48:36]

Epoch 11/100, Training Loss: 0.0958, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9996
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1301, Accuracy: 99.05%
Validation Precision: 0.9910, Recall: 0.9905, F1 Score: 0.9904
Validation ROC AUC: 1.0000, PR AUC: 0.9994, Balanced Accuracy: 0.9894
Validation Top-5 Accuracy: 1.0000

Epoch 12/100: 100%|██████████| 225/225 [00:29<00:00,  7.52it/s,
Loss=0.1057, Accuracy=99.63%, Elapsed=00:06:33, Remaining=00:48:05]

Epoch 12/100, Training Loss: 0.1063, Accuracy: 99.64%
Training Precision: 0.9964, Recall: 0.9964, F1 Score: 0.9964
Training ROC AUC: 0.9999, PR AUC: 0.9994, Balanced Accuracy: 0.9957
Training Top-5 Accuracy: 0.9994
Learning Rate: [0.0001]
Validation Loss: 0.2221, Accuracy: 96.43%
Validation Precision: 0.9707, Recall: 0.9643, F1 Score: 0.9645
Validation ROC AUC: 0.9994, PR AUC: 0.9937, Balanced Accuracy: 0.9538
Validation Top-5 Accuracy: 0.9976

Epoch 13/100: 100%|██████████| 225/225 [00:30<00:00,  7.45it/s,
Loss=0.1288, Accuracy=99.10%, Elapsed=00:07:06, Remaining=00:47:33]
```

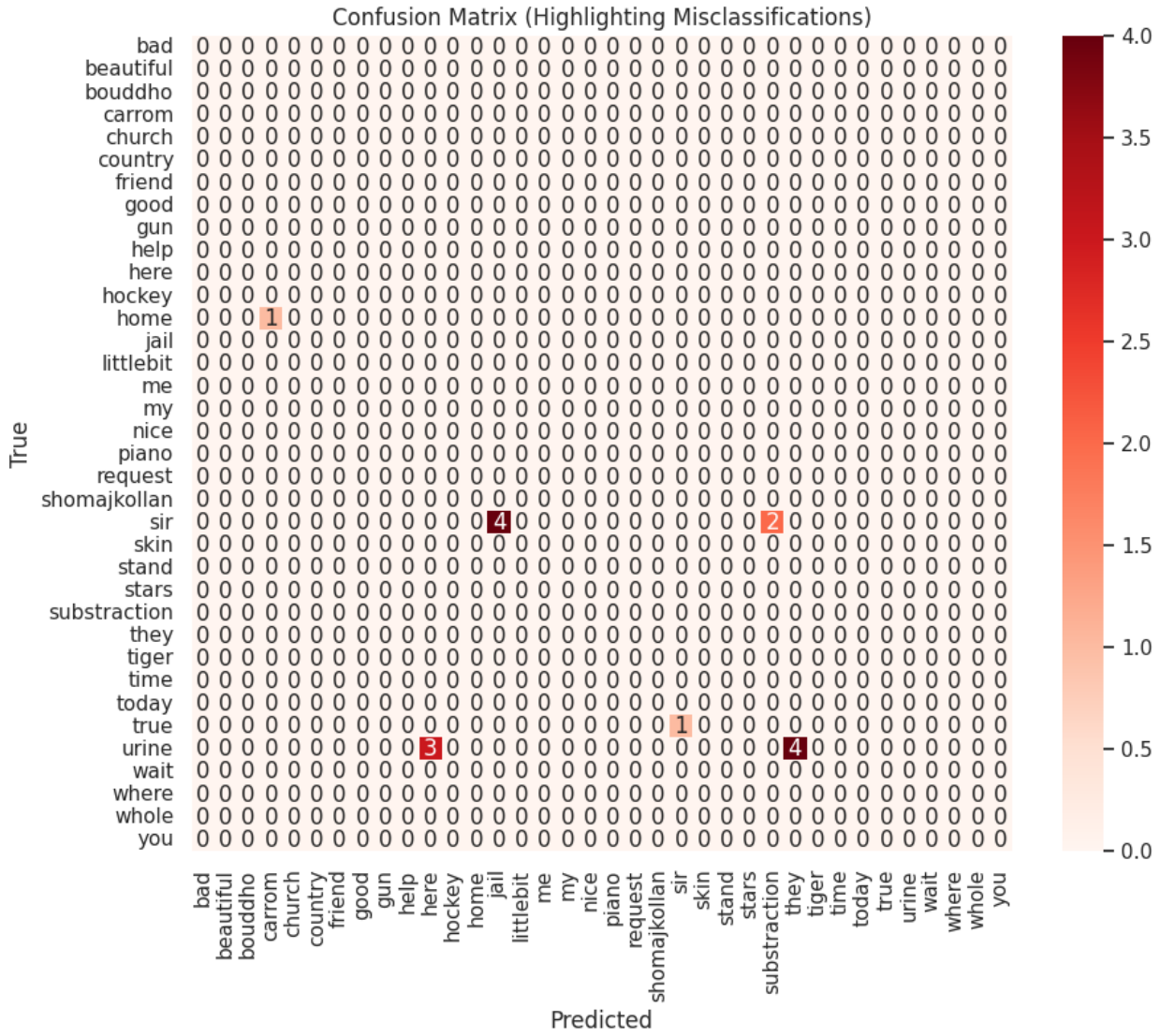
```
Epoch 13/100, Training Loss: 0.1284, Accuracy: 99.11%
Training Precision: 0.9911, Recall: 0.9911, F1 Score: 0.9911
Training ROC AUC: 1.0000, PR AUC: 0.9990, Balanced Accuracy: 0.9911
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1615, Accuracy: 98.21%
Validation Precision: 0.9832, Recall: 0.9821, F1 Score: 0.9821
Validation ROC AUC: 0.9998, PR AUC: 0.9973, Balanced Accuracy: 0.9796
Validation Top-5 Accuracy: 0.9976

Epoch 14/100: 100%|██████████| 225/225 [00:30<00:00,  7.50it/s,
Loss=0.1015, Accuracy=99.89%, Elapsed=00:07:39, Remaining=00:47:00]

Epoch 14/100, Training Loss: 0.1014, Accuracy: 99.89%
Training Precision: 0.9989, Recall: 0.9989, F1 Score: 0.9989
Training ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9990
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1631, Accuracy: 97.98%
Validation Precision: 0.9815, Recall: 0.9798, F1 Score: 0.9798
Validation ROC AUC: 0.9999, PR AUC: 0.9973, Balanced Accuracy: 0.9776
Validation Top-5 Accuracy: 0.9970

Epoch 15/100: 100%|██████████| 225/225 [00:29<00:00,  7.52it/s,
Loss=0.0957, Accuracy=99.97%, Elapsed=00:08:11, Remaining=00:46:27]

Epoch 15/100, Training Loss: 0.0957, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9999
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1528, Accuracy: 98.51%
Validation Precision: 0.9864, Recall: 0.9851, F1 Score: 0.9851
Validation ROC AUC: 0.9999, PR AUC: 0.9984, Balanced Accuracy: 0.9813
Validation Top-5 Accuracy: 0.9976
Average Loss Change: 991.644596, Average Accuracy Change: 0.238095

Epoch 16/100: 100%|██████████| 225/225 [00:29<00:00,  7.56it/s,
Loss=0.0940, Accuracy=100.00%, Elapsed=00:08:44, Remaining=00:45:52]

Epoch 16/100, Training Loss: 0.0940, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
```

```
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1535, Accuracy: 98.39%
Validation Precision: 0.9855, Recall: 0.9839, F1 Score: 0.9839
Validation ROC AUC: 1.0000, PR AUC: 0.9988, Balanced Accuracy: 0.9787
Validation Top-5 Accuracy: 0.9976

Epoch 17/100: 100%|██████████| 225/225 [00:30<00:00,  7.45it/s,
Loss=0.0929, Accuracy=100.00%, Elapsed=00:09:17, Remaining=00:45:21]

Epoch 17/100, Training Loss: 0.0930, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1503, Accuracy: 98.57%
Validation Precision: 0.9867, Recall: 0.9857, F1 Score: 0.9857
Validation ROC AUC: 0.9999, PR AUC: 0.9983, Balanced Accuracy: 0.9824
Validation Top-5 Accuracy: 0.9976

Epoch 18/100: 100%|██████████| 225/225 [00:29<00:00,  7.52it/s,
Loss=0.0925, Accuracy=100.00%, Elapsed=00:09:50, Remaining=00:44:47]

Epoch 18/100, Training Loss: 0.0926, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]
Validation Loss: 0.1574, Accuracy: 98.39%
Validation Precision: 0.9850, Recall: 0.9839, F1 Score: 0.9838
Validation ROC AUC: 0.9999, PR AUC: 0.9980, Balanced Accuracy: 0.9796
Validation Top-5 Accuracy: 0.9970

Epoch 19/100: 100%|██████████| 225/225 [00:30<00:00,  7.46it/s,
Loss=0.0941, Accuracy=100.00%, Elapsed=00:10:22, Remaining=00:44:15]

Epoch 19/100, Training Loss: 0.0941, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1902, Accuracy: 97.80%
Validation Precision: 0.9810, Recall: 0.9780, F1 Score: 0.9782
```

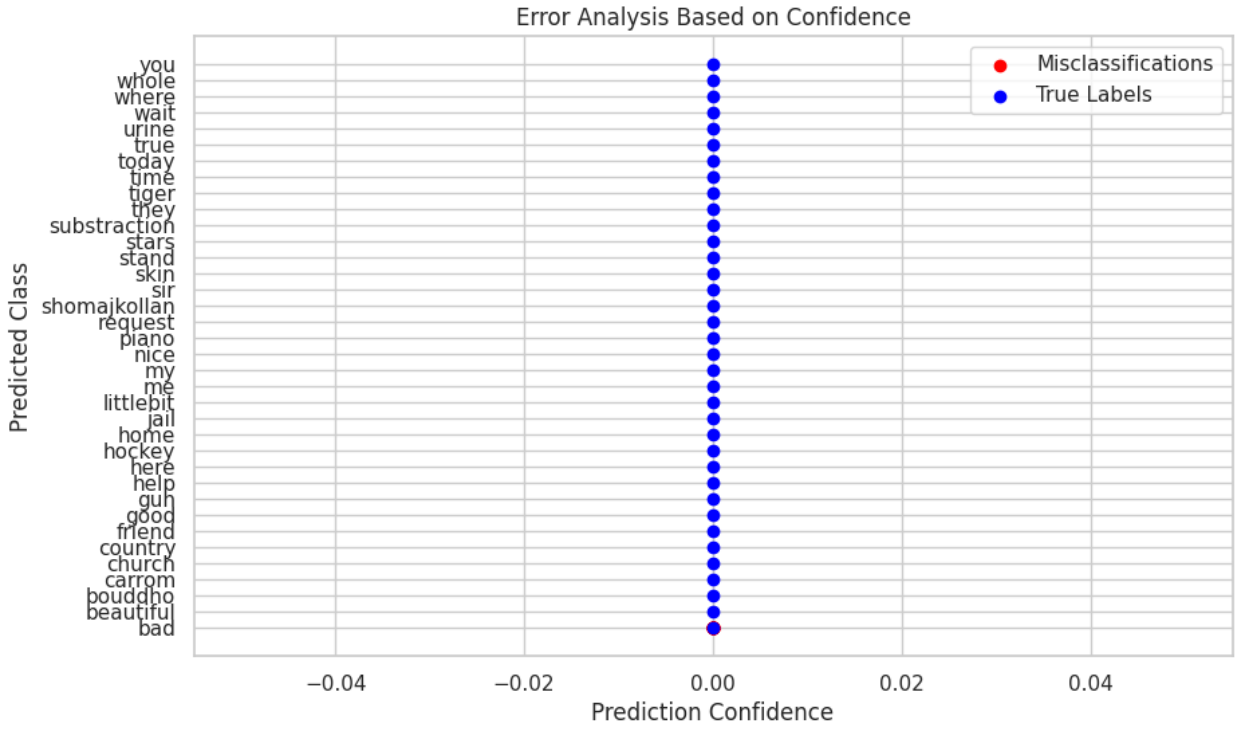
```
Validation ROC AUC: 0.9992, PR AUC: 0.9947, Balanced Accuracy: 0.9735
Validation Top-5 Accuracy: 0.9976

Epoch 20/100: 100%|██████████| 225/225 [00:29<00:00,  7.59it/s,
Loss=0.0937, Accuracy=99.97%, Elapsed=00:10:55, Remaining=00:43:41]

Epoch 20/100, Training Loss: 0.0937, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9996
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1581, Accuracy: 98.63%
Validation Precision: 0.9872, Recall: 0.9863, F1 Score: 0.9863
Validation ROC AUC: 0.9985, PR AUC: 0.9919, Balanced Accuracy: 0.9851
Validation Top-5 Accuracy: 0.9964
Average Loss Change: -0.002535, Average Accuracy Change: -0.029762
Early stopping at epoch 20

<ipython-input-4-ffb78043919f>:294: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  saved_state = torch.load(model_path, map_location=device)

Test Loss: 0.1507, Test Accuracy: 98.14%
Test Precision: 0.9835, Recall: 0.9814, F1 Score: 0.9807
Test ROC AUC: 0.9998, PR AUC:0.9962 Balanced Accuracy: 0.9797
Test Top-5 Accuracy: 1.0000
```



Confusion Matrix:




```
Error Analysis:
```




```
Training and testing RegNet_Y_32GF model...
Training and testing RegNet_Y_32GF model...

Epoch 1/100: 100%|██████████| 225/225 [00:38<00:00,  5.80it/s,
Loss=0.7939, Accuracy=80.35%, Elapsed=00:00:38, Remaining=01:02:58]

Epoch 1/100, Training Loss: 0.7836, Accuracy: 80.58%
Training Precision: 0.8087, Recall: 0.8058, F1 Score: 0.8054
Training ROC AUC: 0.9884, PR AUC: 0.8875, Balanced Accuracy: 0.7887
Training Top-5 Accuracy: 0.9259
Learning Rate: [0.0001]



Validation Loss: 0.3075, Accuracy: 95.00%
Validation Precision: 0.9550, Recall: 0.9500, F1 Score: 0.9485
Validation ROC AUC: 0.9990, PR AUC: 0.9900, Balanced Accuracy: 0.9394
Validation Top-5 Accuracy: 0.9952

Epoch 2/100: 100%|██████████| 225/225 [00:38<00:00,  5.78it/s,
Loss=0.1470, Accuracy=99.12%, Elapsed=00:01:23, Remaining=01:07:55]

Epoch 2/100, Training Loss: 0.1476, Accuracy: 99.14%
Training Precision: 0.9914, Recall: 0.9914, F1 Score: 0.9914
Training ROC AUC: 1.0000, PR AUC: 0.9991, Balanced Accuracy: 0.9907
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]
```

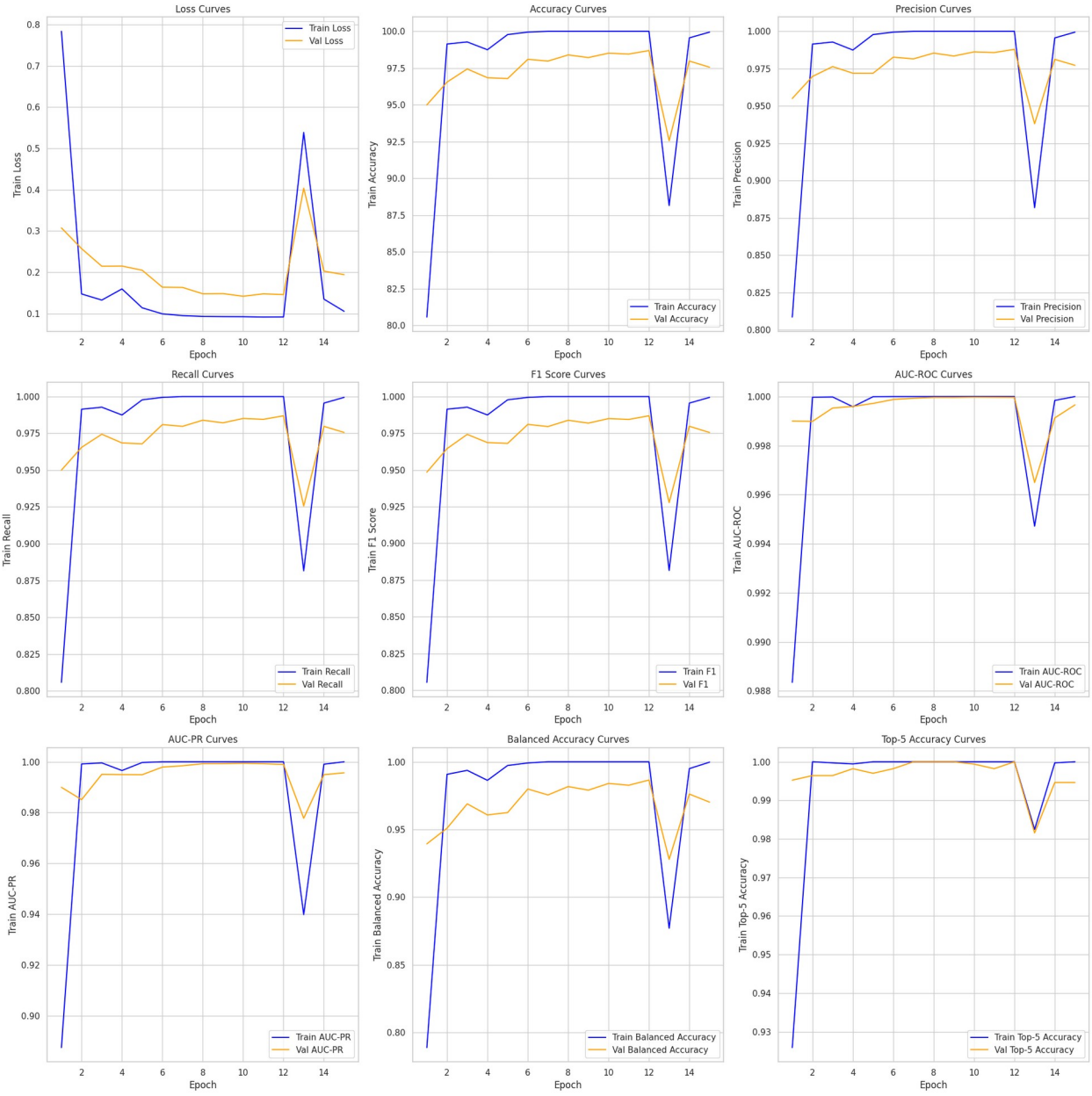
```
Validation Loss: 0.2573, Accuracy: 96.55%
Validation Precision: 0.9698, Recall: 0.9655, F1 Score: 0.9645
Validation ROC AUC: 0.9990, PR AUC: 0.9851, Balanced Accuracy: 0.9509
Validation Top-5 Accuracy: 0.9964

Epoch 3/100: 100%|██████████| 225/225 [00:38<00:00,  5.89it/s,
Loss=0.1324, Accuracy=99.26%, Elapsed=00:02:06, Remaining=01:08:13]

Epoch 3/100, Training Loss: 0.1328, Accuracy: 99.28%
Training Precision: 0.9928, Recall: 0.9928, F1 Score: 0.9928
Training ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9937
Training Top-5 Accuracy: 0.9997
Learning Rate: [0.0001]



Validation Loss: 0.2147, Accuracy: 97.44%
Validation Precision: 0.9764, Recall: 0.9744, F1 Score: 0.9742
Validation ROC AUC: 0.9995, PR AUC: 0.9951, Balanced Accuracy: 0.9689
Validation Top-5 Accuracy: 0.9964

Epoch 4/100: 100%|██████████| 225/225 [00:38<00:00,  5.86it/s,
Loss=0.1596, Accuracy=98.73%, Elapsed=00:02:49, Remaining=01:07:58]

Epoch 4/100, Training Loss: 0.1598, Accuracy: 98.75%
Training Precision: 0.9875, Recall: 0.9875, F1 Score: 0.9874
Training ROC AUC: 0.9996, PR AUC: 0.9966, Balanced Accuracy: 0.9863
Training Top-5 Accuracy: 0.9994
Learning Rate: [0.0001]



Validation Loss: 0.2151, Accuracy: 96.85%
Validation Precision: 0.9719, Recall: 0.9685, F1 Score: 0.9686
Validation ROC AUC: 0.9996, PR AUC: 0.9950, Balanced Accuracy: 0.9607
Validation Top-5 Accuracy: 0.9982

Epoch 5/100: 100%|██████████| 225/225 [00:38<00:00,  5.89it/s,
Loss=0.1140, Accuracy=99.77%, Elapsed=00:03:31, Remaining=01:07:07]

Epoch 5/100, Training Loss: 0.1140, Accuracy: 99.78%
Training Precision: 0.9978, Recall: 0.9978, F1 Score: 0.9978
Training ROC AUC: 1.0000, PR AUC: 0.9997, Balanced Accuracy: 0.9973
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.2051, Accuracy: 96.79%
Validation Precision: 0.9719, Recall: 0.9679, F1 Score: 0.9681
Validation ROC AUC: 0.9997, PR AUC: 0.9949, Balanced Accuracy: 0.9625
Validation Top-5 Accuracy: 0.9970
```

```
Epoch 6/100: 100%|██████████| 225/225 [00:38<00:00,  5.86it/s,
Loss=0.0993, Accuracy=99.94%, Elapsed=00:04:14, Remaining=01:06:22]

Epoch 6/100, Training Loss: 0.0994, Accuracy: 99.94%
Training Precision: 0.9994, Recall: 0.9994, F1 Score: 0.9994
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9991
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1640, Accuracy: 98.10%
Validation Precision: 0.9827, Recall: 0.9810, F1 Score: 0.9810
Validation ROC AUC: 0.9999, PR AUC: 0.9979, Balanced Accuracy: 0.9799
Validation Top-5 Accuracy: 0.9982

Epoch 7/100: 100%|██████████| 225/225 [00:38<00:00,  5.87it/s,
Loss=0.0951, Accuracy=100.00%, Elapsed=00:04:57, Remaining=01:05:51]

Epoch 7/100, Training Loss: 0.0951, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1634, Accuracy: 97.98%
Validation Precision: 0.9815, Recall: 0.9798, F1 Score: 0.9796
Validation ROC AUC: 0.9999, PR AUC: 0.9984, Balanced Accuracy: 0.9755
Validation Top-5 Accuracy: 1.0000

Epoch 8/100: 100%|██████████| 225/225 [00:38<00:00,  5.85it/s,
Loss=0.0931, Accuracy=100.00%, Elapsed=00:05:39, Remaining=01:05:07]

Epoch 8/100, Training Loss: 0.0932, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1481, Accuracy: 98.39%
Validation Precision: 0.9854, Recall: 0.9839, F1 Score: 0.9838
Validation ROC AUC: 1.0000, PR AUC: 0.9993, Balanced Accuracy: 0.9817
Validation Top-5 Accuracy: 1.0000

Epoch 9/100: 100%|██████████| 225/225 [00:37<00:00,  5.95it/s,
Loss=0.0924, Accuracy=100.00%, Elapsed=00:06:22, Remaining=01:04:28]
```

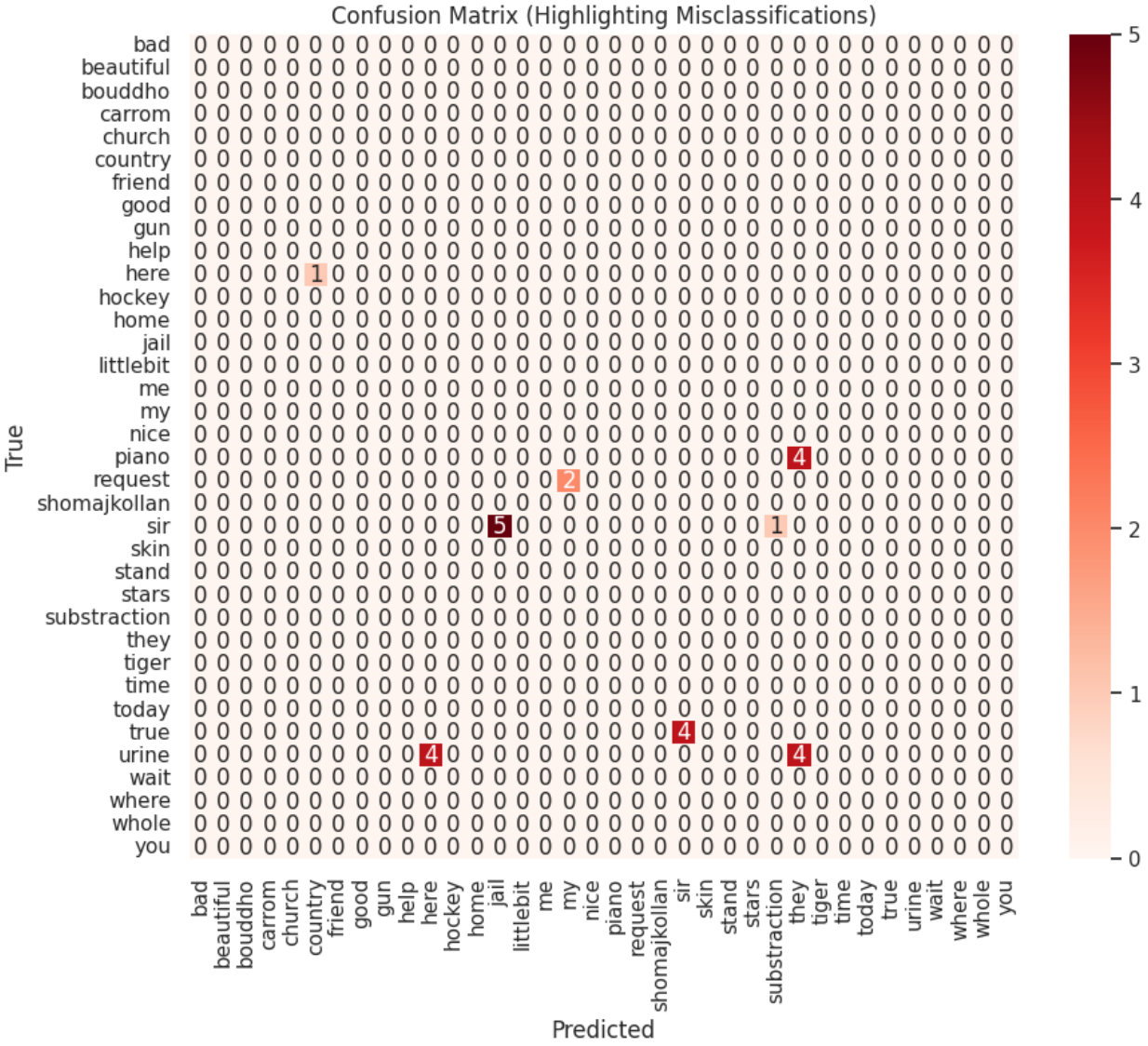
```
Epoch 9/100, Training Loss: 0.0927, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1485, Accuracy: 98.21%
Validation Precision: 0.9834, Recall: 0.9821, F1 Score: 0.9819
Validation ROC AUC: 1.0000, PR AUC: 0.9993, Balanced Accuracy: 0.9790
Validation Top-5 Accuracy: 1.0000

Epoch 10/100: 100%|██████████| 225/225 [00:37<00:00,  5.93it/s,
Loss=0.0925, Accuracy=100.00%, Elapsed=00:07:04, Remaining=01:03:39]

Epoch 10/100, Training Loss: 0.0925, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1420, Accuracy: 98.51%
Validation Precision: 0.9862, Recall: 0.9851, F1 Score: 0.9850
Validation ROC AUC: 1.0000, PR AUC: 0.9994, Balanced Accuracy: 0.9840
Validation Top-5 Accuracy: 0.9994
Average Loss Change: 0.038329, Average Accuracy Change: 0.615079

Epoch 11/100: 100%|██████████| 225/225 [00:38<00:00,  5.88it/s,
Loss=0.0916, Accuracy=100.00%, Elapsed=00:07:47, Remaining=01:03:03]

Epoch 11/100, Training Loss: 0.0917, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1479, Accuracy: 98.45%
Validation Precision: 0.9858, Recall: 0.9845, F1 Score: 0.9844
Validation ROC AUC: 1.0000, PR AUC: 0.9992, Balanced Accuracy: 0.9827
Validation Top-5 Accuracy: 0.9982

Epoch 12/100: 100%|██████████| 225/225 [00:37<00:00,  5.94it/s,
Loss=0.0912, Accuracy=100.00%, Elapsed=00:08:29, Remaining=01:02:15]

Epoch 12/100, Training Loss: 0.0919, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
```

```
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1461, Accuracy: 98.69%
Validation Precision: 0.9879, Recall: 0.9869, F1 Score: 0.9869
Validation ROC AUC: 1.0000, PR AUC: 0.9990, Balanced Accuracy: 0.9865
Validation Top-5 Accuracy: 1.0000

Epoch 13/100: 100%|██████████| 225/225 [00:38<00:00,  5.86it/s,
Loss=0.5416, Accuracy=88.15%, Elapsed=00:09:12, Remaining=01:01:38]

Epoch 13/100, Training Loss: 0.5387, Accuracy: 88.16%
Training Precision: 0.8819, Recall: 0.8816, F1 Score: 0.8816
Training ROC AUC: 0.9947, PR AUC: 0.9398, Balanced Accuracy: 0.8770
Training Top-5 Accuracy: 0.9824
Learning Rate: [7e-05]



Validation Loss: 0.4037, Accuracy: 92.56%
Validation Precision: 0.9381, Recall: 0.9256, F1 Score: 0.9278
Validation ROC AUC: 0.9965, PR AUC: 0.9778, Balanced Accuracy: 0.9279
Validation Top-5 Accuracy: 0.9815

Epoch 14/100: 100%|██████████| 225/225 [00:38<00:00,  5.88it/s,
Loss=0.1351, Accuracy=99.55%, Elapsed=00:09:54, Remaining=01:00:53]

Epoch 14/100, Training Loss: 0.1351, Accuracy: 99.55%
Training Precision: 0.9956, Recall: 0.9955, F1 Score: 0.9955
Training ROC AUC: 0.9998, PR AUC: 0.9991, Balanced Accuracy: 0.9950
Training Top-5 Accuracy: 0.9997
Learning Rate: [7e-05]



Validation Loss: 0.2027, Accuracy: 97.98%
Validation Precision: 0.9812, Recall: 0.9798, F1 Score: 0.9798
Validation ROC AUC: 0.9991, PR AUC: 0.9949, Balanced Accuracy: 0.9761
Validation Top-5 Accuracy: 0.9946

Epoch 15/100: 100%|██████████| 225/225 [00:38<00:00,  5.87it/s,
Loss=0.1057, Accuracy=99.94%, Elapsed=00:10:37, Remaining=01:00:09]

Epoch 15/100, Training Loss: 0.1056, Accuracy: 99.94%
Training Precision: 0.9994, Recall: 0.9994, F1 Score: 0.9994
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9997
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]
```

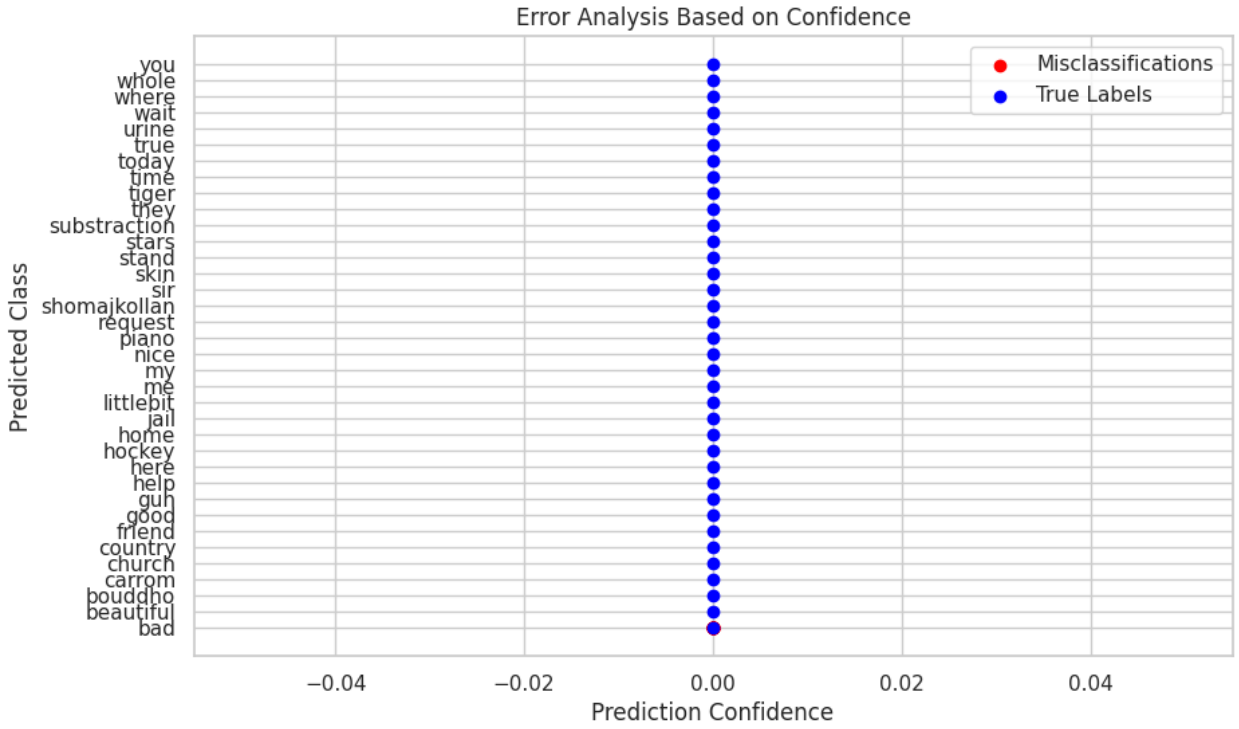
```
Validation Loss: 0.1943, Accuracy: 97.56%
Validation Precision: 0.9772, Recall: 0.9756, F1 Score: 0.9755
Validation ROC AUC: 0.9997, PR AUC: 0.9957, Balanced Accuracy: 0.9702
Validation Top-5 Accuracy: 0.9946
Average Loss Change: -0.010778, Average Accuracy Change: -0.089286
Early stopping at epoch 15

<ipython-input-4-ffb78043919f>:294: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  saved_state = torch.load(model_path, map_location=device)

Test Loss: 0.1672, Test Accuracy: 96.91%
Test Precision: 0.9727, Recall: 0.9691, F1 Score: 0.9687
Test ROC AUC: 0.9999, PR AUC:0.9955 Balanced Accuracy: 0.9637
Test Top-5 Accuracy: 1.0000
```



Confusion Matrix:




```
Error Analysis:
```




```
Training and testing ResNeXt101_64X4D model...
Training and testing ResNeXt101_64X4D model...

Epoch 1/100: 100%|██████████| 113/113 [00:31<00:00,  3.54it/s,
Loss=1.8815, Accuracy=60.53%, Elapsed=00:00:31, Remaining=00:51:59]

Epoch 1/100, Training Loss: 1.8667, Accuracy: 60.80%
Training Precision: 0.7055, Recall: 0.6080, F1 Score: 0.6051
Training ROC AUC: 0.9374, PR AUC: 0.6722, Balanced Accuracy: 0.5159
Training Top-5 Accuracy: 0.7816
Learning Rate: [0.0001]



Validation Loss: 0.2896, Accuracy: 95.48%
Validation Precision: 0.9599, Recall: 0.9548, F1 Score: 0.9541
Validation ROC AUC: 0.9991, PR AUC: 0.9900, Balanced Accuracy: 0.9370
Validation Top-5 Accuracy: 0.9946

Epoch 2/100: 100%|██████████| 113/113 [00:32<00:00,  3.46it/s,
Loss=0.1427, Accuracy=99.18%, Elapsed=00:01:07, Remaining=00:55:19]

Epoch 2/100, Training Loss: 0.1433, Accuracy: 99.19%
Training Precision: 0.9921, Recall: 0.9919, F1 Score: 0.9919
Training ROC AUC: 1.0000, PR AUC: 0.9992, Balanced Accuracy: 0.9896
Training Top-5 Accuracy: 0.9994
Learning Rate: [0.0001]
```

```
Validation Loss: 0.1873, Accuracy: 97.50%
Validation Precision: 0.9762, Recall: 0.9750, F1 Score: 0.9748
Validation ROC AUC: 0.9996, PR AUC: 0.9935, Balanced Accuracy: 0.9689
Validation Top-5 Accuracy: 0.9976

Epoch 3/100: 100%|██████████| 113/113 [00:31<00:00,  3.58it/s,
Loss=0.1005, Accuracy=99.94%, Elapsed=00:01:42, Remaining=00:55:27]

Epoch 3/100, Training Loss: 0.1026, Accuracy: 99.94%
Training Precision: 0.9994, Recall: 0.9994, F1 Score: 0.9994
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9992
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1598, Accuracy: 97.92%
Validation Precision: 0.9802, Recall: 0.9792, F1 Score: 0.9791
Validation ROC AUC: 0.9999, PR AUC: 0.9986, Balanced Accuracy: 0.9776
Validation Top-5 Accuracy: 0.9970

Epoch 4/100: 100%|██████████| 113/113 [00:31<00:00,  3.55it/s,
Loss=0.0984, Accuracy=99.97%, Elapsed=00:02:18, Remaining=00:55:18]

Epoch 4/100, Training Loss: 0.1062, Accuracy: 99.94%
Training Precision: 0.9994, Recall: 0.9994, F1 Score: 0.9994
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9997
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1571, Accuracy: 97.80%
Validation Precision: 0.9791, Recall: 0.9780, F1 Score: 0.9777
Validation ROC AUC: 0.9999, PR AUC: 0.9986, Balanced Accuracy: 0.9754
Validation Top-5 Accuracy: 0.9976

Epoch 5/100: 100%|██████████| 113/113 [00:31<00:00,  3.56it/s,
Loss=0.1037, Accuracy=99.72%, Elapsed=00:02:52, Remaining=00:54:44]

Epoch 5/100, Training Loss: 0.1072, Accuracy: 99.72%
Training Precision: 0.9972, Recall: 0.9972, F1 Score: 0.9972
Training ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9969
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1891, Accuracy: 97.26%
Validation Precision: 0.9742, Recall: 0.9726, F1 Score: 0.9723
Validation ROC AUC: 0.9998, PR AUC: 0.9956, Balanced Accuracy: 0.9708
Validation Top-5 Accuracy: 0.9935
```

```
Epoch 6/100: 100%|██████████| 113/113 [00:31<00:00,  3.60it/s,
Loss=0.0969, Accuracy=99.94%, Elapsed=00:03:27, Remaining=00:54:04]

Epoch 6/100, Training Loss: 0.1042, Accuracy: 99.92%
Training Precision: 0.9992, Recall: 0.9992, F1 Score: 0.9992
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9991
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1653, Accuracy: 98.10%
Validation Precision: 0.9819, Recall: 0.9810, F1 Score: 0.9809
Validation ROC AUC: 0.9999, PR AUC: 0.9962, Balanced Accuracy: 0.9778
Validation Top-5 Accuracy: 0.9964

Epoch 7/100: 100%|██████████| 113/113 [00:31<00:00,  3.56it/s,
Loss=0.1014, Accuracy=99.83%, Elapsed=00:04:02, Remaining=00:53:39]

Epoch 7/100, Training Loss: 0.1032, Accuracy: 99.80%
Training Precision: 0.9981, Recall: 0.9980, F1 Score: 0.9981
Training ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9979
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1837, Accuracy: 97.86%
Validation Precision: 0.9799, Recall: 0.9786, F1 Score: 0.9785
Validation ROC AUC: 0.9996, PR AUC: 0.9909, Balanced Accuracy: 0.9784
Validation Top-5 Accuracy: 0.9946

Epoch 8/100: 100%|██████████| 113/113 [00:31<00:00,  3.59it/s,
Loss=0.0951, Accuracy=99.94%, Elapsed=00:04:36, Remaining=00:53:01]

Epoch 8/100, Training Loss: 0.1012, Accuracy: 99.92%
Training Precision: 0.9992, Recall: 0.9992, F1 Score: 0.9992
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9993
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1734, Accuracy: 97.98%
Validation Precision: 0.9806, Recall: 0.9798, F1 Score: 0.9796
Validation ROC AUC: 0.9997, PR AUC: 0.9927, Balanced Accuracy: 0.9785
Validation Top-5 Accuracy: 0.9958

Epoch 9/100: 100%|██████████| 113/113 [00:31<00:00,  3.56it/s,
Loss=0.0975, Accuracy=99.92%, Elapsed=00:05:11, Remaining=00:52:26]
```

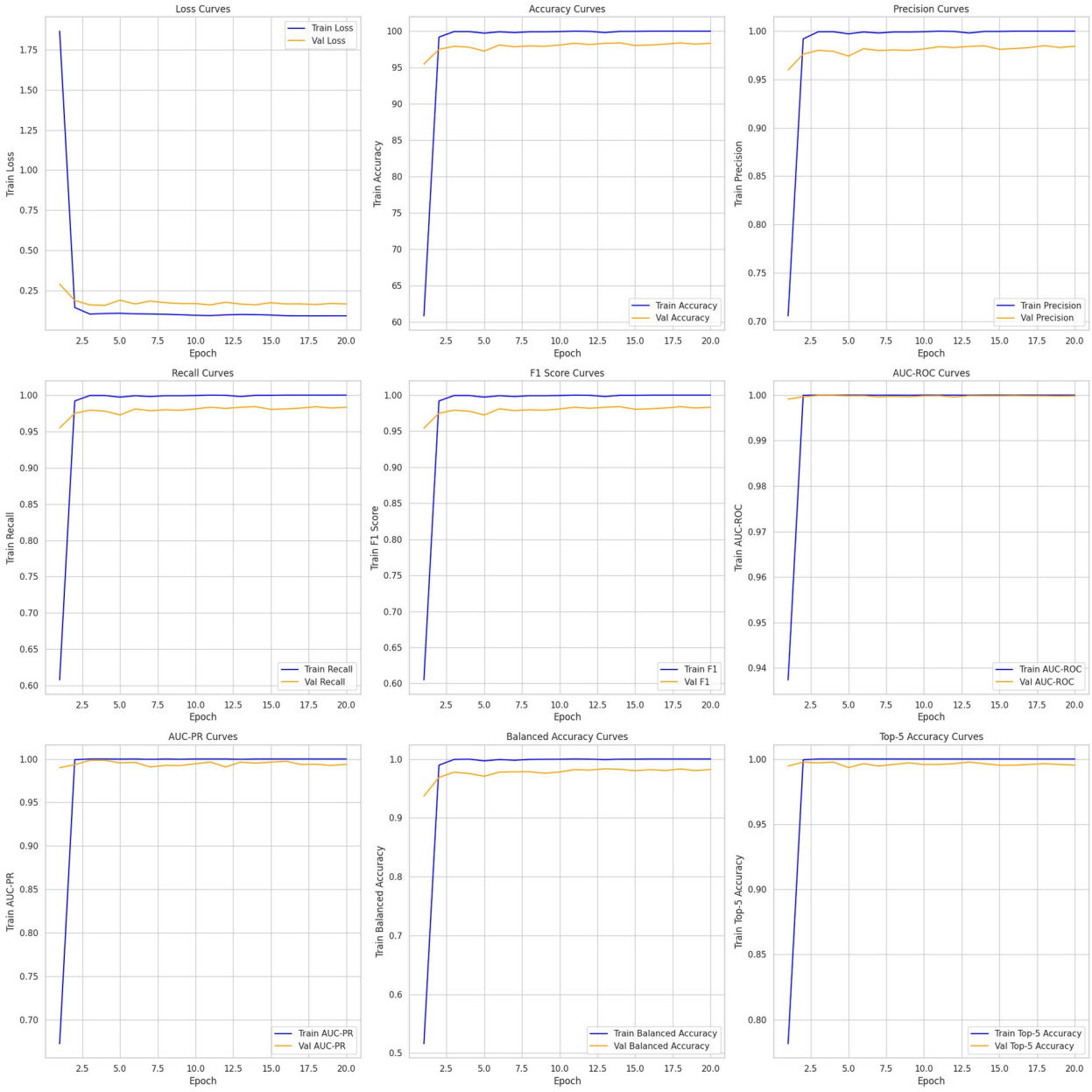
```
Epoch 9/100, Training Loss: 0.0983, Accuracy: 99.92%
Training Precision: 0.9992, Recall: 0.9992, F1 Score: 0.9992
Training ROC AUC: 1.0000, PR AUC: 0.9998, Balanced Accuracy: 0.9995
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1682, Accuracy: 97.92%
Validation Precision: 0.9801, Recall: 0.9792, F1 Score: 0.9790
Validation ROC AUC: 0.9997, PR AUC: 0.9925, Balanced Accuracy: 0.9758
Validation Top-5 Accuracy: 0.9970

Epoch 10/100: 100%|██████████| 113/113 [00:31<00:00,  3.60it/s,
Loss=0.0945, Accuracy=99.94%, Elapsed=00:05:45, Remaining=00:51:49]

Epoch 10/100, Training Loss: 0.0948, Accuracy: 99.94%
Training Precision: 0.9994, Recall: 0.9994, F1 Score: 0.9994
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9997
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1677, Accuracy: 98.10%
Validation Precision: 0.9816, Recall: 0.9810, F1 Score: 0.9808
Validation ROC AUC: 0.9998, PR AUC: 0.9945, Balanced Accuracy: 0.9780
Validation Top-5 Accuracy: 0.9958
Average Loss Change: 0.022901, Average Accuracy Change: 0.476190

Epoch 11/100: 100%|██████████| 113/113 [00:31<00:00,  3.60it/s,
Loss=0.0928, Accuracy=100.00%, Elapsed=00:06:19, Remaining=00:51:12]

Epoch 11/100, Training Loss: 0.0931, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1600, Accuracy: 98.33%
Validation Precision: 0.9839, Recall: 0.9833, F1 Score: 0.9832
Validation ROC AUC: 0.9999, PR AUC: 0.9965, Balanced Accuracy: 0.9820
Validation Top-5 Accuracy: 0.9958

Epoch 12/100: 100%|██████████| 113/113 [00:31<00:00,  3.56it/s,
Loss=0.0918, Accuracy=100.00%, Elapsed=00:06:54, Remaining=00:50:42]

Epoch 12/100, Training Loss: 0.0972, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9998
```

```
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1762, Accuracy: 98.15%
Validation Precision: 0.9831, Recall: 0.9815, F1 Score: 0.9816
Validation ROC AUC: 0.9996, PR AUC: 0.9908, Balanced Accuracy: 0.9812
Validation Top-5 Accuracy: 0.9964

Epoch 13/100: 100%|██████████| 113/113 [00:31<00:00,  3.58it/s,
Loss=0.0996, Accuracy=99.80%, Elapsed=00:07:29, Remaining=00:50:06]

Epoch 13/100, Training Loss: 0.0998, Accuracy: 99.80%
Training Precision: 0.9981, Recall: 0.9980, F1 Score: 0.9981
Training ROC AUC: 1.0000, PR AUC: 0.9997, Balanced Accuracy: 0.9990
Training Top-5 Accuracy: 1.0000
Learning Rate: [3.4299999999999993e-05]



Validation Loss: 0.1647, Accuracy: 98.33%
Validation Precision: 0.9842, Recall: 0.9833, F1 Score: 0.9832
Validation ROC AUC: 0.9999, PR AUC: 0.9964, Balanced Accuracy: 0.9834
Validation Top-5 Accuracy: 0.9976

Epoch 14/100: 100%|██████████| 113/113 [00:31<00:00,  3.58it/s,
Loss=0.0923, Accuracy=100.00%, Elapsed=00:08:03, Remaining=00:49:31]

Epoch 14/100, Training Loss: 0.0988, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9998
Training Top-5 Accuracy: 1.0000
Learning Rate: [3.4299999999999993e-05]



Validation Loss: 0.1603, Accuracy: 98.39%
Validation Precision: 0.9849, Recall: 0.9839, F1 Score: 0.9838
Validation ROC AUC: 0.9999, PR AUC: 0.9952, Balanced Accuracy: 0.9825
Validation Top-5 Accuracy: 0.9964

Epoch 15/100: 100%|██████████| 113/113 [00:31<00:00,  3.56it/s,
Loss=0.0928, Accuracy=99.97%, Elapsed=00:08:38, Remaining=00:49:00]

Epoch 15/100, Training Loss: 0.0963, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9998
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]
```

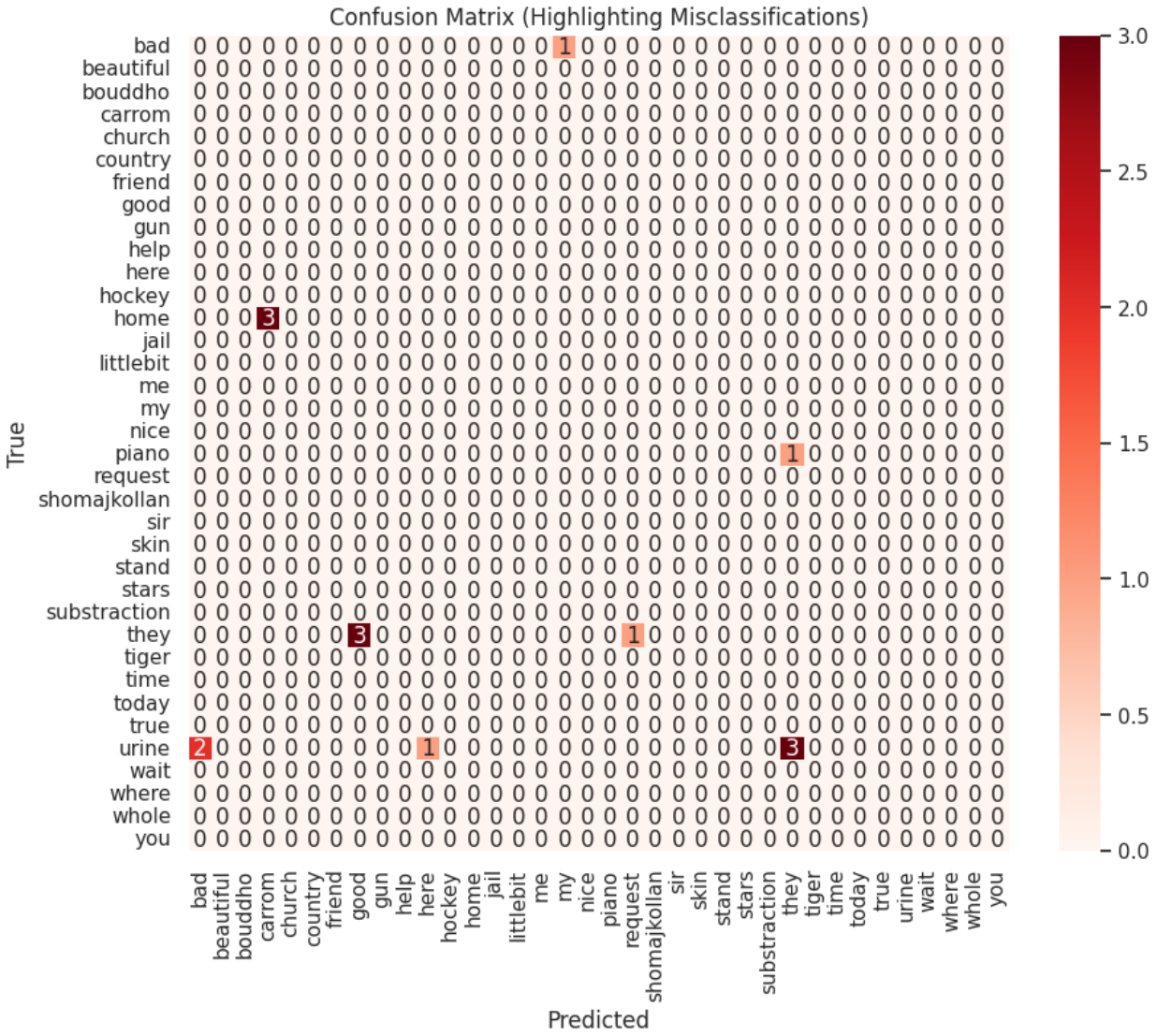
```
Validation Loss: 0.1735, Accuracy: 98.04%
Validation Precision: 0.9813, Recall: 0.9804, F1 Score: 0.9803
Validation ROC AUC: 0.9998, PR AUC: 0.9964, Balanced Accuracy: 0.9802
Validation Top-5 Accuracy: 0.9952
Average Loss Change: 0.004127, Average Accuracy Change: 0.165816

Epoch 16/100: 100%|██████████| 113/113 [00:31<00:00,  3.59it/s,
Loss=0.0915, Accuracy=100.00%, Elapsed=00:09:13, Remaining=00:48:24]

Epoch 16/100, Training Loss: 0.0923, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1656, Accuracy: 98.10%
Validation Precision: 0.9820, Recall: 0.9810, F1 Score: 0.9810
Validation ROC AUC: 0.9999, PR AUC: 0.9975, Balanced Accuracy: 0.9819
Validation Top-5 Accuracy: 0.9952

Epoch 17/100: 100%|██████████| 113/113 [00:31<00:00,  3.58it/s,
Loss=0.0914, Accuracy=100.00%, Elapsed=00:09:47, Remaining=00:47:49]

Epoch 17/100, Training Loss: 0.0915, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1658, Accuracy: 98.21%
Validation Precision: 0.9831, Recall: 0.9821, F1 Score: 0.9821
Validation ROC AUC: 0.9998, PR AUC: 0.9937, Balanced Accuracy: 0.9805
Validation Top-5 Accuracy: 0.9958

Epoch 18/100: 100%|██████████| 113/113 [00:31<00:00,  3.56it/s,
Loss=0.0910, Accuracy=100.00%, Elapsed=00:10:22, Remaining=00:47:15]

Epoch 18/100, Training Loss: 0.0911, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1622, Accuracy: 98.39%
Validation Precision: 0.9850, Recall: 0.9839, F1 Score: 0.9839
```

```
Validation ROC AUC: 0.9998, PR AUC: 0.9939, Balanced Accuracy: 0.9830
Validation Top-5 Accuracy: 0.9964

Epoch 19/100: 100%|██████████| 113/113 [00:31<00:00,  3.59it/s,
Loss=0.0911, Accuracy=100.00%, Elapsed=00:10:56, Remaining=00:46:39]

Epoch 19/100, Training Loss: 0.0914, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [2.4009999999999995e-05]



Validation Loss: 0.1688, Accuracy: 98.21%
Validation Precision: 0.9831, Recall: 0.9821, F1 Score: 0.9820
Validation ROC AUC: 0.9998, PR AUC: 0.9926, Balanced Accuracy: 0.9804
Validation Top-5 Accuracy: 0.9958

Epoch 20/100: 100%|██████████| 113/113 [00:31<00:00,  3.58it/s,
Loss=0.0912, Accuracy=100.00%, Elapsed=00:11:31, Remaining=00:46:04]

Epoch 20/100, Training Loss: 0.0914, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 1.0000
Training Top-5 Accuracy: 1.0000
Learning Rate: [1.6806999999999997e-05]



Validation Loss: 0.1655, Accuracy: 98.33%
Validation Precision: 0.9844, Recall: 0.9833, F1 Score: 0.9833
Validation ROC AUC: 0.9998, PR AUC: 0.9939, Balanced Accuracy: 0.9822
Validation Top-5 Accuracy: 0.9952
Average Loss Change: -0.001892, Average Accuracy Change: -0.059524
Early stopping at epoch 20

<ipython-input-4-ffb78043919f>:294: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
```

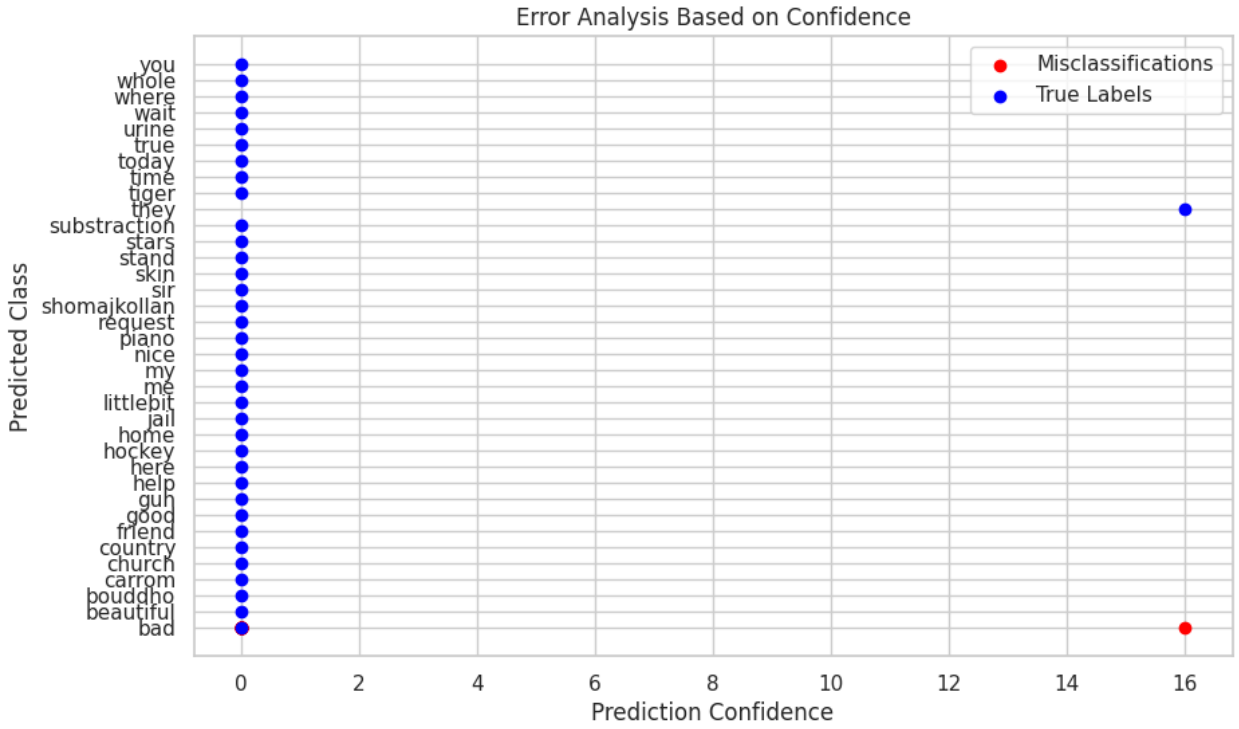
```
related to this experimental feature.
  saved_state = torch.load(model_path, map_location=device)

Test Loss: 0.1485, Test Accuracy: 98.14%
Test Precision: 0.9825, Recall: 0.9814, F1 Score: 0.9814
Test ROC AUC: 0.9999, PR AUC:0.9988 Balanced Accuracy: 0.9898
Test Top-5 Accuracy: 0.9988
```




```
Confusion Matrix:
```




```
Error Analysis:
```




```
Training and testing Swin_V2_B model...
Training and testing Swin_V2_B model...

Epoch 1/100: 100%|██████████| 225/225 [00:43<00:00,  5.21it/s,
Loss=1.3049, Accuracy=66.63%, Elapsed=00:00:42, Remaining=01:10:07]

Epoch 1/100, Training Loss: 1.2870, Accuracy: 67.01%
Training Precision: 0.6974, Recall: 0.6701, F1 Score: 0.6698
Training ROC AUC: 0.9669, PR AUC: 0.7551, Balanced Accuracy: 0.6197
Training Top-5 Accuracy: 0.8520
Learning Rate: [0.0001]



Validation Loss: 0.2639, Accuracy: 95.00%
Validation Precision: 0.9544, Recall: 0.9500, F1 Score: 0.9491
Validation ROC AUC: 0.9996, PR AUC: 0.9912, Balanced Accuracy: 0.9376
Validation Top-5 Accuracy: 0.9964

Epoch 2/100: 100%|██████████| 225/225 [00:42<00:00,  5.30it/s,
Loss=0.1915, Accuracy=98.08%, Elapsed=00:01:30, Remaining=01:13:52]

Epoch 2/100, Training Loss: 0.1913, Accuracy: 98.08%
Training Precision: 0.9809, Recall: 0.9808, F1 Score: 0.9808
Training ROC AUC: 0.9999, PR AUC: 0.9962, Balanced Accuracy: 0.9759
Training Top-5 Accuracy: 0.9997
Learning Rate: [0.0001]
```

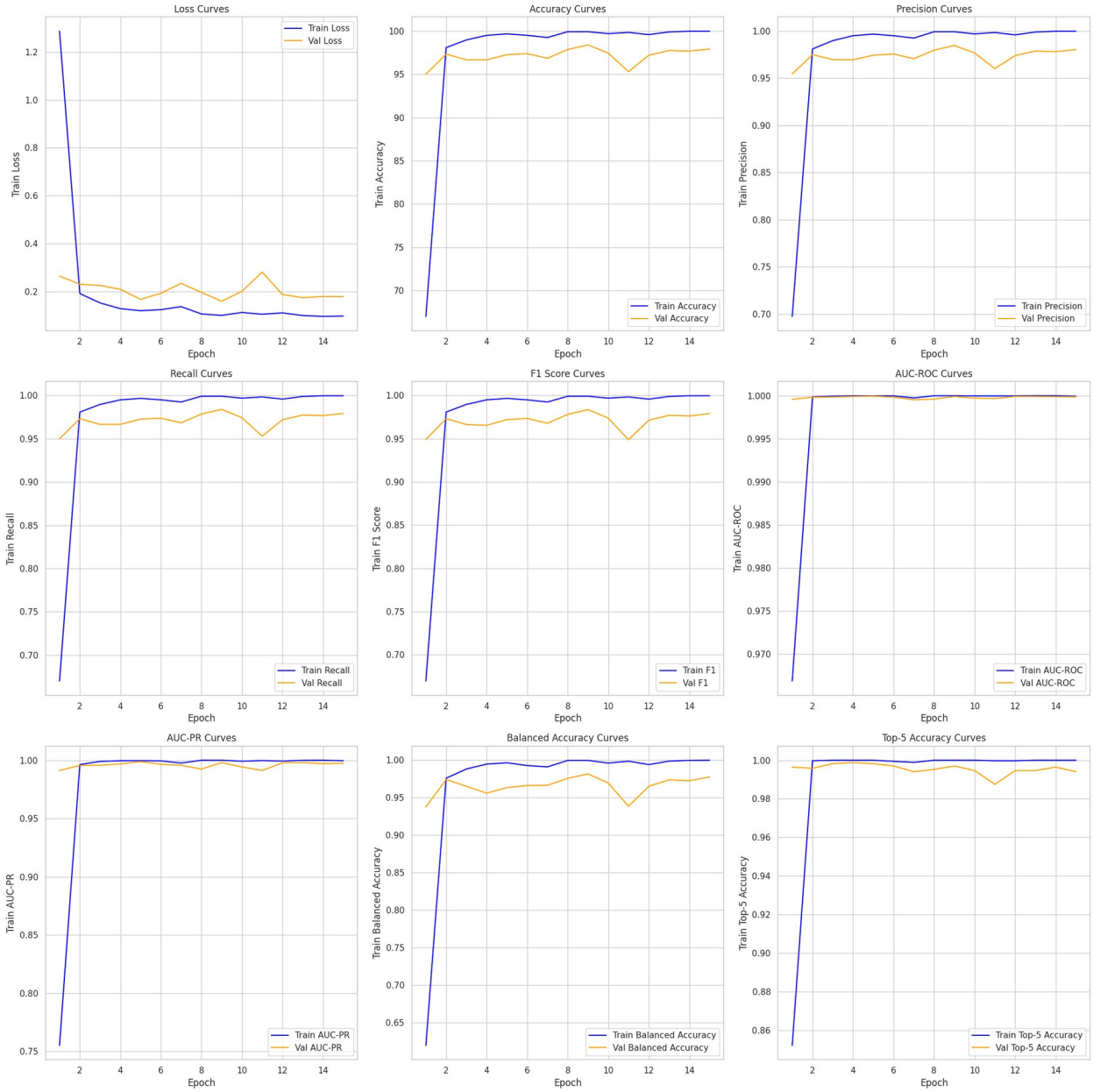
```
Validation Loss: 0.2299, Accuracy: 97.32%
Validation Precision: 0.9749, Recall: 0.9732, F1 Score: 0.9733
Validation ROC AUC: 0.9998, PR AUC: 0.9955, Balanced Accuracy: 0.9742
Validation Top-5 Accuracy: 0.9958

Epoch 3/100: 100%|██████████| 225/225 [00:42<00:00,  5.24it/s,
Loss=0.1523, Accuracy=98.95%, Elapsed=00:02:19, Remaining=01:14:56]

Epoch 3/100, Training Loss: 0.1520, Accuracy: 98.97%
Training Precision: 0.9898, Recall: 0.9897, F1 Score: 0.9897
Training ROC AUC: 1.0000, PR AUC: 0.9990, Balanced Accuracy: 0.9881
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.2248, Accuracy: 96.67%
Validation Precision: 0.9696, Recall: 0.9667, F1 Score: 0.9664
Validation ROC AUC: 0.9999, PR AUC: 0.9957, Balanced Accuracy: 0.9651
Validation Top-5 Accuracy: 0.9982

Epoch 4/100: 100%|██████████| 225/225 [00:42<00:00,  5.31it/s,
Loss=0.1279, Accuracy=99.52%, Elapsed=00:03:06, Remaining=01:14:30]

Epoch 4/100, Training Loss: 0.1280, Accuracy: 99.50%
Training Precision: 0.9950, Recall: 0.9950, F1 Score: 0.9950
Training ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9947
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.2088, Accuracy: 96.67%
Validation Precision: 0.9694, Recall: 0.9667, F1 Score: 0.9655
Validation ROC AUC: 0.9999, PR AUC: 0.9968, Balanced Accuracy: 0.9559
Validation Top-5 Accuracy: 0.9988

Epoch 5/100: 100%|██████████| 225/225 [00:42<00:00,  5.28it/s,
Loss=0.1197, Accuracy=99.66%, Elapsed=00:03:53, Remaining=01:13:59]

Epoch 5/100, Training Loss: 0.1194, Accuracy: 99.67%
Training Precision: 0.9967, Recall: 0.9967, F1 Score: 0.9967
Training ROC AUC: 1.0000, PR AUC: 0.9996, Balanced Accuracy: 0.9964
Training Top-5 Accuracy: 1.0000
Learning Rate: [0.0001]



Validation Loss: 0.1668, Accuracy: 97.26%
Validation Precision: 0.9743, Recall: 0.9726, F1 Score: 0.9721
Validation ROC AUC: 1.0000, PR AUC: 0.9988, Balanced Accuracy: 0.9634
Validation Top-5 Accuracy: 0.9982
```

```
Epoch 6/100: 100%|██████████| 225/225 [00:42<00:00,  5.33it/s,
Loss=0.1236, Accuracy=99.52%, Elapsed=00:04:40, Remaining=01:13:18]

Epoch 6/100, Training Loss: 0.1237, Accuracy: 99.50%
Training Precision: 0.9950, Recall: 0.9950, F1 Score: 0.9950
Training ROC AUC: 1.0000, PR AUC: 0.9994, Balanced Accuracy: 0.9928
Training Top-5 Accuracy: 0.9994
Learning Rate: [0.0001]



Validation Loss: 0.1925, Accuracy: 97.38%
Validation Precision: 0.9756, Recall: 0.9738, F1 Score: 0.9734
Validation ROC AUC: 0.9998, PR AUC: 0.9965, Balanced Accuracy: 0.9661
Validation Top-5 Accuracy: 0.9970

Epoch 7/100: 100%|██████████| 225/225 [00:42<00:00,  5.31it/s,
Loss=0.1370, Accuracy=99.24%, Elapsed=00:05:28, Remaining=01:12:47]

Epoch 7/100, Training Loss: 0.1365, Accuracy: 99.25%
Training Precision: 0.9925, Recall: 0.9925, F1 Score: 0.9925
Training ROC AUC: 0.9997, PR AUC: 0.9976, Balanced Accuracy: 0.9910
Training Top-5 Accuracy: 0.9989
Learning Rate: [7e-05]



Validation Loss: 0.2336, Accuracy: 96.85%
Validation Precision: 0.9706, Recall: 0.9685, F1 Score: 0.9680
Validation ROC AUC: 0.9995, PR AUC: 0.9957, Balanced Accuracy: 0.9665
Validation Top-5 Accuracy: 0.9940

Epoch 8/100: 100%|██████████| 225/225 [00:42<00:00,  5.31it/s,
Loss=0.1059, Accuracy=99.92%, Elapsed=00:06:15, Remaining=01:12:03]

Epoch 8/100, Training Loss: 0.1058, Accuracy: 99.92%
Training Precision: 0.9992, Recall: 0.9992, F1 Score: 0.9992
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9996
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1960, Accuracy: 97.86%
Validation Precision: 0.9797, Recall: 0.9786, F1 Score: 0.9782
Validation ROC AUC: 0.9996, PR AUC: 0.9924, Balanced Accuracy: 0.9758
Validation Top-5 Accuracy: 0.9952

Epoch 9/100: 100%|██████████| 225/225 [00:42<00:00,  5.30it/s,
Loss=0.1000, Accuracy=99.92%, Elapsed=00:07:04, Remaining=01:11:27]
```

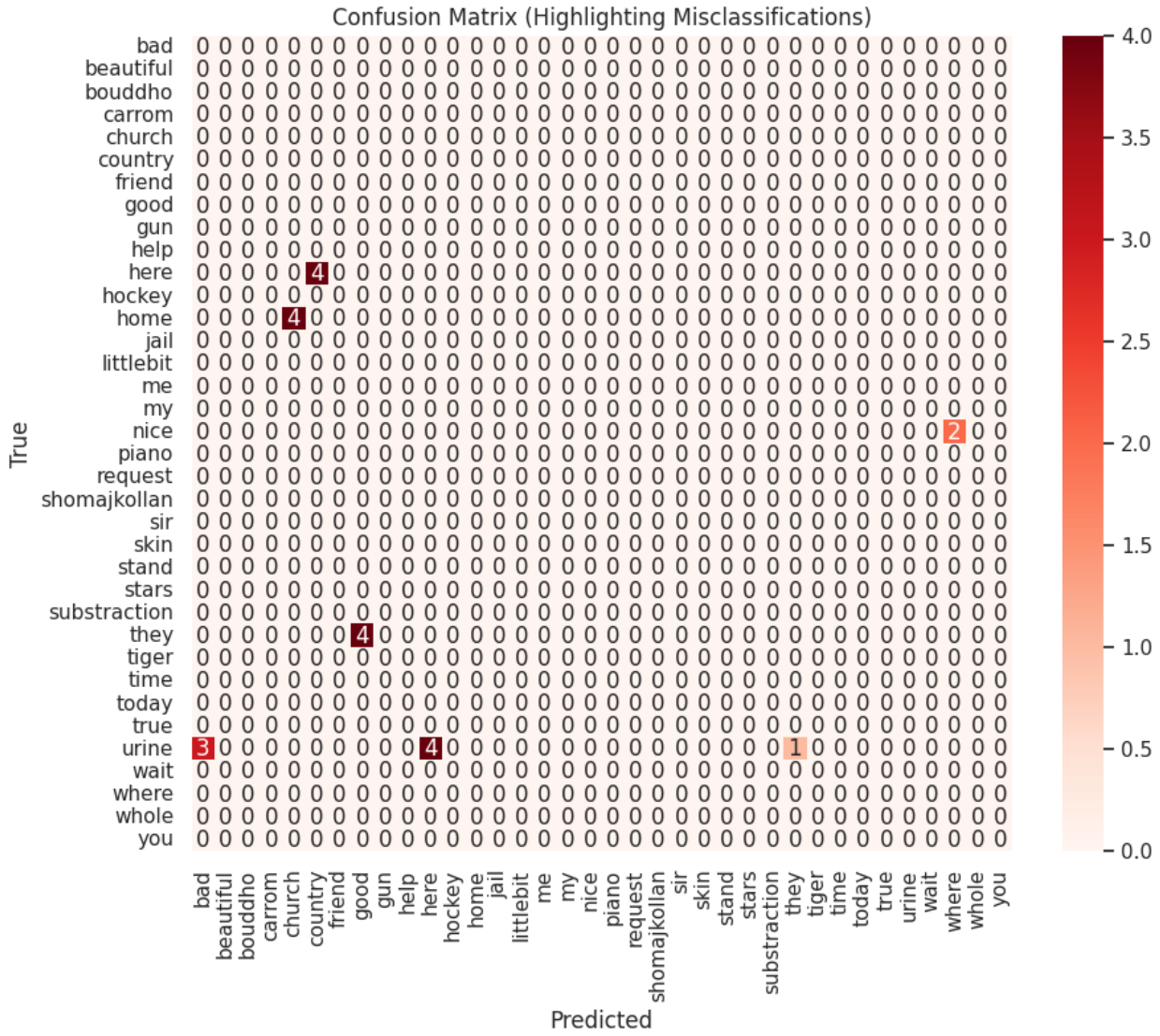
```
Epoch 9/100, Training Loss: 0.0999, Accuracy: 99.92%
Training Precision: 0.9992, Recall: 0.9992, F1 Score: 0.9992
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9996
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.1590, Accuracy: 98.39%
Validation Precision: 0.9846, Recall: 0.9839, F1 Score: 0.9836
Validation ROC AUC: 0.9999, PR AUC: 0.9980, Balanced Accuracy: 0.9816
Validation Top-5 Accuracy: 0.9970

Epoch 10/100: 100%|██████████| 225/225 [00:42<00:00,  5.28it/s,
Loss=0.1122, Accuracy=99.69%, Elapsed=00:07:52, Remaining=01:10:50]

Epoch 10/100, Training Loss: 0.1120, Accuracy: 99.69%
Training Precision: 0.9969, Recall: 0.9969, F1 Score: 0.9969
Training ROC AUC: 1.0000, PR AUC: 0.9991, Balanced Accuracy: 0.9960
Training Top-5 Accuracy: 1.0000
Learning Rate: [7e-05]



Validation Loss: 0.2004, Accuracy: 97.44%
Validation Precision: 0.9767, Recall: 0.9744, F1 Score: 0.9738
Validation ROC AUC: 0.9997, PR AUC: 0.9942, Balanced Accuracy: 0.9695
Validation Top-5 Accuracy: 0.9946
Average Loss Change: 0.030659, Average Accuracy Change: 0.615079

Epoch 11/100: 100%|██████████| 225/225 [00:42<00:00,  5.25it/s,
Loss=0.1034, Accuracy=99.86%, Elapsed=00:08:39, Remaining=01:10:07]

Epoch 11/100, Training Loss: 0.1045, Accuracy: 99.83%
Training Precision: 0.9984, Recall: 0.9983, F1 Score: 0.9983
Training ROC AUC: 1.0000, PR AUC: 0.9997, Balanced Accuracy: 0.9985
Training Top-5 Accuracy: 0.9997
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.2807, Accuracy: 95.30%
Validation Precision: 0.9601, Recall: 0.9530, F1 Score: 0.9488
Validation ROC AUC: 0.9997, PR AUC: 0.9912, Balanced Accuracy: 0.9385
Validation Top-5 Accuracy: 0.9875

Epoch 12/100: 100%|██████████| 225/225 [00:42<00:00,  5.32it/s,
Loss=0.1105, Accuracy=99.58%, Elapsed=00:09:27, Remaining=01:09:18]

Epoch 12/100, Training Loss: 0.1103, Accuracy: 99.58%
Training Precision: 0.9958, Recall: 0.9958, F1 Score: 0.9958
Training ROC AUC: 1.0000, PR AUC: 0.9993, Balanced Accuracy: 0.9940
```

```
Training Top-5 Accuracy: 0.9997
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1872, Accuracy: 97.20%
Validation Precision: 0.9739, Recall: 0.9720, F1 Score: 0.9714
Validation ROC AUC: 0.9999, PR AUC: 0.9978, Balanced Accuracy: 0.9651
Validation Top-5 Accuracy: 0.9946

Epoch 13/100: 100%|██████████| 225/225 [00:42<00:00,  5.31it/s,
Loss=0.0992, Accuracy=99.89%, Elapsed=00:10:14, Remaining=01:08:31]

Epoch 13/100, Training Loss: 0.0992, Accuracy: 99.89%
Training Precision: 0.9989, Recall: 0.9989, F1 Score: 0.9989
Training ROC AUC: 1.0000, PR AUC: 0.9999, Balanced Accuracy: 0.9986
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1739, Accuracy: 97.74%
Validation Precision: 0.9787, Recall: 0.9774, F1 Score: 0.9770
Validation ROC AUC: 0.9999, PR AUC: 0.9980, Balanced Accuracy: 0.9737
Validation Top-5 Accuracy: 0.9946

Epoch 14/100: 100%|██████████| 225/225 [00:42<00:00,  5.28it/s,
Loss=0.0956, Accuracy=99.97%, Elapsed=00:11:01, Remaining=01:07:45]

Epoch 14/100, Training Loss: 0.0956, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 1.0000, PR AUC: 1.0000, Balanced Accuracy: 0.9996
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]



Validation Loss: 0.1789, Accuracy: 97.68%
Validation Precision: 0.9780, Recall: 0.9768, F1 Score: 0.9763
Validation ROC AUC: 0.9999, PR AUC: 0.9971, Balanced Accuracy: 0.9725
Validation Top-5 Accuracy: 0.9964

Epoch 15/100: 100%|██████████| 225/225 [00:42<00:00,  5.28it/s,
Loss=0.0971, Accuracy=99.97%, Elapsed=00:11:49, Remaining=01:06:58]

Epoch 15/100, Training Loss: 0.0971, Accuracy: 99.97%
Training Precision: 0.9997, Recall: 0.9997, F1 Score: 0.9997
Training ROC AUC: 0.9999, PR AUC: 0.9996, Balanced Accuracy: 0.9998
Training Top-5 Accuracy: 1.0000
Learning Rate: [4.899999999999999e-05]
```

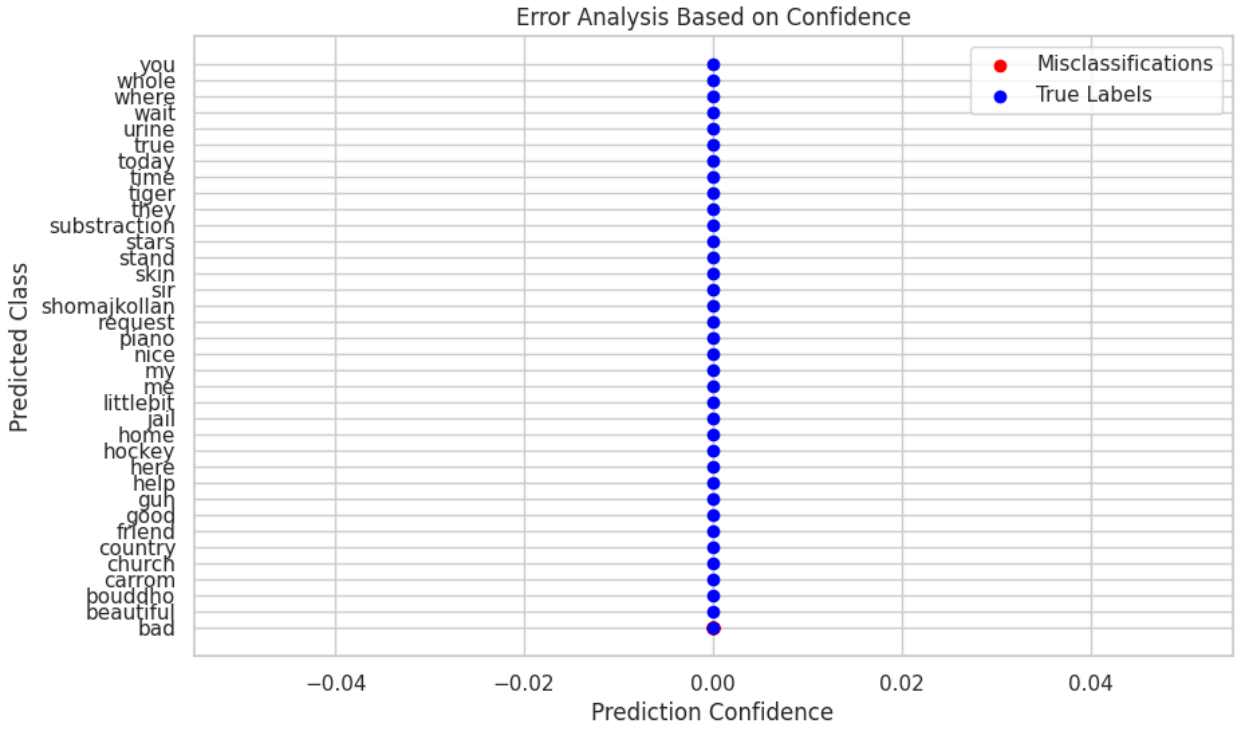
```
Validation Loss: 0.1784, Accuracy: 97.92%
Validation Precision: 0.9803, Recall: 0.9792, F1 Score: 0.9790
Validation ROC AUC: 0.9999, PR AUC: 0.9974, Balanced Accuracy: 0.9776
Validation Top-5 Accuracy: 0.9940
Average Loss Change: -0.032878, Average Accuracy Change: -0.694444
Early stopping at epoch 15

<ipython-input-4-ffb78043919f>:294: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  saved_state = torch.load(model_path, map_location=device)

Test Loss: 0.1897, Test Accuracy: 97.28%
Test Precision: 0.9758, Recall: 0.9728, F1 Score: 0.9729
Test ROC AUC: 0.9997, PR AUC:0.9960 Balanced Accuracy: 0.9778
Test Top-5 Accuracy: 0.9975
```




```
Confusion Matrix:
```




```
Error Analysis:
```



Weighted Prediction

```
test_dir = '/content/BdSLW41WordDatasetSplittedAugmented/test1'

class_names = sorted(os.listdir(test_dir))

# Create a dictionary mapping class name to its index
class_to_idx = {class_name: idx for idx, class_name in
enumerate(class_names)}

# Initialize a list to store the one-hot encoded labels
one_hot_labels = []

# Define supported image extensions
supported_extensions = {'.jpg', '.jpeg', '.png', '.bmp', '.gif'}

# Iterate through each class folder and generate the one-hot encoded
label for each image
for class_name, class_idx in class_to_idx.items():
    class_folder = os.path.join(test_dir, class_name)

    # Loop through each image in the current class folder
    for image_name in os.listdir(class_folder):
        if any(image_name.endswith(ext) for ext in
supported_extensions):  # Check for supported image extensions
            # Create a one-hot vector for the true class label
            one_hot = np.zeros(len(class_names))
            one_hot[class_idx] = 1  # Set the corresponding class
```

```
index to 1
            one_hot_labels.append(one_hot)

# Convert the one-hot encoded labels into a DataFrame
labels_df = pd.DataFrame(one_hot_labels, columns=[f'Class_{i}' for i
in range(len(class_names))])

# Save the DataFrame to a CSV file without the image paths
labels_df.to_csv('/content/true_labels.csv', index=False)

print("True labels with one-hot encoding saved to
'/content/true_labels.csv'")
```

```
True labels with one-hot encoding saved to '/content/true_labels.csv'
```

```
# Load the true labels (one-hot encoded)
true_labels_df = pd.read_csv('true_labels.csv')
true_labels = torch.tensor(true_labels_df.values)  # Shape:
[num_images, num_classes]

# Load the predictions from 6 CSV files
pred_csv_files = [
    ("ConvNeXt",
f"{folder_name}/ConvNeXt_{model_best_training_epoch[0]}.csv"),
    ("DenseNet",
f"{folder_name}/DenseNet_{model_best_training_epoch[1]}.csv"),
    ("EfficientNet",
f"{folder_name}/EfficientNet_{model_best_training_epoch[2]}.csv"),
    ("RegNet",
f"{folder_name}/RegNet_{model_best_training_epoch[3]}.csv"),
    ("ResNet",
f"{folder_name}/ResNet_{model_best_training_epoch[4]}.csv"),
    ("SwinTransformer",
f"{folder_name}/SwinTransformer_{model_best_training_epoch[5]}.csv"),
]
prediction_tensors = {}
# Load predictions from CSV files
for model_name, file_name in pred_csv_files:
    pred_df = pd.read_csv(file_name)
    prediction_tensors[f"predictions_{model_name}"] =
torch.tensor(pred_df.values, dtype=torch.float)

# Verify the tensors are loaded correctly
for model_name, tensor in prediction_tensors.items():
    print(f"{model_name} tensor shape: {tensor.shape}")

print(f"true label shape:{true_labels.shape}")
```

```
predictions_ConvNeXt tensor shape: torch.Size([808, 36])
predictions_DenseNet tensor shape: torch.Size([808, 36])
predictions_EfficientNet tensor shape: torch.Size([808, 36])
```

```
predictions_RegNet tensor shape: torch.Size([808, 36])
predictions_ResNet tensor shape: torch.Size([808, 36])
predictions_SwinTransformer tensor shape: torch.Size([808, 36])
true label shape:torch.Size([808, 36])
```

```
weights = [0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5,
0.55, 0.6, 0.65, 0.7, 0.75]
num_classes = 6

# Function to find valid combinations of weights that sum to 1
def find_valid_combinations(weights, target_sum, num_classes):
    valid_combinations = []
    all_combinations = itertools.product(weights, repeat=num_classes)
    for comb in all_combinations:
        if round(sum(comb), 1) == 1:
            valid_combinations.append(comb)
    return valid_combinations

valid_combinations = find_valid_combinations(weights, 1, num_classes)

# Display the shape of the combinations (number of valid combinations
and number of classes)
print(f"Number of valid combinations: {len(valid_combinations)}")
print(f"Shape of valid combinations: ({len(valid_combinations)},
{num_classes})")
```

```
Number of valid combinations: 18280
Shape of valid combinations: (18280, 6)
```

```
# Initialize lists to store accuracies and RMSE values
accuracies = []
rmses = []
rmsles =[]

# Iterate over all valid weight combinations
for comb in valid_combinations:
    # Initialize a tensor for weighted predictions, shape:
[num_images, num_classes]
    weighted_preds = torch.zeros_like(true_labels)

    # Multiply each model's prediction tensor by its corresponding
weight and sum them
    for idx, (model_name, pred_tensor) in
enumerate(prediction_tensors.items()):
        weighted_preds += pred_tensor * comb[idx]  # Multiply and
accumulate

    # Accuracy: Choose the class with the highest weighted prediction
    predicted_classes = torch.argmax(weighted_preds, dim=1)  # Get the
predicted class indices
    true_classes = torch.argmax(true_labels, dim=1)  # Get the true
```

```
class indices
    accuracy = (predicted_classes ==
true_classes).float().mean().item()  # Calculate accuracy
    accuracies.append(accuracy)

    # RMSE: Compute Root Mean Squared Error
    rmse = root_mean_squared_error(true_labels.numpy(),
weighted_preds.numpy()) # Calculate RMSE
    rmses.append(rmse)

    rmsle = root_mean_squared_log_error(true_labels.numpy(),
weighted_preds.numpy())
    rmsles.append(rmsle)

# Print the results: Combination, Accuracy, RMSE
# for i, comb in enumerate(valid_combinations):
#     print(f"Combination {i+1}: Weights: {comb}, Accuracy:
{accuracies[i]}, RMSE: {rmses[i]}")

# Find the indices of the top 5 combinations based on accuracy
top_5_accuracy_indices = np.argpartition(accuracies, -10)[-10:]
top_5_accuracy_combinations = [valid_combinations[i] for i in
top_5_accuracy_indices]

# Find the indices of the top 5 combinations based on RMSE (lowest
RMSE is better)
top_5_rmse_indices = np.argpartition(rmses, 10)[:10]  # Get indices of
5 lowest RMSE values
top_5_rmse_combinations = [valid_combinations[i] for i in
top_5_rmse_indices]

# Find the indices of the top 5 combinations based on RMSLE (lowest
RMSLE is better)
top_5_rmsle_indices = np.argpartition(rmsles, 10)[:10]
top_5_rmsle_combinations = [valid_combinations[i] for i in
top_5_rmsle_indices]

# Find common combinations between top 5 accuracy and top 5 RMSE
common_combinations = list(set(top_5_accuracy_combinations) &
set(top_5_rmse_combinations) & set(top_5_rmsle_combinations))

# Print the top 5 combinations and their accuracies
print("Top 5 Combinations (Based on Accuracy):")
for i, idx in enumerate(top_5_accuracy_indices):
    print(f"Combination {idx+1}: Weights: {valid_combinations[idx]},
Accuracy: {accuracies[idx]}")

# Print the top 5 combinations and their RMSEs
print("\nTop 5 Combinations (Based on RMSE):")
for i, idx in enumerate(top_5_rmse_indices):
```

```
    print(f"Combination {idx+1}: Weights: {valid_combinations[idx]},
RMSE: {rmses[idx]}")

# Print the top 5 combinations and their RMSLEs
print("\nTop 5 Combinations (Based on RMSLE):")
for i, idx in enumerate(top_5_rmsle_indices):
    print(f"Combination {idx+1}: Weights: {valid_combinations[idx]},
RMSLE: {rmsles[idx]}")

# Normalize accuracy and RMSE to the range [0, 1]
# Convert accuracies and rmses to NumPy arrays
accuracies_np = np.array(accuracies)
rmses_np = np.array(rmses)
rmsles_np = np.array(rmsles)

normalized_accuracies = (accuracies_np - accuracies_np.min()) /
(accuracies_np.max() - accuracies_np.min())
normalized_rmses = 1 - (rmses_np - rmses_np.min()) / (rmses_np.max() -
rmses_np.min())  # Invert RMSE for higher is better
normalized_rmsles = 1 - (rmsles_np - rmsles_np.min()) /
(rmsles_np.max() - rmsles_np.min())

# Create a combined score (e.g., weighted average of accuracy and
RMSE)
# You can adjust the weights (0.7 and 0.3 here) based on your
preference
combined_scores = 0.6 * normalized_accuracies + 0.25 *
normalized_rmses + 0.15 * normalized_rmsles

# Find the index of the combination with the highest combined score
best_combined_index = np.argmax(combined_scores)

# Get the best combination and its accuracy and RMSE
best_combined_weights = valid_combinations[best_combined_index]
best_combined_accuracy = accuracies[best_combined_index]
best_combined_rmse = rmses[best_combined_index]
best_combined_rmsle = rmsles[best_combined_index]

# Print the best combination
print(f"\nBest Combination: Weights: {best_combined_weights},
Accuracy: {best_combined_accuracy:.6f}, RMSE:
{best_combined_rmse:.6f}, RMLSE: {best_combined_rmsle:.6f}")
```

```
Top 5 Combinations (Based on Accuracy):
Combination 8490: Weights: (0.1, 0.5, 0.1, 0.1, 0.1, 0.15), Accuracy:
0.9950494766235352
Combination 8491: Weights: (0.1, 0.5, 0.1, 0.1, 0.15, 0.05), Accuracy:
0.9950494766235352
Combination 8498: Weights: (0.1, 0.5, 0.15, 0.05, 0.05, 0.15),
Accuracy: 0.9950494766235352
```

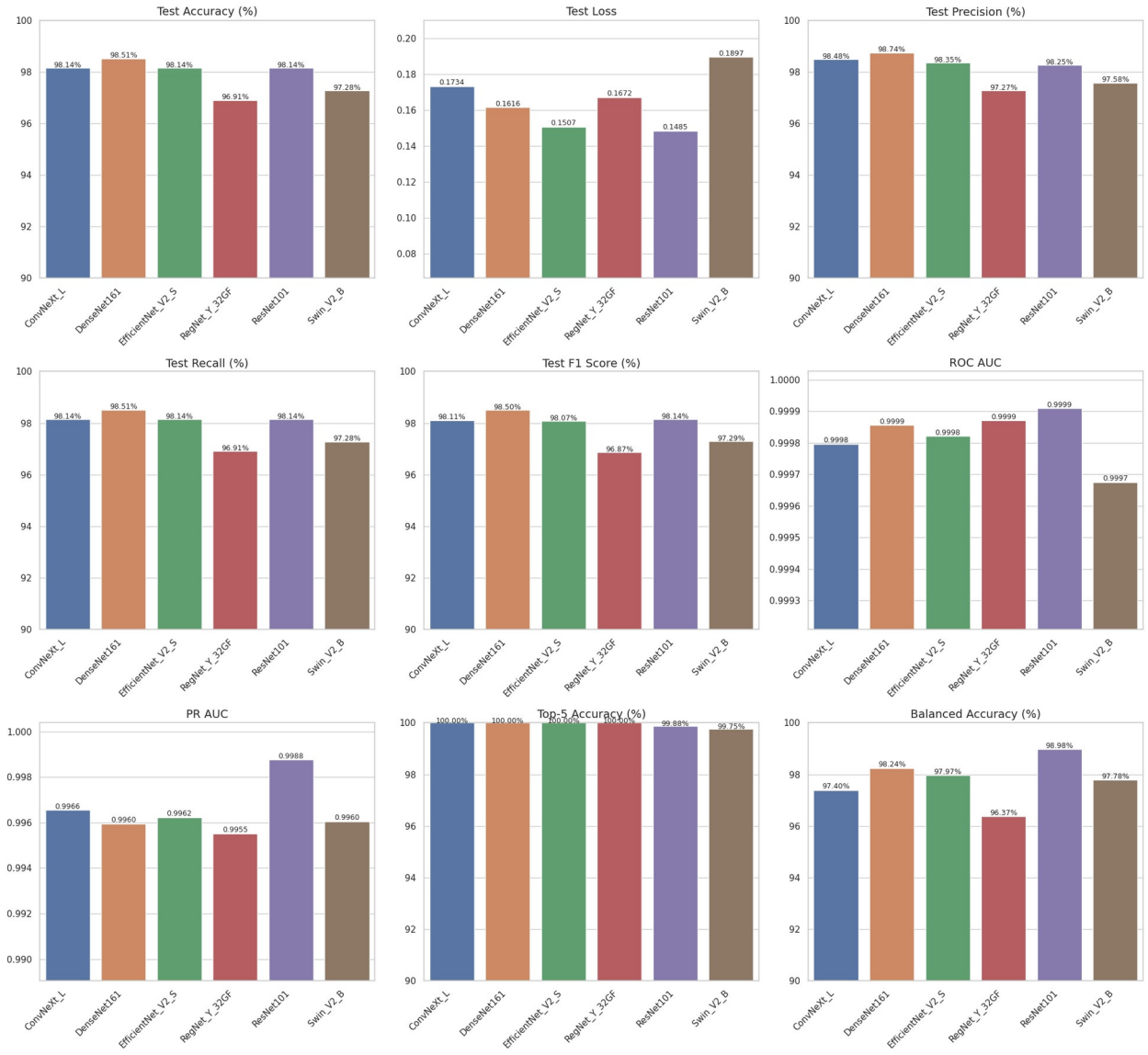
```
Combination 13483: Weights: (0.2, 0.4, 0.05, 0.15, 0.1, 0.1),
Accuracy: 0.9950494766235352
Combination 13478: Weights: (0.2, 0.4, 0.05, 0.1, 0.15, 0.1),
Accuracy: 0.9950494766235352
Combination 13481: Weights: (0.2, 0.4, 0.05, 0.15, 0.05, 0.15),
Accuracy: 0.9950494766235352
Combination 13160: Weights: (0.2, 0.25, 0.15, 0.05, 0.2, 0.1),
Accuracy: 0.9950494766235352
Combination 8489: Weights: (0.1, 0.5, 0.1, 0.1, 0.1, 0.1), Accuracy:
0.9950494766235352
Combination 8488: Weights: (0.1, 0.5, 0.1, 0.1, 0.05, 0.15), Accuracy:
0.9950494766235352
Combination 11099: Weights: (0.15, 0.35, 0.05, 0.05, 0.2, 0.15),
Accuracy: 0.9950494766235352

Top 5 Combinations (Based on RMSE):
Combination 5682: Weights: (0.1, 0.05, 0.6, 0.05, 0.15, 0.05), RMSE:
0.013743623332114475
Combination 1944: Weights: (0.05, 0.1, 0.6, 0.05, 0.15, 0.05), RMSE:
0.013752771299095164
Combination 1067: Weights: (0.05, 0.05, 0.7, 0.05, 0.1, 0.05), RMSE:
0.01372601575628507
Combination 5668: Weights: (0.1, 0.05, 0.55, 0.05, 0.2, 0.05), RMSE:
0.013753271550200986
Combination 1060: Weights: (0.05, 0.05, 0.65, 0.05, 0.15, 0.05), RMSE:
0.013702634682633536
Combination 1046: Weights: (0.05, 0.05, 0.6, 0.05, 0.2, 0.05), RMSE:
0.013707630743521344
Combination 1025: Weights: (0.05, 0.05, 0.55, 0.05, 0.25, 0.05), RMSE:
0.013737650093221273
Combination 1930: Weights: (0.05, 0.1, 0.55, 0.05, 0.2, 0.05), RMSE:
0.013761486500669188
Combination 1951: Weights: (0.05, 0.1, 0.65, 0.05, 0.1, 0.05), RMSE:
0.013773464631120459
Combination 5689: Weights: (0.1, 0.05, 0.65, 0.05, 0.1, 0.05), RMSE:
0.013762729450913953

Top 5 Combinations (Based on RMSLE):
Combination 1067: Weights: (0.05, 0.05, 0.7, 0.05, 0.1, 0.05), RMSLE:
0.009615907493228505
Combination 1930: Weights: (0.05, 0.1, 0.55, 0.05, 0.2, 0.05), RMSLE:
0.009632447938320072
Combination 1944: Weights: (0.05, 0.1, 0.6, 0.05, 0.15, 0.05), RMSLE:
0.009624967963205812
Combination 1060: Weights: (0.05, 0.05, 0.65, 0.05, 0.15, 0.05),
RMSLE: 0.009605695625500722
Combination 1046: Weights: (0.05, 0.05, 0.6, 0.05, 0.2, 0.05), RMSLE:
0.009612957452150687
Combination 1025: Weights: (0.05, 0.05, 0.55, 0.05, 0.25, 0.05),
```

```
RMSLE: 0.00963479212628796
Combination 1070: Weights: (0.05, 0.05, 0.75, 0.05, 0.05, 0.05),
RMSLE: 0.009649619871466055
Combination 1951: Weights: (0.05, 0.1, 0.65, 0.05, 0.1, 0.05), RMSLE:
0.009636038177913397
Combination 1908: Weights: (0.05, 0.1, 0.5, 0.05, 0.25, 0.05), RMSLE:
0.009655563090341079
Combination 5682: Weights: (0.1, 0.05, 0.6, 0.05, 0.15, 0.05), RMSLE:
0.009660629304713602

Best Combination: Weights: (0.15, 0.35, 0.2, 0.05, 0.2, 0.05),
Accuracy: 0.995049, RMSE: 0.014754, RMLSE: 0.010330

import matplotlib.pyplot as plt
import seaborn as sns

# Data for the models
pretrained_models = [
    "ConvNeXt_L", "DenseNet161", "EfficientNet_V2_S", "RegNet_Y_32GF",
"ResNet101", "Swin_V2_B"
]

metrics = {
    "Test Accuracy (%)": [val * 1 for val in all_test_accuracies],
    "Test Loss": all_test_losses,
    "Test Precision (%)": [val * 100 for val in all_test_precisions],
    "Test Recall (%)": [val * 100 for val in all_test_recalls],
    "Test F1 Score (%)": [val * 100 for val in all_test_f1_scores],
    "ROC AUC": all_test_roc_auc_scores,
    "PR AUC": all_test_pr_auc_scores,
    "Top-5 Accuracy (%)": [val * 1 for val in
all_test_top_5_accuracies],
    "Balanced Accuracy (%)": [val * 1 for val in
all_test_balanced_accuracies]
}

epochs_ran = model_training_epoch

# Create subplots for each metric
fig, axes = plt.subplots(3, 3, figsize=(20, 20))
axes = axes.flatten()

# Add the metrics to the plots
for idx, (metric_name, metric_values) in enumerate(metrics.items()):
    sns.barplot(
        x=pretrained_models,
        y=metric_values,
        ax=axes[idx],
        hue=pretrained_models,  # Added `hue` to resolve warning
        dodge=False  # No legend required for repeated hues
```

```
    )
    axes[idx].set_title(metric_name, fontsize=14)

    # Adjust y-axis limits dynamically
    y_min, y_max = min(metric_values), max(metric_values)
    y_range = y_max - y_min

    # Set the y-axis range with extra padding
    if metric_name.endswith("(%)"):
        if y_max > 90:  # If all values are in the last 10%
            axes[idx].set_ylim(90, 100)
        else:
            axes[idx].set_ylim(0, 100)
    else:
        axes[idx].set_ylim(y_min-(y_range * 2), y_max + (y_range*0.5))
# 10% padding

    # Show only up to 100 for percentage metrics
    if metric_name.endswith("(%)") and y_max <= 100:
        axes[idx].set_ylim(top=100)

    # Add exact values on top of bars
    for i, value in enumerate(metric_values):
      text_value = f"{value:.4f}" if not metric_name.endswith("(%)")
else f"{value:.2f}%"
      # Ensure the text stays within bounds
      text_y = min(value + (y_range * 0.02), axes[idx].get_ylim()[1] -
(y_range * 0.01))
      axes[idx].text(i, text_y, text_value, ha="center", fontsize=9)



    axes[idx].set_xticks(range(len(pretrained_models)))  # Explicitly
set tick locations
    axes[idx].set_xticklabels(pretrained_models, rotation=45,
ha="right")

# Add text at the bottom
best_combination_text = (
    f"Best Combination: Weights: {best_combined_weights}, "
    f"Accuracy: {best_combined_accuracy*100:.4f}%, RMSE:
{best_combined_rmse:.6f}, RMLSE: {best_combined_rmsle:.6f}\n"
    f"Number of epochs each model ran: {epochs_ran}"
)

fig.suptitle(best_combination_text, fontsize=16, y=0.95)  # Adjusted
fontsize and title placement
fig.tight_layout(rect=[0, 0, 1, 0.95])
# plt.subplots(layout="constrained")
```

```
plt.show()

# Normalize metric values
normalized_metrics = {}

for metric_name, metric_values in metrics.items():
    metric_values = np.array(metric_values)
    if "Loss" in metric_name:  # For loss, lower is better
        normalized = 1 - (metric_values - metric_values.min()) /
(metric_values.max() - metric_values.min())
    else:  # For other metrics, higher is better
        normalized = (metric_values - metric_values.min()) /
(metric_values.max() - metric_values.min())
    normalized_metrics[metric_name] = normalized

# Create a DataFrame for normalized scores
normalized_df = pd.DataFrame(normalized_metrics,
index=pretrained_models)

# Calculate total scores for each model
normalized_df["Total Score"] = normalized_df.sum(axis=1)

# Rank models based on total score
normalized_df["Rank"] = normalized_df["Total
Score"].rank(ascending=False)

# Sort by rank
ranking_df = normalized_df.sort_values("Rank")

# Combine original metric values with rank
final_ranking = pd.DataFrame(metrics, index=pretrained_models)
final_ranking["Total Score"] = ranking_df["Total Score"]
final_ranking["Rank"] = ranking_df["Rank"].astype(int)

# Sort by rank for display
final_ranking = final_ranking.sort_values("Rank")

print(final_ranking)

# Display in one-line format
for idx, row in final_ranking.iterrows():
    metrics_str = ", ".join([f"{metric}: {value}" for metric, value in
row.items() if metric != "Rank"])
    print(f"Rank {int(row['Rank'])}: {idx} ({metrics_str})")
```

Best Combination: Weights: (0.15, 0.35, 0.2, 0.05, 0.2, 0.05), Accuracy: 99.5049%, RMSE: 0.014754, RMLSE: 0.010330
Number of epochs each model ran: [30, 45, 20, 15, 20, 15]



	Test Accuracy (%)	Test Loss	Test Precision (%)	\
ResNet101	98.143564	0.148505	98.252598	
DenseNet161	98.514851	0.161605	98.740351	
EfficientNet_V2_S	98.143564	0.150691	98.348048	
ConvNeXt_L	98.143564	0.173355	98.478045	
RegNet_Y_32GF	96.905941	0.167239	97.269459	
Swin_V2_B	97.277228	0.189694	97.575922	

	Test Recall (%)	Test F1 Score (%)	ROC AUC	PR AUC	\
ResNet101	98.143564	98.140015	0.999909	0.998766	
DenseNet161	98.514851	98.497536	0.999856	0.995952	

```
EfficientNet_V2_S        98.143564          98.074311  0.999821
0.996237
ConvNeXt_L               98.143564          98.107839  0.999796
0.996553
RegNet_Y_32GF            96.905941          96.866694  0.999871
0.995525
Swin_V2_B                97.277228          97.285943  0.999676
0.996048

                    Top-5 Accuracy (%)  Balanced Accuracy (%)  Total
Score  \
ResNet101                    99.876238              98.981131
7.487633
DenseNet161                 100.000000              98.244599
7.302458
EfficientNet_V2_S           100.000000              97.969276
6.411858
ConvNeXt_L                  100.000000              97.395833
5.743266
RegNet_Y_32GF               100.000000              96.373457
2.381260
Swin_V2_B                    99.752475              97.779882
1.627617

                    Rank
ResNet101              1
DenseNet161            2
EfficientNet_V2_S      3
ConvNeXt_L             4
RegNet_Y_32GF          5
Swin_V2_B              6
Rank 1: ResNet101 (Test Accuracy (%): 98.14356435643565, Test Loss:
0.148504772151892, Test Precision (%): 98.25259785818217, Test Recall
(%): 98.14356435643565, Test F1 Score (%): 98.14001478084397, ROC AUC:
0.9999090176979119, PR AUC: 0.9987655391478998, Top-5 Accuracy (%):
99.87623762376238, Balanced Accuracy (%): 98.98113075196409, Total
Score: 7.487632824909546)
Rank 2: DenseNet161 (Test Accuracy (%): 98.51485148514851, Test Loss:
0.16160541113752586, Test Precision (%): 98.74035115132746, Test
Recall (%): 98.51485148514851, Test F1 Score (%): 98.49753636094084,
ROC AUC: 0.9998556004133506, PR AUC: 0.9959520488691219, Top-5
Accuracy (%): 100.0, Balanced Accuracy (%): 98.2445987654321, Total
Score: 7.3024584592975295)
Rank 3: EfficientNet_V2_S (Test Accuracy (%): 98.14356435643565, Test
Loss: 0.15069104935608657, Test Precision (%): 98.34804828252702, Test
Recall (%): 98.14356435643565, Test F1 Score (%): 98.07431112493728,
ROC AUC: 0.9998205674673524, PR AUC: 0.996236532748469, Top-5 Accuracy
(%): 100.0, Balanced Accuracy (%): 97.96927609427608, Total Score:
6.411858152647278)
```

```
Rank 4: ConvNeXt_L (Test Accuracy (%): 98.14356435643565, Test Loss: 0.1733549510439237, Test Precision (%): 98.47804461297193, Test Recall (%): 98.14356435643565, Test F1 Score (%): 98.10783868233013, ROC AUC: 0.9997960670339714, PR AUC: 0.9965525908718553, Top-5 Accuracy (%): 100.0, Balanced Accuracy (%): 97.39583333333331, Total Score: 5.743265510240546)
Rank 5: RegNet_Y_32GF (Test Accuracy (%): 96.9059405940594, Test Loss: 0.16723936065739275, Test Precision (%): 97.26945868471785, Test Recall (%): 96.9059405940594, Test F1 Score (%): 96.86669381558221, ROC AUC: 0.999870746711672, PR AUC: 0.9955252004922971, Top-5 Accuracy (%): 100.0, Balanced Accuracy (%): 96.37345679012346, Total Score: 2.3812600120891747)
Rank 6: Swin_V2_B (Test Accuracy (%): 97.27722772277228, Test Loss: 0.18969404025405062, Test Precision (%): 97.57592242740759, Test Recall (%): 97.27722772277228, Test F1 Score (%): 97.2859429863582, ROC AUC: 0.9996755134915594, PR AUC: 0.9960478994510034, Top-5 Accuracy (%): 99.75247524752476, Balanced Accuracy (%): 97.77988215488216, Total Score: 1.6276168349318798)

GPU_unload()

def load_architectures():
    # Load architectures
    convnext = convnext_large(pretrained=False, num_classes=36).to("cuda")
    # Load the state dictionary using torch.load and then pass it to load_state_dict
    state_dict = torch.load(f"{folder_name}/ConvNeXt_{model_best_training_epoch[0]}.pth")
    convnext.load_state_dict(state_dict['model_state_dict'])

    densenet = densenet161(pretrained=False, num_classes=36).to("cuda")
    # Load the state dictionary using torch.load and then pass it to load_state_dict
    state_dict = torch.load(f"{folder_name}/DenseNet_{model_best_training_epoch[1]}.pth")
    densenet.load_state_dict(state_dict['model_state_dict'])

    efficientnet =  efficientnet_v2_s(pretrained=False, num_classes=36).to("cuda")
    # Load the state dictionary using torch.load and then pass it to load_state_dict
    state_dict = torch.load(f"{folder_name}/EfficientNet_{model_best_training_epoch[2]}.pth")
    efficientnet.load_state_dict(state_dict['model_state_dict'])
```

```
    regnet = regnet_y_32gf(pretrained=False,
num_classes=36).to("cuda")
    # Load the state dictionary using torch.load and then pass it to
load_state_dict
    state_dict =
torch.load(f"{folder_name}/RegNet_{model_best_training_epoch[3]}.pth")
    regnet.load_state_dict(state_dict['model_state_dict'])

    resnet = resnext101_64x4d(pretrained=False,
num_classes=36).to("cuda")
    # Load the state dictionary using torch.load and then pass it to
load_state_dict
    state_dict =
torch.load(f"{folder_name}/ResNet_{model_best_training_epoch[4]}.pth")
    resnet.load_state_dict(state_dict['model_state_dict'])

    swin = swin_v2_b(pretrained=False, num_classes=36).to("cuda")
    state_dict =
torch.load(f"{folder_name}/SwinTransformer_{model_best_training_epoch[
5]}.pth")
    swin.load_state_dict(state_dict['model_state_dict'])

    return convnext, densenet, efficientnet, regnet, resnet, swin

convnext, densenet, efficientnet, regnet, resnet, swin =
load_architectures()

fine_models = [convnext, densenet, efficientnet, regnet, resnet, swin]
for fine_model in fine_models:
    fine_model.eval()
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
<ipython-input-21-cd410b975fbd>:7: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
```

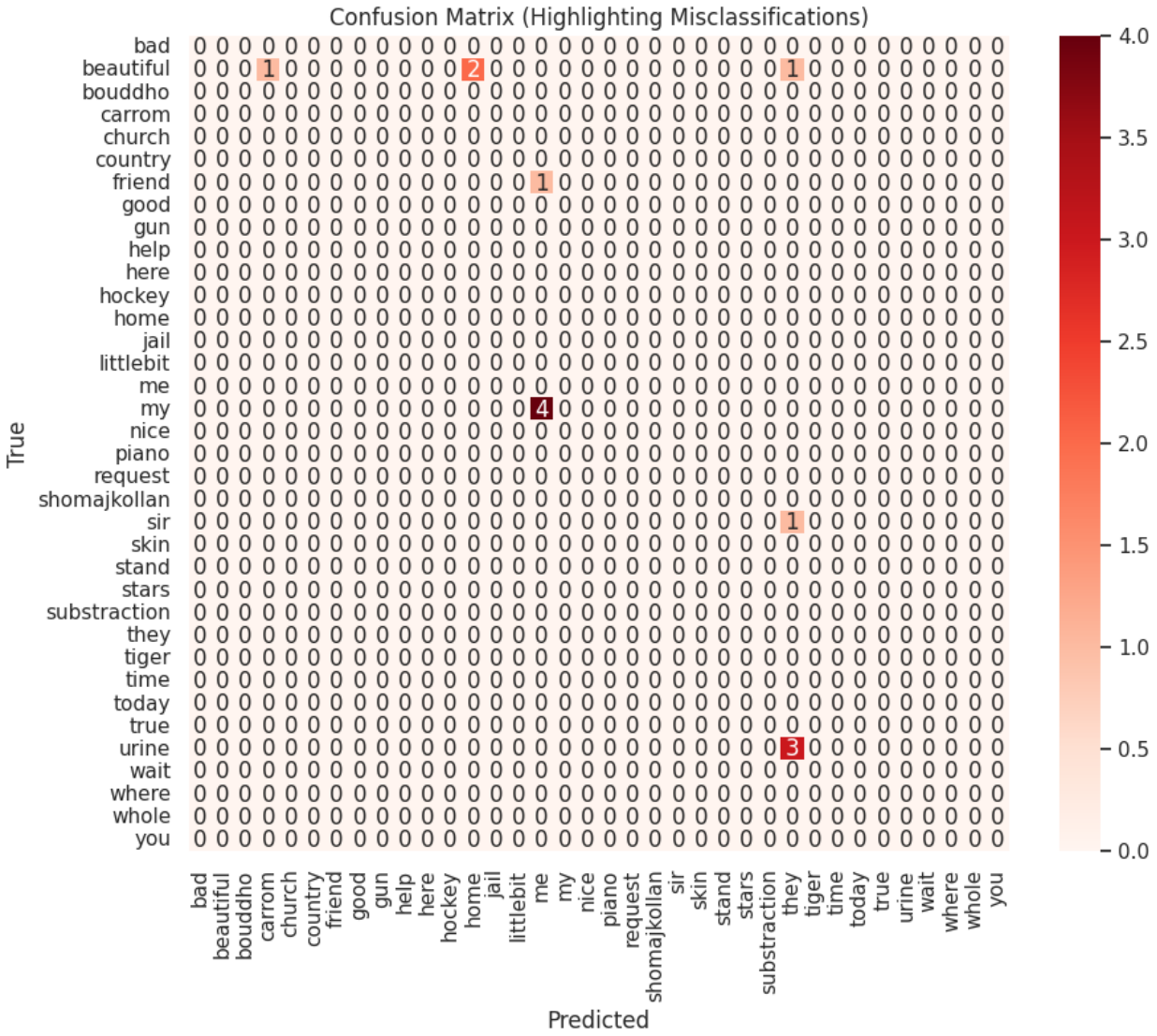
```
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/ConvNeXt_{model_best_training_epoch[0]}.pth
")
<ipython-input-21-cd410b975fbd>:12: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/DenseNet_{model_best_training_epoch[1]}.pth
")
<ipython-input-21-cd410b975fbd>:17: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/EfficientNet_{model_best_training_epoch[2]}
.pth")
<ipython-input-21-cd410b975fbd>:22: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
```

```
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/RegNet_{model_best_training_epoch[3]}.pth")
<ipython-input-21-cd410b975fbd>:27: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/ResNet_{model_best_training_epoch[4]}.pth")
<ipython-input-21-cd410b975fbd>:31: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/SwinTransformer_{model_best_training_epoch[
5]}.pth")
```

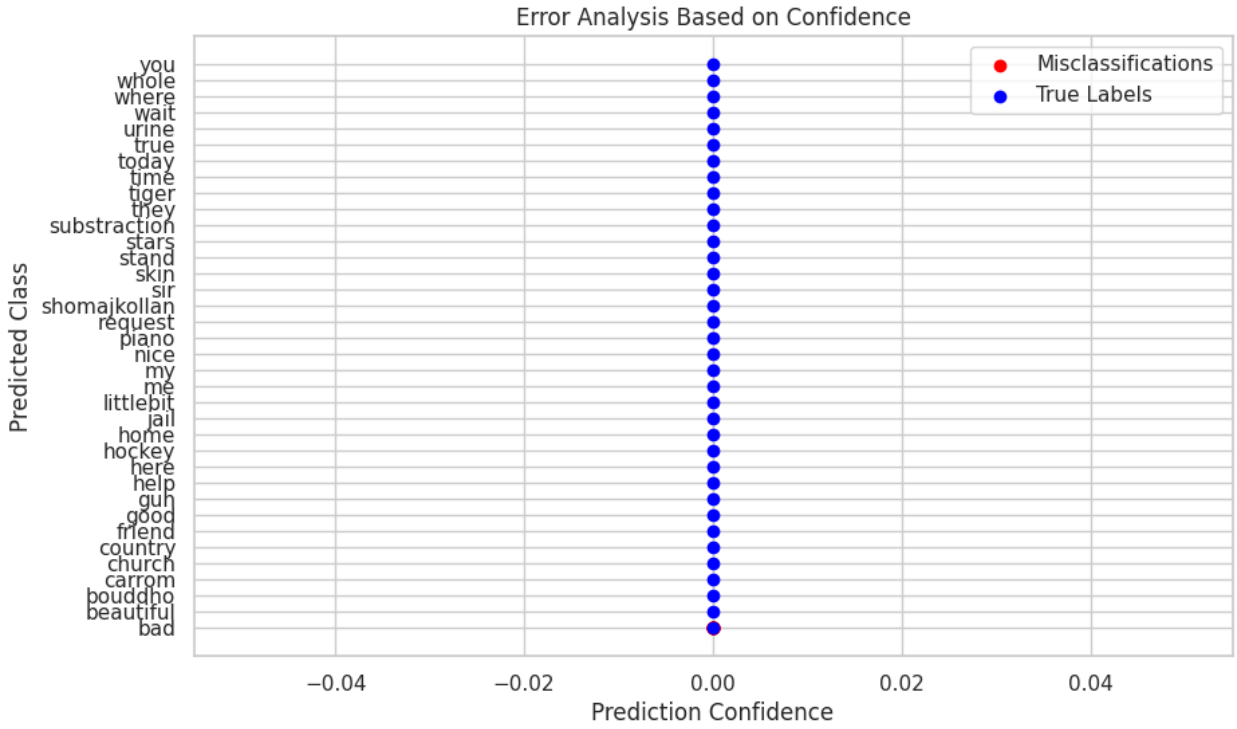
```
def ensemble_predict(models, weights, input_tensor):
    # Ensure weights sum to 1
    weights = [w / sum(weights) for w in weights]

    # Accumulate predictions
    ensemble_output = 0
    for model, weight in zip(models, weights):
        with torch.no_grad():
            output = F.softmax(model(input_tensor), dim=1)
            ensemble_output += weight * output

    return ensemble_output

GPU_unload()

# Retrieve the transformation pipeline from weights_fn
preprocess =
torchvision.models.ConvNeXt_Large_Weights.IMAGENET1K_V1.transforms()

# Define weights (example: equal weights for all models)
weights = best_combined_weights

# Load your test dataset
test_dataset =
datasets.ImageFolder(root="/content/BdSLW41WordDatasetSplittedAugmente
d/test2", transform=preprocess)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)

# Initialize visualizer
model_names = ['ConvNeXt', 'DenseNet', 'EfficientNet', 'RegNet',
'ResNeXt', 'Swin']
class_names = test_dataset.classes

# Lists to store true and predicted labels
all_true_labels = []
all_predicted_labels = []
all_test_probs = []

# Evaluate on the test dataset
correct = 0
total = 0

with torch.no_grad():
    for inputs, labels in tqdm(test_loader):
        inputs, labels = inputs.to("cuda"), labels.to("cuda")

        # Get ensemble predictions
        outputs = ensemble_predict(fine_models, weights, inputs)
        _, predicted = torch.max(outputs, dim=1)

        # Update metrics
```

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

        # Store labels for confusion matrix
        all_true_labels.extend(labels.cpu().numpy())
        all_predicted_labels.extend(predicted.cpu().numpy())
        all_test_probs.extend(outputs.cpu().numpy())

# Calculate test accuracy
test_accuracy = 100 * correct / total

# Calculate metrics

test_precision, test_recall, test_f1, test_roc_auc_score,
test_pr_auc_score, test_balanced_accuracy, test_top_5_accuracy =
calculate_metrics(
        all_true_labels, all_predicted_labels, all_test_probs
    )

# Print test results
print(f"Test Accuracy: {test_accuracy:.4f}%")
print(f"Test Precision: {test_precision:.4f}, Recall:
{test_recall:.4f}, F1 Score: {test_f1:.4f}")
print(f"Test ROC AUC: {test_roc_auc_score:.4f}, PR AUC:
{test_pr_auc_score:.4f} Balanced Accuracy:
{test_balanced_accuracy:.4f}")
print(f"Test Top-5 Accuracy: {test_top_5_accuracy:.4f}")


# Plot confusion matrix and error analysis
print("\nConfusion Matrix:")
dv.plot_confusion_matrix(all_true_labels, all_predicted_labels,
test_dataset.classes)
print("\nError Analysis:")
dv.plot_error_analysis(all_true_labels, all_predicted_labels,
test_dataset.classes)
print("\nModel Metrics:")
```

```
100%|██████████| 8/8 [00:11<00:00,  1.48s/it]

Test Accuracy: 98.6895%
Test Precision: 0.9880, Recall: 0.9869, F1 Score: 0.9869
Test ROC AUC: 0.9999, PR AUC:0.9988 Balanced Accuracy: 0.9913
Test Top-5 Accuracy: 0.9970

Confusion Matrix:
```




```
Error Analysis:
```



Model Metrics:

```
# Define a dictionary to save the models and weights
ensemble_checkpoint = {
    "models": [convnext.state_dict(), densenet.state_dict(),
efficientnet.state_dict(),
               regnet.state_dict(), resnet.state_dict(),
swin.state_dict()],
    "weights": weights,
}

# Save the checkpoint
torch.save(ensemble_checkpoint, "/content/ensemble_model.pth")

GPU_unload()

# Load the saved checkpoint
checkpoint = torch.load("/content/ensemble_model.pth")

# Initialize models (make sure they are defined the same way as in
your code)
convnext = convnext_large(pretrained=False, num_classes=36).to("cuda")
densenet = densenet161(pretrained=False, num_classes=36).to("cuda")
efficientnet = efficientnet_v2_s(pretrained=False,
num_classes=36).to("cuda")
regnet = regnet_y_32gf(pretrained=False, num_classes=36).to("cuda")
resnet = resnext101_64x4d(pretrained=False, num_classes=36).to("cuda")
```

```
swin = swin_v2_b(pretrained=False, num_classes=36).to("cuda")

# Load the state dicts into each model
convnext.load_state_dict(checkpoint["models"][0])
densenet.load_state_dict(checkpoint["models"][1])
efficientnet.load_state_dict(checkpoint["models"][2])
regnet.load_state_dict(checkpoint["models"][3])
resnet.load_state_dict(checkpoint["models"][4])
swin.load_state_dict(checkpoint["models"][5])

# Define weights (load from checkpoint)
weights = checkpoint["weights"]

# Put models in evaluation mode
fine_models = [convnext, densenet, efficientnet, regnet, resnet, swin]
for fine_model in fine_models:
    fine_model.eval()
```

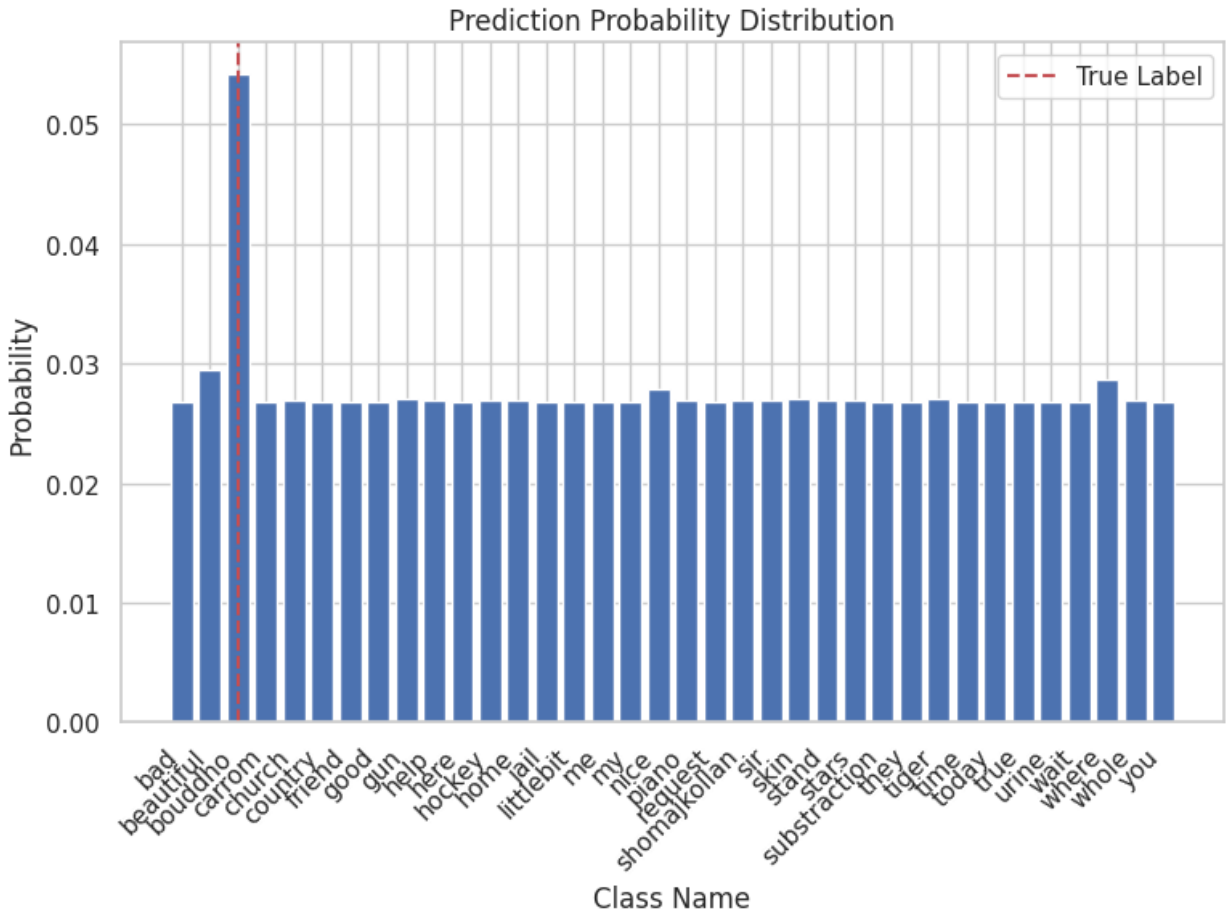
```
<ipython-input-25-a4c42aee91bd>:4: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  checkpoint = torch.load("/content/ensemble_model.pth")
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:2
08: UserWarning: The parameter 'pretrained' is deprecated since 0.13
and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
```

```
# For single image prediction
def predict_and_visualize(img_path, fine_models, weights,
class_names):
    # Load and preprocess image
    img = Image.open(img_path)
    preprocess =
```

```
models.ConvNeXt_Large_Weights.IMAGENET1K_V1.transforms()
    input_tensor = preprocess(img).unsqueeze(0).to("cuda")

    # Get ensemble predictions
    outputs = ensemble_predict(fine_models, weights, input_tensor)
    _, predicted = torch.max(outputs, dim=1)

    # Get predicted class name
    predicted_class_name = class_names[predicted.item()]

    # Visualize image and prediction
    plt.figure(figsize=(15, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(img)
    plt.axis('off')
    plt.title('Input Image')
    plt.tight_layout()
    probs = F.softmax(outputs, dim=1)[0].cpu().numpy()

    plt.subplot(1, 2, 2)
    plt.bar(class_names, probs)
    plt.xticks(rotation=45, ha='right')
    plt.xlabel('Class Name')
    plt.ylabel('Probability')
    plt.title('Prediction Probability Distribution')
    plt.axvline(x=predicted_class_name, color='r', linestyle='--',
label='True Label')
    plt.legend()
    plt.tight_layout()
    plt.show()
    top_k = 5
    top_probs, top_indices = torch.topk(F.softmax(outputs, dim=1)[0],
k=top_k)
    print("\nTop {} Predictions:".format(top_k))
    for i in range(top_k):
        print(f"{class_names[top_indices[i]]}:
{top_probs[i].item()*100:.2f}%")

    return predicted_class_name

# When using the prediction function:
img_path =
"/content/BdSLW41WordDatasetSplittedAugmented/test2/bouddho/bouddho_5_
aug_3.jpg"
predicted_class = predict_and_visualize(img_path, fine_models,
weights, test_dataset.classes)
print(f"\nPredicted class: {predicted_class}")
```

```
<ipython-input-26-e8929a681b82>:24: MatplotlibDeprecationWarning:
Auto-removal of overlapping axes is deprecated since 3.6 and will be
```

```
removed two minor releases later; explicitly call ax.remove() as
needed.
  plt.subplot(1, 2, 2)
```




```
Top 5 Predictions:
bouddho: 5.41%
beautiful: 2.95%
where: 2.87%
nice: 2.79%
tiger: 2.70%

Predicted class: bouddho

GPU_unload()

# Load architectures
convnext, densenet, efficientnet, regnet, resnet, swin =
load_architectures()

fine_models = [convnext, densenet, efficientnet, regnet, resnet, swin]
```

```
for fine_model in fine_models:
    fine_model.eval()

# Freeze models
for model in fine_models:
    for param in model.parameters():
        param.requires_grad = False
```

```
<ipython-input-21-cd410b975fbd>:7: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/ConvNeXt_{model_best_training_epoch[0]}.pth
")
<ipython-input-21-cd410b975fbd>:12: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/DenseNet_{model_best_training_epoch[1]}.pth
")
<ipython-input-21-cd410b975fbd>:17: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
```

```
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/EfficientNet_{model_best_training_epoch[2]}
.pth")
<ipython-input-21-cd410b975fbd>:22: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/RegNet_{model_best_training_epoch[3]}.pth")
<ipython-input-21-cd410b975fbd>:27: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/ResNet_{model_best_training_epoch[4]}.pth")
<ipython-input-21-cd410b975fbd>:31: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
```

```
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/SwinTransformer_{model_best_training_epoch[
5]}.pth")

class EnhancedMetaLearner(nn.Module):
    def __init__(self, input_size, num_classes, dropout_rate=0.35):
        super().__init__()

        # Calculate the total input size (number of features * number
of models)
        self.input_size = input_size

        self.layers = nn.Sequential(
            nn.Linear(input_size, 512),
            nn.BatchNorm1d(512),
            nn.GELU(),
            nn.Dropout(dropout_rate),

            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.GELU(),
            nn.Dropout(dropout_rate),

            nn.Linear(256, 128),
            nn.BatchNorm1d(128),
            nn.GELU(),
            nn.Dropout(dropout_rate),

            nn.Linear(128, num_classes)
        )

        # Initialize weights using He initialization
        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.kaiming_normal_(m.weight)
                nn.init.constant_(m.bias, 0)

    def forward(self, x):
```

```
        # Ensure input is correctly shaped
        x = x.view(-1, self.input_size)
        return self.layers(x)

def prepare_data(fine_models, train_loader, val_loader, test_loader,
device="cuda"):
    def extract_features(models, dataloader):
        features = []
        labels = []

        with torch.no_grad():
            for inputs, targets in tqdm(dataloader, desc="Extracting
features"):
                inputs = inputs.to(device)
                batch_features = []

                for model in models:
                    outputs = model(inputs)
                    batch_features.append(outputs.cpu())

                # Concatenate features from all models
                combined_features = torch.cat(batch_features, dim=1)
                features.append(combined_features)
                labels.append(targets)

        return torch.cat(features, dim=0), torch.cat(labels, dim=0)

    print("Extracting training features...")
    train_features, train_labels = extract_features(fine_models,
train_loader)
    print("Extracting validation features...")
    val_features, val_labels = extract_features(fine_models,
val_loader)
    print("Extracting test features...")
    test_features, test_labels = extract_features(fine_models,
test_loader)

    # Create datasets and dataloaders
    train_dataset = torch.utils.data.TensorDataset(train_features,
train_labels)
    val_dataset = torch.utils.data.TensorDataset(val_features,
val_labels)
    test_dataset = torch.utils.data.TensorDataset(test_features,
test_labels)

    train_loader = DataLoader(train_dataset, batch_size=64,
shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
    test_loader = DataLoader(test_dataset, batch_size=64,
shuffle=False)
```

```
    return train_loader, val_loader, test_loader,
train_features.shape[1]

class EnsembleTrainer:
    def __init__(self, meta_learner, train_loader, val_loader,
test_loader, criterion, optimizer,
                 scheduler, device, class_names):
        self.meta_learner = meta_learner
        self.train_loader = train_loader
        self.val_loader = val_loader
        self.test_loader = test_loader
        self.criterion = criterion
        self.optimizer = optimizer
        self.scheduler = scheduler
        self.device = device
        self.class_names = class_names

    def train_epoch(self):
        self.meta_learner.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for inputs, targets in tqdm(self.train_loader,
desc="Training"):
            inputs = inputs.to(self.device)
            targets = targets.to(self.device)

            self.optimizer.zero_grad()
            outputs = self.meta_learner(inputs)
            loss = self.criterion(outputs, targets)

            loss.backward()

torch.nn.utils.clip_grad_norm_(self.meta_learner.parameters(),
max_norm=1.0)
            self.optimizer.step()

            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

        return running_loss / len(self.train_loader), 100. * correct /
total

    def validate(self):
        self.meta_learner.eval()
        running_loss = 0.0
```

```
        correct = 0
        total = 0
        all_preds = []
        all_targets = []
        all_probs = []

        with torch.no_grad():
            for inputs, targets in tqdm(self.val_loader,
desc="Validating"):
                inputs = inputs.to(self.device)
                targets = targets.to(self.device)

                outputs = self.meta_learner(inputs)
                loss = self.criterion(outputs, targets)
                probs = F.softmax(outputs, dim=1).type(torch.float32)

                # print("Sum of probabilities:", probs.sum(dim=1))
                assert
np.allclose(probs.sum(dim=1).cpu().detach().numpy(), 1.0, atol=1e-6),
"Probabilities do not sum to 1"

                running_loss += loss.item()
                _, predicted = torch.max(probs, 1)
                total += targets.size(0)
                correct += predicted.eq(targets).sum().item()

                all_preds.extend(predicted.cpu().numpy())
                all_targets.extend(targets.cpu().numpy())
                all_probs.extend(probs.cpu().detach().numpy())

        return running_loss / len(self.val_loader), 100. * correct /
total, all_preds, all_targets, all_probs

    def train(self, num_epochs):
        history = {
            'train_loss': [], 'train_acc': [],
            'val_loss': [], 'val_acc': []
        }
        best_val_acc = 0

        for epoch in range(num_epochs):
            print(f"\nEpoch {epoch+1}/{num_epochs}")

            train_loss, train_acc = self.train_epoch()
            val_loss, val_acc, preds, targets, probs = self.validate()

            self.scheduler.step(val_loss)

            if val_acc > best_val_acc:
                best_val_acc = val_acc
```

```
                torch.save({
                    'epoch': epoch,
                    'model_state_dict':
self.meta_learner.state_dict(),
                    'optimizer_state_dict':
self.optimizer.state_dict(),
                    'val_acc': val_acc,
                }, 'meta_model.pth')

            history['train_loss'].append(train_loss)
            history['train_acc'].append(train_acc)
            history['val_loss'].append(val_loss)
            history['val_acc'].append(val_acc)

            print(f"Train Loss: {train_loss:.4f}, Train Acc:
{train_acc:.2f}%")
            print(f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}
%")

        print("\nConfusion Matrix (Highlighted Mismatched)")
        dv.plot_confusion_matrix(targets, preds, self.class_names)
        #plot a graph based on epoch and train_accuracy
        print("\nValidation Accuracies")
        history_df = pd.DataFrame(history)
        sns.lineplot(x=history_df.index, y='val_acc', data=history_df)
        plt.xlabel("Epoch")
        plt.ylabel("Validation Accuracy")
        plt.title("Validation Accuracy Over Epochs")
        plt.show()

        return history

    def test(self, test_loader):
        """Evaluate the model on test data and save detailed
metrics"""
        self.meta_learner.eval()
        test_loss = 0
        correct = 0
        total = 0
        all_preds = []
        all_targets = []
        all_probs = []

        with torch.no_grad():
            for inputs, targets in tqdm(test_loader, desc="Testing"):
                inputs = inputs.to(self.device)
                targets = targets.to(self.device)

                outputs = self.meta_learner(inputs)
                loss = self.criterion(outputs, targets)
```

```
                test_loss += loss.item()
                _, predicted = outputs.max(1)
                total += targets.size(0)
                correct += predicted.eq(targets).sum().item()

                all_preds.extend(predicted.cpu().numpy())
                all_targets.extend(targets.cpu().numpy())
                all_probs.extend(F.softmax(outputs,
dim=1).cpu().detach().numpy())

        # Calculate metrics
        test_loss = test_loss / len(test_loader)
        test_acc = correct / total

        from sklearn.metrics import precision_recall_fscore_support
        # Calculate additional metrics using sklearn
        test_precision, test_recall, test_f1, _ =
precision_recall_fscore_support(
            all_targets, all_preds, average='weighted'
        )
        test_bal_acc = balanced_accuracy_score(all_targets, all_preds)

        # Save detailed test results
        test_results = {
            'loss': test_loss,
            'accuracy': test_acc,
            'precision': test_precision,
            'recall': test_recall,
            'f1_score': test_f1,
            'balanced_accuracy': test_bal_acc,
            'predictions': all_preds,
            'true_labels': all_targets
        }

        # Save test results
        torch.save(test_results, 'test_results.pth')

        # Print metrics
        print(f"Test Loss: {test_loss:.6f}, Test Accuracy: {test_acc *
100:.6f}%")
        print(f"Test Precision: {test_precision:.6f}, Recall:
{test_recall:.4f}, F1 Score: {test_f1:.6f}")
        print(f"Test Balanced Accuracy: {test_bal_acc:.6f}")

        dv.plot_confusion_matrix(all_targets, all_preds,
self.class_names)

        return test_results
```

```
    def save_model(self, path='meta_model.pth'):
        """Save the trained model with all necessary information for
later use"""
        save_dict = {
            'model_state_dict': self.meta_learner.state_dict(),
            'optimizer_state_dict': self.optimizer.state_dict(),
            'model_architecture': {
                'input_size': self.meta_learner.input_size,
                'num_classes': self.meta_learner.layers[-
1].out_features
            },
            'class_names': self.class_names
        }
        torch.save(save_dict, path)
        print(f"Model saved to {path}")


def load_trained_model(model_path='ensemble_model.pth',
device='cuda'):
    """Load a trained model and return it ready for inference"""
    # Load the saved model data
    checkpoint = torch.load(model_path, map_location=device)

    # Create a new model instance with the same architecture
    model = EnhancedMetaLearner(
        input_size=checkpoint['model_architecture']['input_size'],
        num_classes=checkpoint['model_architecture']['num_classes']
    ).to(device)

    # Load the trained weights
    model.load_state_dict(checkpoint['model_state_dict'])
    model.eval()  # Set to evaluation mode

    return model, checkpoint['class_names']

GPU_unload()

# Dataset preprocessing
preprocess = models.ConvNeXt_Large_Weights.IMAGENET1K_V1.transforms()

# Load dataset
train_dataset =
datasets.ImageFolder(root="/content/BdSLW41WordDatasetSplittedAugmente
d/train", transform=preprocess)
val_dataset =
datasets.ImageFolder(root="/content/BdSLW41WordDatasetSplittedAugmente
d/val", transform=preprocess)
test_dataset =
datasets.ImageFolder(root="/content/BdSLW41WordDatasetSplittedAugmente
d/test", transform=preprocess)
```

```python
# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

print("Preparing data...")
train_loader, val_loader, test_loader, input_size = prepare_data(
    fine_models, train_loader, val_loader, test_loader
)
```

```
Preparing data...
Extracting training features...

Extracting features: 100%|██████████| 161/161 [01:04<00:00,  2.51it/s]

Extracting validation features...

Extracting features: 100%|██████████| 53/53 [00:20<00:00,  2.55it/s]

Extracting test features...

Extracting features: 100%|██████████| 31/31 [00:12<00:00,  2.51it/s]
```

```python
model_names = ['ConvNeXt', 'DenseNet', 'EfficientNet', 'RegNet',
'ResNet', 'Swin']
class_names = train_dataset.classes

meta_learner = EnhancedMetaLearner(
    input_size=input_size,
    num_classes=36
).to("cuda")

criterion = nn.CrossEntropyLoss(label_smoothing=0.01)
optimizer = optim.AdamW(
    meta_learner.parameters(),
    lr=0.0001,
    weight_decay=0.00002
)
scheduler = ReduceLROnPlateau(
    optimizer,
    mode='min',
    factor=0.25,
    patience=4
)

trainer = EnsembleTrainer(
    meta_learner=meta_learner,
    train_loader=train_loader,
    val_loader=val_loader,
    test_loader=test_loader,
```

```
    criterion=criterion,
    optimizer=optimizer,
    scheduler=scheduler,
    device="cuda",
    class_names=class_names
)

print("Starting training...")
history = trainer.train(num_epochs=100)
```

```
Starting training...

Epoch 1/100

Training: 100%|██████████| 81/81 [00:00<00:00, 230.40it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 639.06it/s]

Train Loss: 3.1613, Train Acc: 20.50%
Val Loss: 1.5728, Val Acc: 93.57%

Epoch 2/100

Training: 100%|██████████| 81/81 [00:00<00:00, 339.17it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 888.96it/s]

Train Loss: 1.5206, Train Acc: 72.41%
Val Loss: 0.6677, Val Acc: 98.15%

Epoch 3/100

Training: 100%|██████████| 81/81 [00:00<00:00, 336.39it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 695.11it/s]

Train Loss: 0.7974, Train Acc: 93.23%
Val Loss: 0.3515, Val Acc: 98.51%

Epoch 4/100

Training: 100%|██████████| 81/81 [00:00<00:00, 340.11it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 882.08it/s]

Train Loss: 0.4722, Train Acc: 98.54%
Val Loss: 0.2319, Val Acc: 98.63%

Epoch 5/100

Training: 100%|██████████| 81/81 [00:00<00:00, 334.46it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 855.11it/s]

Train Loss: 0.3308, Train Acc: 99.59%
Val Loss: 0.1930, Val Acc: 98.75%
```

```
Epoch 6/100

Training: 100%|██████████| 81/81 [00:00<00:00, 321.70it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 873.22it/s]

Train Loss: 0.2642, Train Acc: 99.80%
Val Loss: 0.1756, Val Acc: 98.75%

Epoch 7/100

Training: 100%|██████████| 81/81 [00:00<00:00, 322.93it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 889.10it/s]

Train Loss: 0.2197, Train Acc: 99.84%
Val Loss: 0.1688, Val Acc: 98.81%

Epoch 8/100

Training: 100%|██████████| 81/81 [00:00<00:00, 330.95it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 869.55it/s]

Train Loss: 0.1970, Train Acc: 99.96%
Val Loss: 0.1633, Val Acc: 98.81%

Epoch 9/100

Training: 100%|██████████| 81/81 [00:00<00:00, 330.47it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 840.84it/s]

Train Loss: 0.1782, Train Acc: 99.94%
Val Loss: 0.1600, Val Acc: 98.87%

Epoch 10/100

Training: 100%|██████████| 81/81 [00:00<00:00, 318.31it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 859.57it/s]

Train Loss: 0.1704, Train Acc: 99.92%
Val Loss: 0.1574, Val Acc: 98.81%

Epoch 11/100

Training: 100%|██████████| 81/81 [00:00<00:00, 330.63it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 879.58it/s]

Train Loss: 0.1577, Train Acc: 100.00%
Val Loss: 0.1568, Val Acc: 98.75%

Epoch 12/100

Training: 100%|██████████| 81/81 [00:00<00:00, 327.08it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 840.11it/s]
```

```
Train Loss: 0.1512, Train Acc: 99.94%
Val Loss: 0.1582, Val Acc: 98.81%

Epoch 13/100

Training: 100%|██████████| 81/81 [00:00<00:00, 334.46it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 839.64it/s]

Train Loss: 0.1462, Train Acc: 99.96%
Val Loss: 0.1567, Val Acc: 98.87%

Epoch 14/100

Training: 100%|██████████| 81/81 [00:00<00:00, 341.85it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 889.87it/s]

Train Loss: 0.1411, Train Acc: 100.00%
Val Loss: 0.1587, Val Acc: 98.75%

Epoch 15/100

Training: 100%|██████████| 81/81 [00:00<00:00, 333.33it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 856.17it/s]

Train Loss: 0.1394, Train Acc: 100.00%
Val Loss: 0.1543, Val Acc: 98.87%

Epoch 16/100

Training: 100%|██████████| 81/81 [00:00<00:00, 338.32it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 777.54it/s]

Train Loss: 0.1324, Train Acc: 100.00%
Val Loss: 0.1571, Val Acc: 98.75%

Epoch 17/100

Training: 100%|██████████| 81/81 [00:00<00:00, 325.82it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 843.31it/s]

Train Loss: 0.1363, Train Acc: 99.92%
Val Loss: 0.1580, Val Acc: 98.93%

Epoch 18/100

Training: 100%|██████████| 81/81 [00:00<00:00, 333.36it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 771.48it/s]

Train Loss: 0.1308, Train Acc: 100.00%
Val Loss: 0.1539, Val Acc: 98.87%

Epoch 19/100
```

```
Training: 100%|██████████| 81/81 [00:00<00:00, 334.21it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 836.06it/s]

Train Loss: 0.1278, Train Acc: 100.00%
Val Loss: 0.1616, Val Acc: 98.81%

Epoch 20/100

Training: 100%|██████████| 81/81 [00:00<00:00, 328.31it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 828.51it/s]

Train Loss: 0.1287, Train Acc: 100.00%
Val Loss: 0.1595, Val Acc: 98.81%

Epoch 21/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.59it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 749.03it/s]

Train Loss: 0.1268, Train Acc: 99.96%
Val Loss: 0.1577, Val Acc: 98.99%

Epoch 22/100

Training: 100%|██████████| 81/81 [00:00<00:00, 313.86it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 665.02it/s]

Train Loss: 0.1230, Train Acc: 100.00%
Val Loss: 0.1604, Val Acc: 98.81%

Epoch 23/100

Training: 100%|██████████| 81/81 [00:00<00:00, 322.86it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 702.55it/s]

Train Loss: 0.1224, Train Acc: 100.00%
Val Loss: 0.1594, Val Acc: 98.87%

Epoch 24/100

Training: 100%|██████████| 81/81 [00:00<00:00, 319.45it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 715.44it/s]

Train Loss: 0.1236, Train Acc: 99.98%
Val Loss: 0.1583, Val Acc: 98.87%

Epoch 25/100

Training: 100%|██████████| 81/81 [00:00<00:00, 311.68it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 700.46it/s]

Train Loss: 0.1211, Train Acc: 100.00%
Val Loss: 0.1564, Val Acc: 99.05%
```

```
Epoch 26/100

Training: 100%|██████████| 81/81 [00:00<00:00, 320.76it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 662.45it/s]

Train Loss: 0.1228, Train Acc: 100.00%
Val Loss: 0.1589, Val Acc: 98.75%

Epoch 27/100

Training: 100%|██████████| 81/81 [00:00<00:00, 307.89it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 741.88it/s]

Train Loss: 0.1227, Train Acc: 100.00%
Val Loss: 0.1608, Val Acc: 98.87%

Epoch 28/100

Training: 100%|██████████| 81/81 [00:00<00:00, 320.15it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 695.98it/s]

Train Loss: 0.1237, Train Acc: 99.98%
Val Loss: 0.1607, Val Acc: 98.69%

Epoch 29/100

Training: 100%|██████████| 81/81 [00:00<00:00, 316.29it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 764.05it/s]

Train Loss: 0.1193, Train Acc: 100.00%
Val Loss: 0.1593, Val Acc: 98.87%

Epoch 30/100

Training: 100%|██████████| 81/81 [00:00<00:00, 316.23it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 681.64it/s]

Train Loss: 0.1226, Train Acc: 99.98%
Val Loss: 0.1578, Val Acc: 98.99%

Epoch 31/100

Training: 100%|██████████| 81/81 [00:00<00:00, 331.29it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 761.22it/s]

Train Loss: 0.1236, Train Acc: 99.98%
Val Loss: 0.1597, Val Acc: 98.87%

Epoch 32/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.85it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 723.61it/s]
```

```
Train Loss: 0.1222, Train Acc: 99.98%
Val Loss: 0.1571, Val Acc: 98.93%

Epoch 33/100

Training: 100%|██████████| 81/81 [00:00<00:00, 319.45it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 640.36it/s]

Train Loss: 0.1211, Train Acc: 100.00%
Val Loss: 0.1598, Val Acc: 98.87%

Epoch 34/100

Training: 100%|██████████| 81/81 [00:00<00:00, 320.73it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 697.03it/s]

Train Loss: 0.1213, Train Acc: 100.00%
Val Loss: 0.1608, Val Acc: 98.81%

Epoch 35/100

Training: 100%|██████████| 81/81 [00:00<00:00, 310.58it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 681.14it/s]

Train Loss: 0.1210, Train Acc: 100.00%
Val Loss: 0.1563, Val Acc: 98.87%

Epoch 36/100

Training: 100%|██████████| 81/81 [00:00<00:00, 315.37it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 685.51it/s]

Train Loss: 0.1223, Train Acc: 100.00%
Val Loss: 0.1575, Val Acc: 98.81%

Epoch 37/100

Training: 100%|██████████| 81/81 [00:00<00:00, 311.22it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 650.68it/s]

Train Loss: 0.1240, Train Acc: 99.98%
Val Loss: 0.1588, Val Acc: 98.93%

Epoch 38/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.71it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 730.39it/s]

Train Loss: 0.1208, Train Acc: 99.98%
Val Loss: 0.1632, Val Acc: 98.69%

Epoch 39/100
```

```
Training: 100%|██████████| 81/81 [00:00<00:00, 313.24it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 717.19it/s]

Train Loss: 0.1216, Train Acc: 100.00%
Val Loss: 0.1597, Val Acc: 98.93%

Epoch 40/100

Training: 100%|██████████| 81/81 [00:00<00:00, 305.23it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 675.46it/s]

Train Loss: 0.1216, Train Acc: 99.98%
Val Loss: 0.1612, Val Acc: 98.75%

Epoch 41/100

Training: 100%|██████████| 81/81 [00:00<00:00, 298.90it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 756.23it/s]

Train Loss: 0.1215, Train Acc: 100.00%
Val Loss: 0.1620, Val Acc: 98.87%

Epoch 42/100

Training: 100%|██████████| 81/81 [00:00<00:00, 313.83it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 712.93it/s]

Train Loss: 0.1202, Train Acc: 100.00%
Val Loss: 0.1585, Val Acc: 98.87%

Epoch 43/100

Training: 100%|██████████| 81/81 [00:00<00:00, 314.21it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 704.04it/s]

Train Loss: 0.1221, Train Acc: 100.00%
Val Loss: 0.1590, Val Acc: 98.75%

Epoch 44/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.22it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 741.79it/s]

Train Loss: 0.1183, Train Acc: 100.00%
Val Loss: 0.1613, Val Acc: 98.69%

Epoch 45/100

Training: 100%|██████████| 81/81 [00:00<00:00, 316.65it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 770.73it/s]

Train Loss: 0.1204, Train Acc: 100.00%
Val Loss: 0.1588, Val Acc: 98.75%
```

```
Epoch 46/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.29it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 692.68it/s]

Train Loss: 0.1214, Train Acc: 100.00%
Val Loss: 0.1586, Val Acc: 98.87%

Epoch 47/100

Training: 100%|██████████| 81/81 [00:00<00:00, 304.29it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 719.53it/s]

Train Loss: 0.1208, Train Acc: 99.98%
Val Loss: 0.1592, Val Acc: 98.81%

Epoch 48/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.42it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 738.91it/s]

Train Loss: 0.1203, Train Acc: 100.00%
Val Loss: 0.1608, Val Acc: 98.87%

Epoch 49/100

Training: 100%|██████████| 81/81 [00:00<00:00, 314.94it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 662.73it/s]

Train Loss: 0.1205, Train Acc: 100.00%
Val Loss: 0.1579, Val Acc: 98.81%

Epoch 50/100

Training: 100%|██████████| 81/81 [00:00<00:00, 311.12it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 704.05it/s]

Train Loss: 0.1221, Train Acc: 100.00%
Val Loss: 0.1613, Val Acc: 98.81%

Epoch 51/100

Training: 100%|██████████| 81/81 [00:00<00:00, 311.87it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 773.07it/s]

Train Loss: 0.1215, Train Acc: 100.00%
Val Loss: 0.1583, Val Acc: 98.81%

Epoch 52/100

Training: 100%|██████████| 81/81 [00:00<00:00, 319.58it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 729.60it/s]
```

```
Train Loss: 0.1203, Train Acc: 100.00%
Val Loss: 0.1607, Val Acc: 98.81%

Epoch 53/100

Training: 100%|██████████| 81/81 [00:00<00:00, 314.21it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 762.15it/s]

Train Loss: 0.1211, Train Acc: 99.98%
Val Loss: 0.1593, Val Acc: 98.81%

Epoch 54/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.50it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 577.64it/s]

Train Loss: 0.1219, Train Acc: 100.00%
Val Loss: 0.1596, Val Acc: 98.81%

Epoch 55/100

Training: 100%|██████████| 81/81 [00:00<00:00, 316.39it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 735.31it/s]

Train Loss: 0.1204, Train Acc: 100.00%
Val Loss: 0.1587, Val Acc: 98.93%

Epoch 56/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.36it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 748.08it/s]

Train Loss: 0.1208, Train Acc: 100.00%
Val Loss: 0.1573, Val Acc: 98.93%

Epoch 57/100

Training: 100%|██████████| 81/81 [00:00<00:00, 301.11it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 632.97it/s]

Train Loss: 0.1223, Train Acc: 100.00%
Val Loss: 0.1588, Val Acc: 98.87%

Epoch 58/100

Training: 100%|██████████| 81/81 [00:00<00:00, 311.71it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 683.23it/s]

Train Loss: 0.1198, Train Acc: 100.00%
Val Loss: 0.1620, Val Acc: 98.81%

Epoch 59/100
```

```
Training: 100%|██████████| 81/81 [00:00<00:00, 312.18it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 733.95it/s]

Train Loss: 0.1207, Train Acc: 100.00%
Val Loss: 0.1592, Val Acc: 98.87%

Epoch 60/100

Training: 100%|██████████| 81/81 [00:00<00:00, 302.97it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 760.68it/s]

Train Loss: 0.1205, Train Acc: 100.00%
Val Loss: 0.1587, Val Acc: 98.87%

Epoch 61/100

Training: 100%|██████████| 81/81 [00:00<00:00, 320.23it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 776.75it/s]

Train Loss: 0.1217, Train Acc: 100.00%
Val Loss: 0.1590, Val Acc: 98.81%

Epoch 62/100

Training: 100%|██████████| 81/81 [00:00<00:00, 309.63it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 761.61it/s]

Train Loss: 0.1199, Train Acc: 100.00%
Val Loss: 0.1584, Val Acc: 98.87%

Epoch 63/100

Training: 100%|██████████| 81/81 [00:00<00:00, 312.65it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 633.42it/s]

Train Loss: 0.1211, Train Acc: 99.96%
Val Loss: 0.1575, Val Acc: 98.87%

Epoch 64/100

Training: 100%|██████████| 81/81 [00:00<00:00, 275.96it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 642.33it/s]

Train Loss: 0.1229, Train Acc: 100.00%
Val Loss: 0.1597, Val Acc: 98.87%

Epoch 65/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.92it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 709.09it/s]

Train Loss: 0.1204, Train Acc: 100.00%
Val Loss: 0.1609, Val Acc: 98.87%
```

```
Epoch 66/100

Training: 100%|██████████| 81/81 [00:00<00:00, 315.02it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 712.61it/s]

Train Loss: 0.1252, Train Acc: 99.98%
Val Loss: 0.1593, Val Acc: 98.81%

Epoch 67/100

Training: 100%|██████████| 81/81 [00:00<00:00, 316.65it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 729.96it/s]

Train Loss: 0.1218, Train Acc: 100.00%
Val Loss: 0.1553, Val Acc: 99.05%

Epoch 68/100

Training: 100%|██████████| 81/81 [00:00<00:00, 306.04it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 672.85it/s]

Train Loss: 0.1206, Train Acc: 100.00%
Val Loss: 0.1570, Val Acc: 98.87%

Epoch 69/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.68it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 689.44it/s]

Train Loss: 0.1206, Train Acc: 100.00%
Val Loss: 0.1585, Val Acc: 98.81%

Epoch 70/100

Training: 100%|██████████| 81/81 [00:00<00:00, 319.67it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 690.29it/s]

Train Loss: 0.1225, Train Acc: 100.00%
Val Loss: 0.1612, Val Acc: 98.81%

Epoch 71/100

Training: 100%|██████████| 81/81 [00:00<00:00, 305.75it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 744.21it/s]

Train Loss: 0.1199, Train Acc: 100.00%
Val Loss: 0.1614, Val Acc: 98.75%

Epoch 72/100

Training: 100%|██████████| 81/81 [00:00<00:00, 307.30it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 793.41it/s]
```

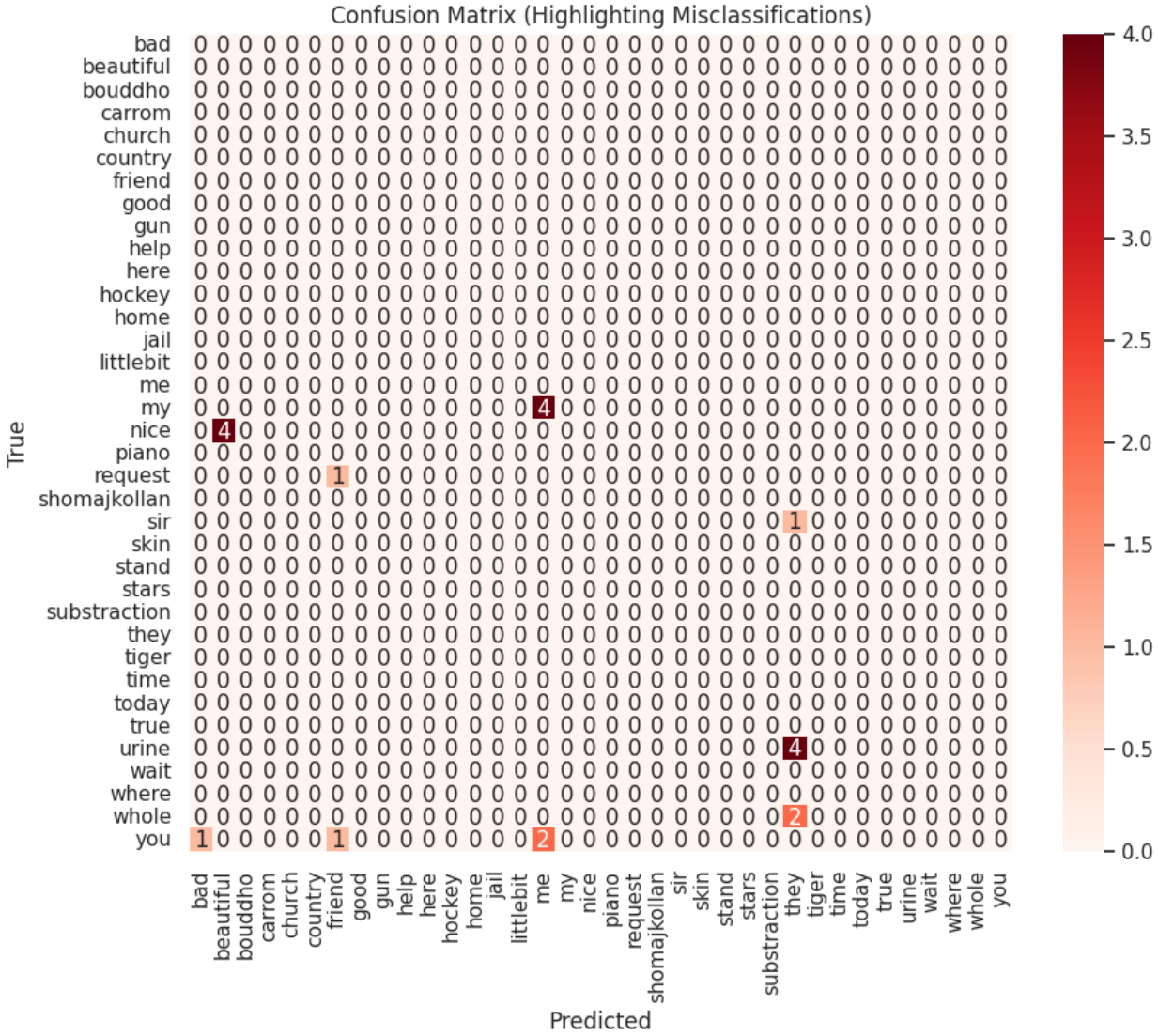
```
Train Loss: 0.1205, Train Acc: 100.00%
Val Loss: 0.1574, Val Acc: 98.93%

Epoch 73/100

Training: 100%|██████████| 81/81 [00:00<00:00, 311.24it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 678.33it/s]

Train Loss: 0.1210, Train Acc: 100.00%
Val Loss: 0.1578, Val Acc: 98.81%

Epoch 74/100

Training: 100%|██████████| 81/81 [00:00<00:00, 302.58it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 747.98it/s]

Train Loss: 0.1220, Train Acc: 100.00%
Val Loss: 0.1573, Val Acc: 98.93%

Epoch 75/100

Training: 100%|██████████| 81/81 [00:00<00:00, 307.91it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 667.49it/s]

Train Loss: 0.1210, Train Acc: 100.00%
Val Loss: 0.1616, Val Acc: 98.87%

Epoch 76/100

Training: 100%|██████████| 81/81 [00:00<00:00, 306.06it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 755.83it/s]

Train Loss: 0.1224, Train Acc: 99.98%
Val Loss: 0.1583, Val Acc: 98.87%

Epoch 77/100

Training: 100%|██████████| 81/81 [00:00<00:00, 309.45it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 668.54it/s]

Train Loss: 0.1212, Train Acc: 100.00%
Val Loss: 0.1635, Val Acc: 98.69%

Epoch 78/100

Training: 100%|██████████| 81/81 [00:00<00:00, 304.57it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 754.36it/s]

Train Loss: 0.1212, Train Acc: 100.00%
Val Loss: 0.1604, Val Acc: 98.93%

Epoch 79/100
```

```
Training: 100%|██████████| 81/81 [00:00<00:00, 309.99it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 693.64it/s]

Train Loss: 0.1225, Train Acc: 99.98%
Val Loss: 0.1584, Val Acc: 98.93%

Epoch 80/100

Training: 100%|██████████| 81/81 [00:00<00:00, 301.11it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 587.02it/s]

Train Loss: 0.1210, Train Acc: 100.00%
Val Loss: 0.1593, Val Acc: 98.87%

Epoch 81/100

Training: 100%|██████████| 81/81 [00:00<00:00, 304.61it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 589.46it/s]

Train Loss: 0.1188, Train Acc: 100.00%
Val Loss: 0.1580, Val Acc: 98.87%

Epoch 82/100

Training: 100%|██████████| 81/81 [00:00<00:00, 314.66it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 637.55it/s]

Train Loss: 0.1199, Train Acc: 100.00%
Val Loss: 0.1627, Val Acc: 98.75%

Epoch 83/100

Training: 100%|██████████| 81/81 [00:00<00:00, 314.27it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 619.93it/s]

Train Loss: 0.1217, Train Acc: 100.00%
Val Loss: 0.1616, Val Acc: 98.81%

Epoch 84/100

Training: 100%|██████████| 81/81 [00:00<00:00, 314.47it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 663.67it/s]

Train Loss: 0.1224, Train Acc: 99.98%
Val Loss: 0.1600, Val Acc: 98.81%

Epoch 85/100

Training: 100%|██████████| 81/81 [00:00<00:00, 306.72it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 626.58it/s]

Train Loss: 0.1205, Train Acc: 100.00%
Val Loss: 0.1615, Val Acc: 98.87%
```

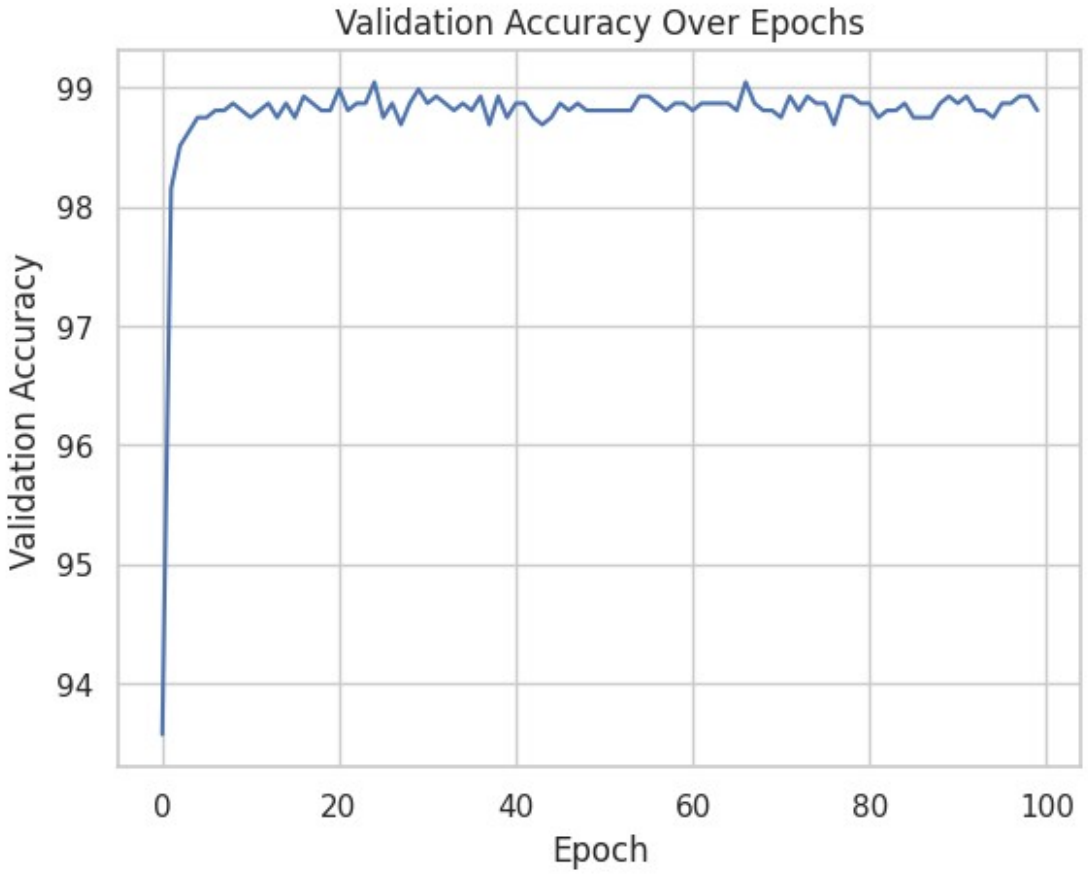
```
Epoch 86/100

Training: 100%|██████████| 81/81 [00:00<00:00, 292.50it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 755.57it/s]

Train Loss: 0.1210, Train Acc: 100.00%
Val Loss: 0.1612, Val Acc: 98.75%

Epoch 87/100

Training: 100%|██████████| 81/81 [00:00<00:00, 298.34it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 745.35it/s]

Train Loss: 0.1207, Train Acc: 100.00%
Val Loss: 0.1609, Val Acc: 98.75%

Epoch 88/100

Training: 100%|██████████| 81/81 [00:00<00:00, 309.04it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 760.57it/s]

Train Loss: 0.1207, Train Acc: 100.00%
Val Loss: 0.1608, Val Acc: 98.75%

Epoch 89/100

Training: 100%|██████████| 81/81 [00:00<00:00, 309.58it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 796.07it/s]

Train Loss: 0.1213, Train Acc: 99.96%
Val Loss: 0.1610, Val Acc: 98.87%

Epoch 90/100

Training: 100%|██████████| 81/81 [00:00<00:00, 312.73it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 768.73it/s]

Train Loss: 0.1193, Train Acc: 99.98%
Val Loss: 0.1585, Val Acc: 98.93%

Epoch 91/100

Training: 100%|██████████| 81/81 [00:00<00:00, 310.48it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 678.81it/s]

Train Loss: 0.1199, Train Acc: 100.00%
Val Loss: 0.1596, Val Acc: 98.87%

Epoch 92/100

Training: 100%|██████████| 81/81 [00:00<00:00, 318.50it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 661.37it/s]
```

```
Train Loss: 0.1225, Train Acc: 100.00%
Val Loss: 0.1588, Val Acc: 98.93%

Epoch 93/100

Training: 100%|██████████| 81/81 [00:00<00:00, 300.45it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 685.35it/s]

Train Loss: 0.1206, Train Acc: 100.00%
Val Loss: 0.1582, Val Acc: 98.81%

Epoch 94/100

Training: 100%|██████████| 81/81 [00:00<00:00, 315.78it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 587.73it/s]

Train Loss: 0.1215, Train Acc: 100.00%
Val Loss: 0.1579, Val Acc: 98.81%

Epoch 95/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.51it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 756.27it/s]

Train Loss: 0.1197, Train Acc: 100.00%
Val Loss: 0.1573, Val Acc: 98.75%

Epoch 96/100

Training: 100%|██████████| 81/81 [00:00<00:00, 301.99it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 783.88it/s]

Train Loss: 0.1202, Train Acc: 100.00%
Val Loss: 0.1568, Val Acc: 98.87%

Epoch 97/100

Training: 100%|██████████| 81/81 [00:00<00:00, 308.67it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 635.11it/s]

Train Loss: 0.1221, Train Acc: 100.00%
Val Loss: 0.1608, Val Acc: 98.87%

Epoch 98/100

Training: 100%|██████████| 81/81 [00:00<00:00, 318.18it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 704.42it/s]

Train Loss: 0.1200, Train Acc: 100.00%
Val Loss: 0.1565, Val Acc: 98.93%

Epoch 99/100
```

```
Training: 100%|██████████| 81/81 [00:00<00:00, 307.08it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 683.27it/s]

Train Loss: 0.1209, Train Acc: 100.00%
Val Loss: 0.1548, Val Acc: 98.93%

Epoch 100/100

Training: 100%|██████████| 81/81 [00:00<00:00, 307.61it/s]
Validating: 100%|██████████| 27/27 [00:00<00:00, 683.04it/s]

Train Loss: 0.1228, Train Acc: 100.00%
Val Loss: 0.1585, Val Acc: 98.81%

Confusion Matrix (Highlighted Mismatched)
```



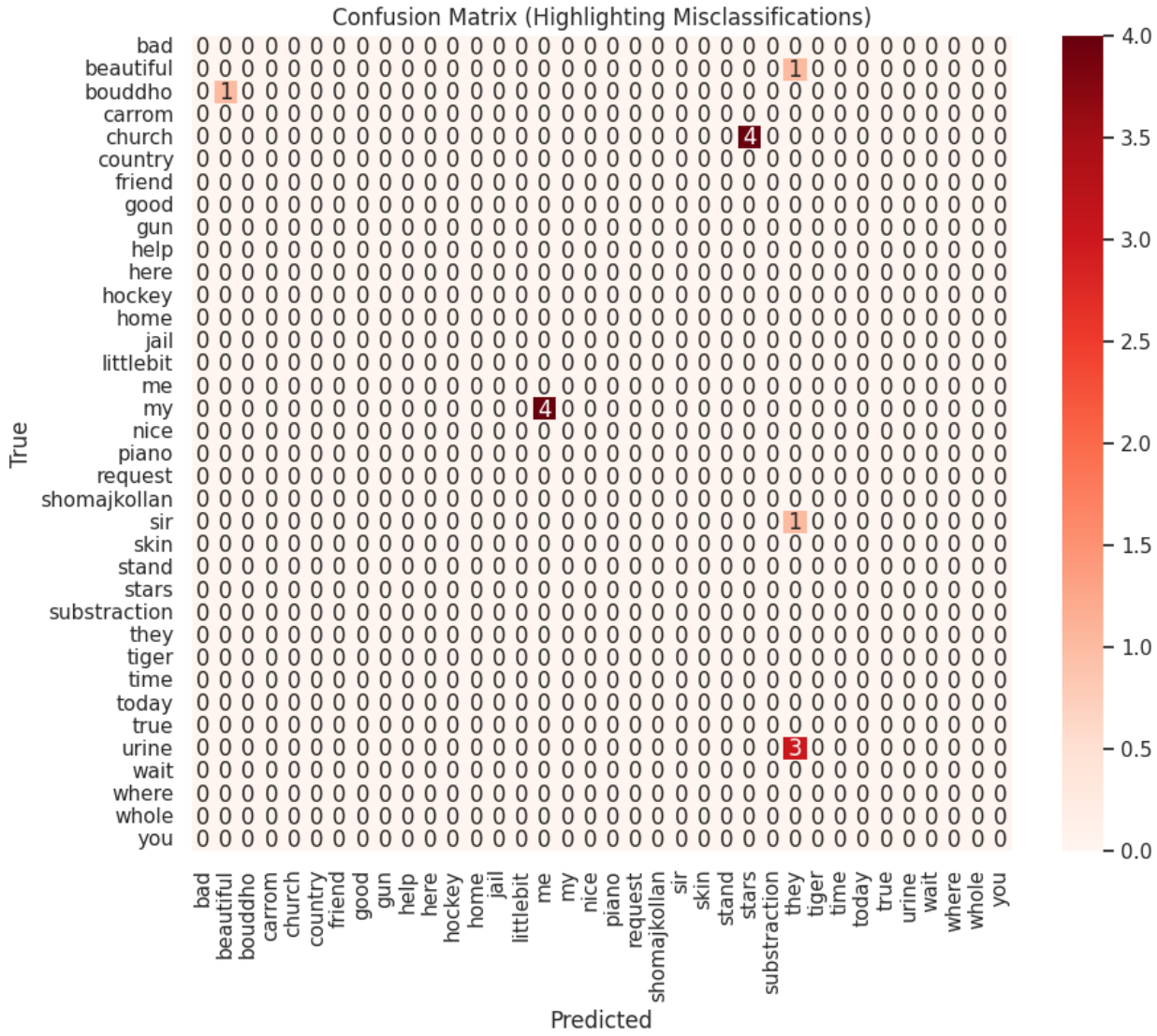
```
Validation Accuracies
```




```python
def extract_features(models, dataloader):
    features = []
    labels = []

    with torch.no_grad():
        for inputs, targets in tqdm(dataloader, desc="Extracting
features"):
            inputs = inputs.to("cuda") # Assuming your device is
'cuda'
            batch_features = []

            for model in models:
                outputs = model(inputs)
                batch_features.append(outputs.cpu())

            # Concatenate features from all models
            combined_features = torch.cat(batch_features, dim=1)
            features.append(combined_features)
            labels.append(targets)
```

```
    return torch.cat(features, dim=0), torch.cat(labels, dim=0)

print("Preparing test data...")

trainer.save_model()

print("\nEvaluating on test set...")
test_results = trainer.test(test_loader)
```

```
Preparing test data...
Model saved to meta_model.pth

Evaluating on test set...

Testing: 100%|██████████| 16/16 [00:00<00:00, 554.35it/s]

Test Loss: 0.163773, Test Accuracy: 98.588710%
Test Precision: 0.987473, Recall: 0.9859, F1 Score: 0.985828
Test Balanced Accuracy: 0.984921
```

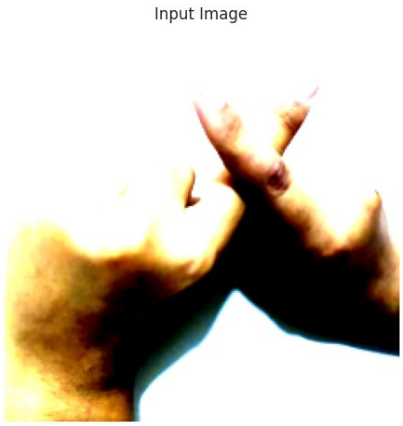


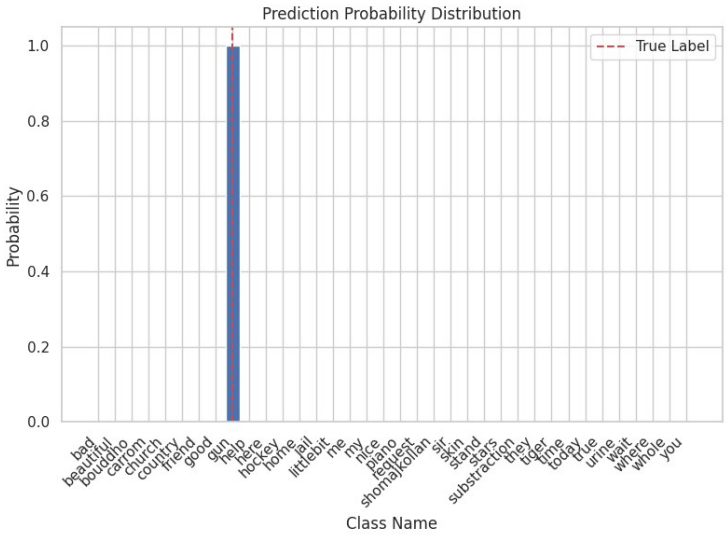

```
def predict_image(image_path, model, class_names, fine_models,
device='cuda'):
    transform =
models.ConvNeXt_Large_Weights.IMAGENET1K_V1.transforms()
    image = Image.open(image_path)
    image = transform(image).unsqueeze(0).to(device)

    # Extract features using fine_models
    with torch.no_grad():
        fine_features = []
        for fine_model in fine_models:
            fine_model.eval()
            fine_output = fine_model(image)
            fine_features.append(fine_output)

        # Concatenate features from all fine models
```

```
        fine_features = torch.cat(fine_features, dim=1).to(device)

    with torch.no_grad():
        meta_output = model(fine_features)

    _, pred_class = torch.max(meta_output, 1)
    confidence = torch.softmax(meta_output, dim=1)[0]
[pred_class].item()

    # Visualize image and prediction
    plt.figure(figsize=(15, 6))  # Create one figure for both plots

    # Move the tensor to the CPU and convert it to a NumPy array
    image_np = image.cpu().numpy()
    # Squeeze the array to remove the extra dimension
    image_np = np.squeeze(image_np, axis=0)
    image_np = image_np.transpose(1, 2, 0)
    # Plot the input image
    plt.subplot(1, 2, 1)
    plt.imshow(image_np)
    plt.axis('off')
    plt.title('Input Image')


    # Plot the prediction probabilities
    plt.subplot(1, 2, 2)
    probs = F.softmax(meta_output, dim=1)[0].cpu().numpy()
    plt.bar(class_names, probs)
    plt.xticks(rotation=45, ha='right')
    plt.xlabel('Class Name')
    plt.ylabel('Probability')
    plt.title('Prediction Probability Distribution')
    plt.tight_layout()

    # Add a red line for the predicted class
    predicted_class_name =
class_names[torch.argmax(F.softmax(meta_output, dim=1)).item()]
    plt.axvline(x=predicted_class_name, color='r', linestyle='--',
label='True Label')
    plt.legend()

    plt.show()

    # Print the top K predictions
    top_k = 5
    top_probs, top_indices = torch.topk(F.softmax(meta_output, dim=1)
[0], k=top_k)
    print("\nTop {} Predictions:".format(top_k))
    for i in range(top_k):
        print(f"{class_names[top_indices[i]]}:
```

```
{top_probs[i].item()*100:.2f}%")

    return class_names[pred_class], confidence

GPU_unload()

# Example of loading and using the saved model
loaded_model, class_names = load_trained_model('meta_model.pth')

# Example prediction
image_path =
'/content/BdSLW41WordDatasetSplittedAugmented/test2/gun/gun_2_aug_3.jp
g'
predicted_class, confidence = predict_image(
    image_path,
    loaded_model,
    class_names,
    fine_models
)
print(f"Predicted class: {predicted_class}")
print(f"Confidence: {confidence:.2%}")
```

```
<ipython-input-36-e08806f22bff>:196: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  checkpoint = torch.load(model_path, map_location=device)
WARNING:matplotlib.image:Clipping input data to the valid range for
imshow with RGB data ([0..1] for floats or [0..255] for integers).
```






```
Top 5 Predictions:
gun: 99.89%
true: 0.01%
good: 0.01%
sir: 0.01%
here: 0.01%
Predicted class: gun
Confidence: 99.89%

GPU_unload()

load_architectures()

fine_models = [convnext, densenet, efficientnet, regnet, resnet, swin]
for fine_model in fine_models:
    fine_model.eval()

# Freeze models
for model in fine_models:
    for param in model.parameters():
        param.requires_grad = False

/usr/local/lib/python3.10/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
```

```
<ipython-input-21-cd410b975fbd>:7: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/ConvNeXt_{model_best_training_epoch[0]}.pth
")
<ipython-input-21-cd410b975fbd>:12: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/DenseNet_{model_best_training_epoch[1]}.pth
")
<ipython-input-21-cd410b975fbd>:17: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
```

```
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/EfficientNet_{model_best_training_epoch[2]}
.pth")
<ipython-input-21-cd410b975fbd>:22: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/RegNet_{model_best_training_epoch[3]}.pth")
<ipython-input-21-cd410b975fbd>:27: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/ResNet_{model_best_training_epoch[4]}.pth")
<ipython-input-21-cd410b975fbd>:31: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
```

```
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/SwinTransformer_{model_best_training_epoch[
5]}.pth")

import torch
import torch.nn as nn
import torchvision
from torch.nn import TransformerEncoder, TransformerEncoderLayer

class TransformerFusionModel(nn.Module):
    def __init__(self, models, num_classes=36, embedding_dim=256,
nhead=4, num_layers=2):
        super(TransformerFusionModel, self).__init__()
        self.models = nn.ModuleList(models)
        self.embedding_dim = embedding_dim

        # Projection layers to map model outputs to the same embedding
dimension
        self.projections = nn.ModuleList([nn.Linear(num_classes,
embedding_dim) for _ in models])

        # Transformer Encoder
        encoder_layer = TransformerEncoderLayer(d_model=embedding_dim,
nhead=nhead)
        self.transformer_encoder = TransformerEncoder(encoder_layer,
num_layers=num_layers)

        # Fully connected layer for final classification
        self.fc = nn.Linear(embedding_dim, num_classes)
        self.dropout = nn.Dropout(0.35)  # Add dropout for
regularization

    def forward(self, x):
        batch_embeddings = []

        # Pass input through each model and collect embeddings
        for model, projection in zip(self.models, self.projections):
            with torch.no_grad():
                outputs = model(x)  # Shape: (batch_size, num_classes)
            embeddings = projection(outputs)  # Shape: (batch_size,
embedding_dim)
            embeddings = self.dropout(embeddings)  # Apply dropout
            batch_embeddings.append(embeddings.unsqueeze(1))  # Add
sequence dimension
```

```
        # Concatenate embeddings as input tokens for the transformer
        tokens = torch.cat(batch_embeddings, dim=1)  # Shape:
(batch_size, num_models, embedding_dim)

        # Pass tokens through the transformer encoder
        transformer_output =
self.transformer_encoder(tokens.permute(1, 0, 2))  # Permute to
(seq_len, batch_size, embedding_dim)
        transformer_embedding = transformer_output[0]  # Use the first
token (CLS token equivalent)

        # Final classification
        logits = self.fc(transformer_embedding)
        return logits

from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch.optim as optim
from torch.optim.lr_scheduler import ReduceLROnPlateau

# Dataset preparation
preprocess = models.ConvNeXt_Large_Weights.IMAGENET1K_V1.transforms()

train_dataset =
datasets.ImageFolder(root="/content/BdSLW41WordDatasetSplittedAugmente
d/train", transform=preprocess)
val_dataset =
datasets.ImageFolder(root="/content/BdSLW41WordDatasetSplittedAugmente
d/val", transform=preprocess)
test_dataset =
datasets.ImageFolder(root="/content/BdSLW41WordDatasetSplittedAugmente
d/test", transform=preprocess)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True,
num_workers=12, pin_memory=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False,
num_workers=12, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False,
num_workers=12, pin_memory=True)

# Initialize the model
num_classes = 36
fusion_model = TransformerFusionModel(models=fine_models,
num_classes=num_classes).to("cuda")

# Loss and optimizer
criterion = nn.CrossEntropyLoss(label_smoothing=0.01)
optimizer = optim.AdamW(fusion_model.parameters(), lr=0.001,
weight_decay=0.00001)
```

```
scheduler = ReduceLROnPlateau(optimizer, mode='min', factor=0.7,
patience=3)
```

```
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/
transformer.py:379: UserWarning: enable_nested_tensor is True, but
self.use_nested_tensor is False because
encoder_layer.self_attn.batch_first was not True(use batch_first for
better inference performance)
  warnings.warn(
```

```
from tqdm import tqdm
from sklearn.metrics import precision_score, recall_score, f1_score,
roc_auc_score, balanced_accuracy_score, accuracy_score

# Training loop
num_epochs = 10
all_val_accuracies = []
best_val_acc=0
for epoch in range(num_epochs):
    fusion_model.train()
    train_loss = 0
    correct = 0
    total = 0
    train_targets = []
    train_preds = []

    # Progress bar for training
    with tqdm(total=len(train_loader), desc=f"Epoch
{epoch+1}/{num_epochs}") as pbar:
        for inputs, targets in train_loader:
            inputs, targets = inputs.to("cuda"), targets.to("cuda")

            # Forward pass
            outputs = fusion_model(inputs)
            loss = criterion(outputs, targets)

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Track metrics
            train_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += (predicted == targets).sum().item()

            train_targets.extend(targets.cpu().numpy())
            train_preds.extend(predicted.cpu().numpy())
```

```
            pbar.set_postfix(Loss=f"{train_loss/(total/32):.4f}",
Accuracy=f"{(100*correct/total):.2f}%")
            pbar.update(1)

    # Compute additional training metrics
    train_acc = accuracy_score(train_targets, train_preds)
    train_precision = precision_score(train_targets, train_preds,
average='macro')
    train_recall = recall_score(train_targets, train_preds,
average='macro')
    train_f1 = f1_score(train_targets, train_preds, average='macro')
    train_bal_acc = balanced_accuracy_score(train_targets,
train_preds)

    print(f"Epoch {epoch + 1}/{num_epochs}, Training Loss:
{train_loss:.4f}, Accuracy: {train_acc * 100:.2f}%")
    print(f"Training Precision: {train_precision:.4f}, Recall:
{train_recall:.4f}, F1 Score: {train_f1:.4f}")
    print(f"Training Balanced Accuracy: {train_bal_acc:.4f}")

    # Validation loop
    fusion_model.eval()
    val_loss = 0
    correct = 0
    total = 0
    val_targets = []
    val_preds = []

    with torch.no_grad():
        for inputs, targets in val_loader:
            inputs, targets = inputs.to("cuda"), targets.to("cuda")
            outputs = fusion_model(inputs)
            loss = criterion(outputs, targets)
            val_loss += loss.item()

            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += (predicted == targets).sum().item()

            val_targets.extend(targets.cpu().numpy())
            val_preds.extend(predicted.cpu().numpy())

    # Compute additional validation metrics
    val_acc = accuracy_score(val_targets, val_preds)
    all_val_accuracies.append(val_acc)
    val_precision = precision_score(val_targets, val_preds,
average='macro')
    val_recall = recall_score(val_targets, val_preds, average='macro')
    val_f1 = f1_score(val_targets, val_preds, average='macro')
    val_bal_acc = balanced_accuracy_score(val_targets, val_preds)
```

```
    if val_acc>best_val_acc:
        best_val_acc=val_acc
        torch.save({
            'epoch': epoch,
            'model_state_dict': fusion_model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': criterion,
            },
            '/content/fusion_model.pth')

    print(f"Validation Loss: {val_loss:.4f}, Accuracy: {val_acc *
100:.2f}%")
    print(f"Validation Precision: {val_precision:.4f}, Recall:
{val_recall:.4f}, F1 Score: {val_f1:.4f}")
    print(f"Validation Balanced Accuracy: {val_bal_acc:.4f}")

print("\nConfusion Matrix (Highlighted Mismatched)")
dv.plot_confusion_matrix(val_targets, val_preds,
train_dataset.classes)

print("\nValidation Accuracies")
history = []
for epoch, val_acc in enumerate(all_val_accuracies):
    history.append({"epoch": epoch + 1, "val_acc": val_acc})  # Create
a dictionary for each epoch

history_df = pd.DataFrame(history)
sns.lineplot(x='epoch', y='val_acc', data=history_df)  # Use 'epoch'
for x-axis
plt.xlabel("Epoch")
plt.ylabel("Validation Accuracy")
plt.title("Validation Accuracy Over Epochs")
plt.show()

Epoch 1/20: 100%|██████████| 161/161 [00:48<00:00,  3.30it/s,
Accuracy=97.72%, Loss=0.1986]

Epoch 1/20, Training Loss: 31.8264, Accuracy: 97.72%
Training Precision: 0.9776, Recall: 0.9722, F1 Score: 0.9747
Training Balanced Accuracy: 0.9722



Validation Loss: 6.0802, Accuracy: 99.52%
Validation Precision: 0.9963, Recall: 0.9961, F1 Score: 0.9961
Validation Balanced Accuracy: 0.9961

Epoch 2/20: 100%|██████████| 161/161 [00:47<00:00,  3.41it/s,
Accuracy=100.00%, Loss=0.0924]
```

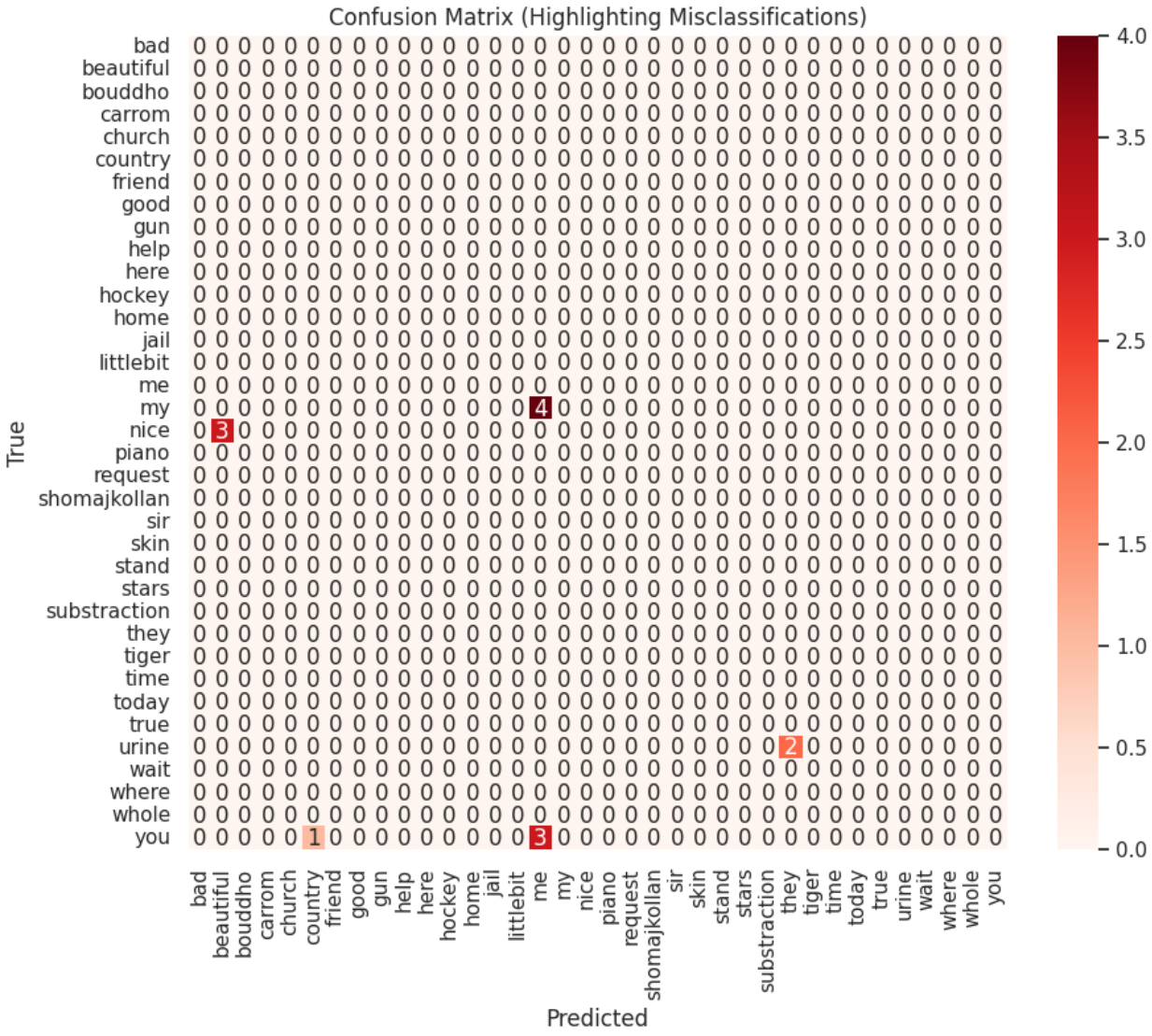
```
Epoch 2/20, Training Loss: 14.7992, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000



Validation Loss: 6.7490, Accuracy: 99.29%
Validation Precision: 0.9956, Recall: 0.9921, F1 Score: 0.9935
Validation Balanced Accuracy: 0.9921

Epoch 3/20: 100%|██████████| 161/161 [00:46<00:00,  3.43it/s,
Accuracy=99.96%, Loss=0.0927]

Epoch 3/20, Training Loss: 14.8514, Accuracy: 99.96%
Training Precision: 0.9996, Recall: 0.9996, F1 Score: 0.9996
Training Balanced Accuracy: 0.9996
Validation Loss: 6.0628, Accuracy: 99.52%
Validation Precision: 0.9966, Recall: 0.9956, F1 Score: 0.9960
Validation Balanced Accuracy: 0.9956

Epoch 4/20: 100%|██████████| 161/161 [00:49<00:00,  3.25it/s,
Accuracy=99.47%, Loss=0.1156]

Epoch 4/20, Training Loss: 18.5263, Accuracy: 99.47%
Training Precision: 0.9927, Recall: 0.9920, F1 Score: 0.9924
Training Balanced Accuracy: 0.9920



Validation Loss: 7.5690, Accuracy: 99.35%
Validation Precision: 0.9944, Recall: 0.9918, F1 Score: 0.9928
Validation Balanced Accuracy: 0.9918

Epoch 5/20: 100%|██████████| 161/161 [00:49<00:00,  3.27it/s,
Accuracy=100.00%, Loss=0.0944]

Epoch 5/20, Training Loss: 15.1225, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000
Validation Loss: 7.1371, Accuracy: 99.46%
Validation Precision: 0.9952, Recall: 0.9936, F1 Score: 0.9941
Validation Balanced Accuracy: 0.9936

Epoch 6/20: 100%|██████████| 161/161 [00:47<00:00,  3.37it/s,
Accuracy=100.00%, Loss=0.0914]

Epoch 6/20, Training Loss: 14.6527, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000
Validation Loss: 8.4107, Accuracy: 98.99%
Validation Precision: 0.9928, Recall: 0.9867, F1 Score: 0.9889
Validation Balanced Accuracy: 0.9867
```

```
Epoch 7/20: 100%|██████████| 161/161 [00:47<00:00,  3.40it/s,
Accuracy=99.94%, Loss=0.0946]

Epoch 7/20, Training Loss: 15.1603, Accuracy: 99.94%
Training Precision: 0.9993, Recall: 0.9995, F1 Score: 0.9994
Training Balanced Accuracy: 0.9995



Validation Loss: 6.9051, Accuracy: 99.35%
Validation Precision: 0.9941, Recall: 0.9920, F1 Score: 0.9928
Validation Balanced Accuracy: 0.9920

Epoch 8/20: 100%|██████████| 161/161 [00:50<00:00,  3.22it/s,
Accuracy=100.00%, Loss=0.0908]

Epoch 8/20, Training Loss: 14.5500, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000



Validation Loss: 6.5747, Accuracy: 99.46%
Validation Precision: 0.9958, Recall: 0.9939, F1 Score: 0.9947
Validation Balanced Accuracy: 0.9939

Epoch 9/20: 100%|██████████| 161/161 [00:49<00:00,  3.27it/s,
Accuracy=100.00%, Loss=0.0908]

Epoch 9/20, Training Loss: 14.5477, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000



Validation Loss: 6.9323, Accuracy: 99.46%
Validation Precision: 0.9958, Recall: 0.9939, F1 Score: 0.9947
Validation Balanced Accuracy: 0.9939

Epoch 10/20: 100%|██████████| 161/161 [00:47<00:00,  3.40it/s,
Accuracy=100.00%, Loss=0.0905]

Epoch 10/20, Training Loss: 14.4993, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000



Validation Loss: 6.9313, Accuracy: 99.35%
Validation Precision: 0.9953, Recall: 0.9921, F1 Score: 0.9934
Validation Balanced Accuracy: 0.9921
```

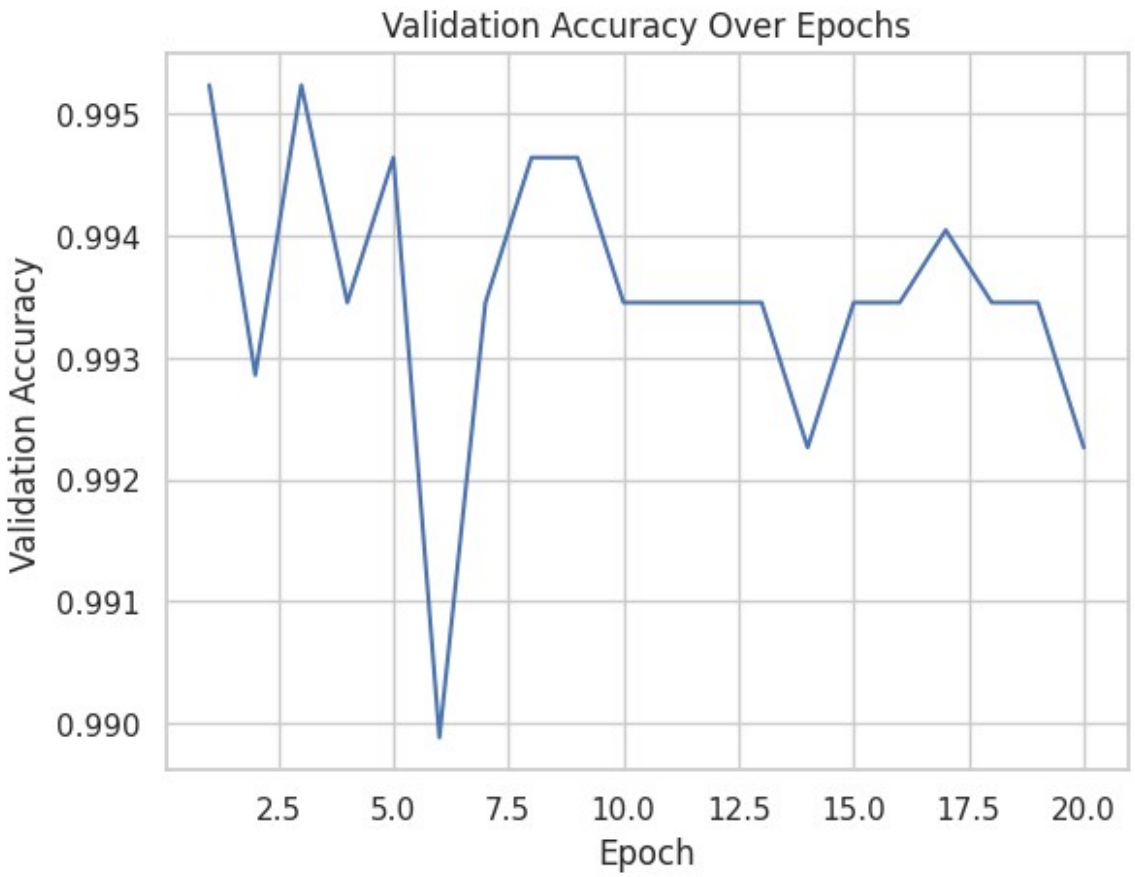
```
Epoch 11/20: 100%|██████████| 161/161 [00:49<00:00,  3.25it/s,
Accuracy=100.00%, Loss=0.0903]

Epoch 11/20, Training Loss: 14.4705, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000
Validation Loss: 6.9176, Accuracy: 99.35%
Validation Precision: 0.9962, Recall: 0.9921, F1 Score: 0.9939
Validation Balanced Accuracy: 0.9921

Epoch 12/20: 100%|██████████| 161/161 [00:49<00:00,  3.24it/s,
Accuracy=100.00%, Loss=0.0902]

Epoch 12/20, Training Loss: 14.4596, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000



Validation Loss: 6.9908, Accuracy: 99.35%
Validation Precision: 0.9953, Recall: 0.9921, F1 Score: 0.9934
Validation Balanced Accuracy: 0.9921

Epoch 13/20: 100%|██████████| 161/161 [00:49<00:00,  3.25it/s,
Accuracy=100.00%, Loss=0.0902]

Epoch 13/20, Training Loss: 14.4522, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000



Validation Loss: 7.1627, Accuracy: 99.35%
Validation Precision: 0.9962, Recall: 0.9921, F1 Score: 0.9939
Validation Balanced Accuracy: 0.9921

Epoch 14/20: 100%|██████████| 161/161 [00:49<00:00,  3.24it/s,
Accuracy=100.00%, Loss=0.0902]

Epoch 14/20, Training Loss: 14.4471, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000
Validation Loss: 7.4317, Accuracy: 99.23%
Validation Precision: 0.9945, Recall: 0.9901, F1 Score: 0.9919
Validation Balanced Accuracy: 0.9901

Epoch 15/20: 100%|██████████| 161/161 [00:49<00:00,  3.25it/s,
Accuracy=100.00%, Loss=0.0901]

Epoch 15/20, Training Loss: 14.4413, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000
```

```
Validation Loss: 7.0477, Accuracy: 99.35%
Validation Precision: 0.9962, Recall: 0.9921, F1 Score: 0.9939
Validation Balanced Accuracy: 0.9921

Epoch 16/20: 100%|██████████| 161/161 [00:49<00:00,  3.24it/s,
Accuracy=100.00%, Loss=0.0902]

Epoch 16/20, Training Loss: 14.4583, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000
Validation Loss: 6.9406, Accuracy: 99.35%
Validation Precision: 0.9957, Recall: 0.9921, F1 Score: 0.9936
Validation Balanced Accuracy: 0.9921

Epoch 17/20: 100%|██████████| 161/161 [00:49<00:00,  3.24it/s,
Accuracy=100.00%, Loss=0.0902]

Epoch 17/20, Training Loss: 14.4511, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000
Validation Loss: 6.6793, Accuracy: 99.40%
Validation Precision: 0.9965, Recall: 0.9931, F1 Score: 0.9946
Validation Balanced Accuracy: 0.9931

Epoch 18/20: 100%|██████████| 161/161 [00:49<00:00,  3.24it/s,
Accuracy=100.00%, Loss=0.0901]

Epoch 18/20, Training Loss: 14.4437, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000
Validation Loss: 6.7958, Accuracy: 99.35%
Validation Precision: 0.9957, Recall: 0.9921, F1 Score: 0.9936
Validation Balanced Accuracy: 0.9921

Epoch 19/20: 100%|██████████| 161/161 [00:49<00:00,  3.24it/s,
Accuracy=100.00%, Loss=0.0901]

Epoch 19/20, Training Loss: 14.4319, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000
Validation Loss: 7.2472, Accuracy: 99.35%
Validation Precision: 0.9950, Recall: 0.9919, F1 Score: 0.9931
Validation Balanced Accuracy: 0.9919

Epoch 20/20: 100%|██████████| 161/161 [00:49<00:00,  3.23it/s,
Accuracy=100.00%, Loss=0.0900]

Epoch 20/20, Training Loss: 14.4275, Accuracy: 100.00%
Training Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
Training Balanced Accuracy: 1.0000
Validation Loss: 7.4602, Accuracy: 99.23%
```

```
Validation Precision: 0.9949, Recall: 0.9907, F1 Score: 0.9925
Validation Balanced Accuracy: 0.9907

Confusion Matrix (Highlighted Mismatched)
```



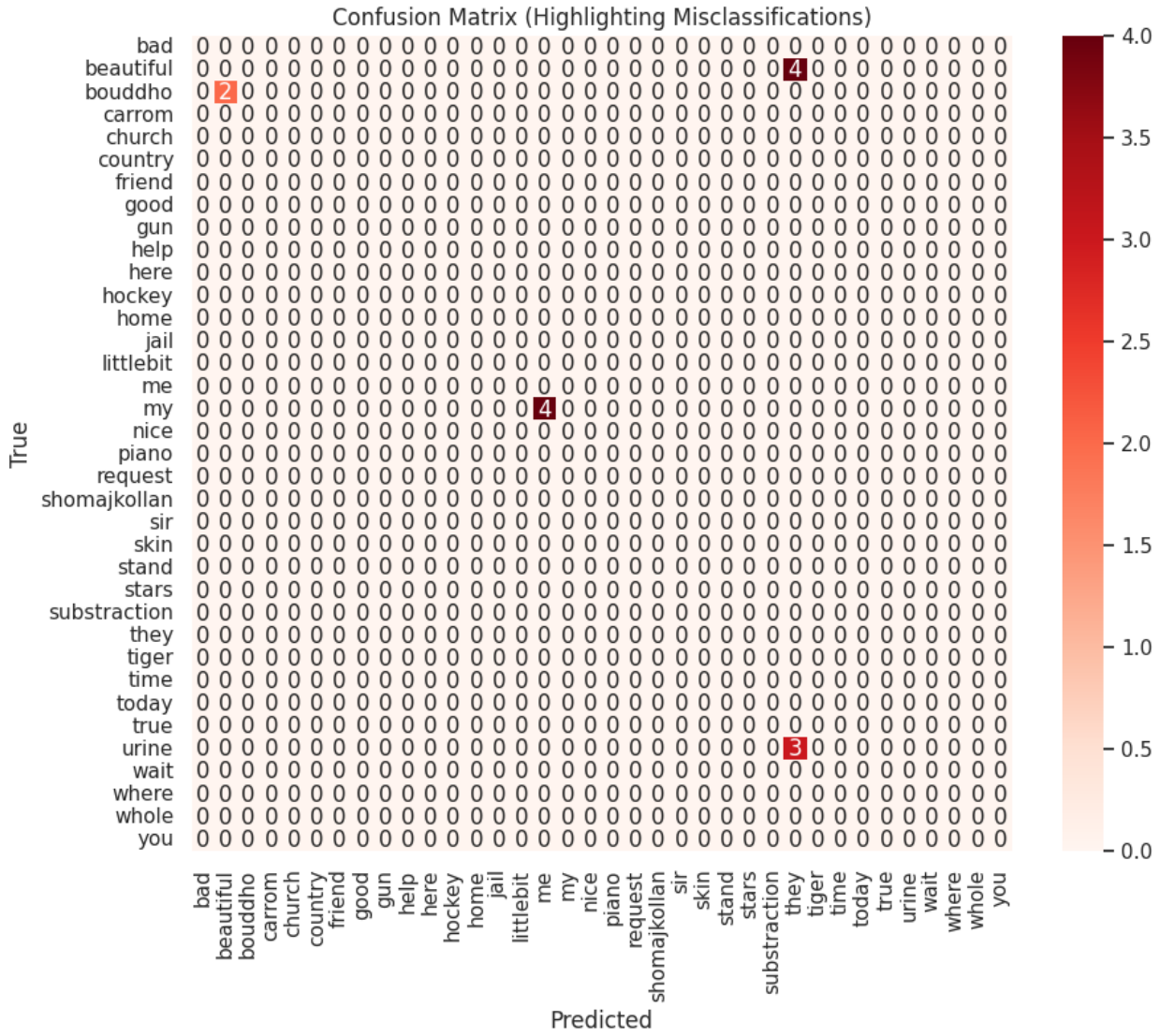

```
Validation Accuracies
```




```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, balanced_accuracy_score, roc_auc_score

# Test evaluation
fusion_model.eval()
test_loss = 0
correct = 0
total = 0
test_targets = []
test_preds = []

with torch.no_grad():
    with tqdm(total=len(test_loader), desc="Testing", unit="epoch") as
pbar:
        for inputs, targets in test_loader:
            inputs, targets = inputs.to("cuda"), targets.to("cuda")

            # Forward pass
            outputs = fusion_model(inputs)
            loss = criterion(outputs, targets)
            test_loss += loss.item()

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

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

            test_targets.extend(targets.cpu().numpy())
            test_preds.extend(predicted.cpu().numpy())

            pbar.set_postfix(Loss=f"{test_loss/(total/32):.4f}",
Accuracy=f"{(100*correct/total):.2f}%")
            pbar.update(1)

# Calculate overall metrics
test_acc = accuracy_score(test_targets, test_preds)
test_precision = precision_score(test_targets, test_preds,
average='macro')
test_recall = recall_score(test_targets, test_preds, average='macro')
test_f1 = f1_score(test_targets, test_preds, average='macro')
test_bal_acc = balanced_accuracy_score(test_targets, test_preds)

# Print results
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_acc *
100:.2f}%")
print(f"Test Precision: {test_precision:.4f}, Recall:
{test_recall:.4f}, F1 Score: {test_f1:.4f}")
print(f"Test Balanced Accuracy: {test_bal_acc:.4f}")

# Plot confusion matrix
print("\nConfusion Matrix (Highlighted Mismatched)")
dv.plot_confusion_matrix(test_targets, test_preds,
train_dataset.classes)
```

```
Testing: 100%|██████████| 31/31 [00:09<00:00,  3.33epoch/s,
Accuracy=98.69%, Loss=0.1573]

Test Loss: 4.8762, Test Accuracy: 98.69%
Test Precision: 0.9934, Recall: 0.9904, F1 Score: 0.9915
Test Balanced Accuracy: 0.9904

Confusion Matrix (Highlighted Mismatched)
```

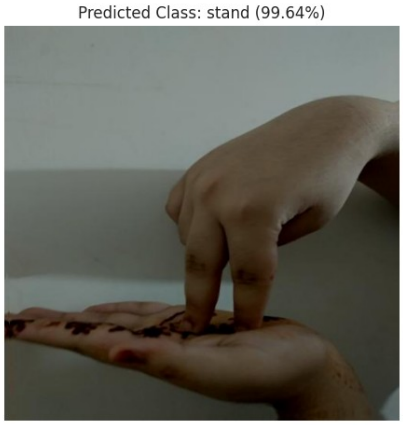


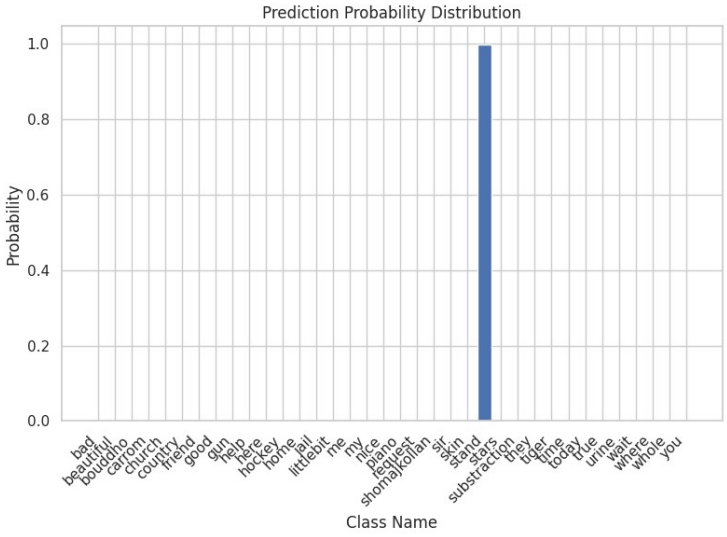

```
GPU_unload()

def load_model(model_path, fine_models, num_classes):
    fusion_model = TransformerFusionModel(models=fine_models,
num_classes=num_classes).to("cuda")
    checkpoint = torch.load(model_path)
    fusion_model.load_state_dict(checkpoint['model_state_dict'])
    fusion_model.eval()
    return fusion_model

# Function to preprocess the image
def preprocess_image(image_path):
    transform =
models.ConvNeXt_Large_Weights.IMAGENET1K_V1.transforms()
    image = Image.open(image_path).convert('RGB')
    return transform(image).unsqueeze(0).to("cuda")
```

```python
# Function to predict and visualize results
def predict_and_visualize(image_path, model, class_names, top_k=5):
    # Preprocess the image
    input_tensor = preprocess_image(image_path)

    # Predict the class probabilities
    with torch.no_grad():
        outputs = model(input_tensor)
        probs = F.softmax(outputs, dim=1)[0].cpu().numpy()

    # Get the predicted class and top K predictions
    predicted_index = np.argmax(probs)
    predicted_class = class_names[predicted_index]
    top_probs, top_indices = torch.topk(torch.tensor(probs), k=top_k)

    # Load and visualize the image
    img = Image.open(image_path)
    plt.figure(figsize=(15, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(img)
    plt.axis('off')
    plt.title(f"Predicted Class: {predicted_class}
({probs[predicted_index] * 100:.2f}%)")

    # Plot the probability distribution
    plt.subplot(1, 2, 2)
    plt.bar(class_names, probs)
    plt.xticks(rotation=45, ha='right')
    plt.xlabel('Class Name')
    plt.ylabel('Probability')
    plt.title('Prediction Probability Distribution')
    plt.tight_layout()
    plt.show()

    # Print the top K predictions
    print("\nTop {} Predictions:".format(top_k))
    for i in range(top_k):
        print(f"{class_names[top_indices[i]]}: {top_probs[i].item() *
100:.2f}%")

# Usage example
model_path = '/content/fusion_model.pth'

# Load the model
fusion_model = load_model(model_path, fine_models, num_classes)

# Predict and visualize
image_path =
'/content/BdSLW41WordDatasetSplittedAugmented/test2/stand/stand_2_aug_
```

```
3.jpg'
predict_and_visualize(image_path, fusion_model, train_dataset.classes)
```

```
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/
transformer.py:379: UserWarning: enable_nested_tensor is True, but
self.use_nested_tensor is False because
encoder_layer.self_attn.batch_first was not True(use batch_first for
better inference performance)
  warnings.warn(
<ipython-input-60-4f8664970776>:5: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  checkpoint = torch.load(model_path)
```






```
Top 5 Predictions:
stand: 99.64%
nice: 0.03%
skin: 0.02%
```

```
tiger: 0.02%
littlebit: 0.02%

GPU_unload()

load_architectures()

fine_models = [convnext, densenet, efficientnet, regnet, resnet, swin]

# Set all models to evaluation mode and freeze parameters
for model in fine_models:
    model.eval()
    for param in model.parameters():
        param.requires_grad = False

/usr/local/lib/python3.10/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
<ipython-input-21-cd410b975fbd>:7: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/ConvNeXt_{model_best_training_epoch[0]}.pth
")
<ipython-input-21-cd410b975fbd>:12: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
```

```
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/DenseNet_{model_best_training_epoch[1]}.pth
")
<ipython-input-21-cd410b975fbd>:17: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/EfficientNet_{model_best_training_epoch[2]}
.pth")
<ipython-input-21-cd410b975fbd>:22: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/RegNet_{model_best_training_epoch[3]}.pth")
<ipython-input-21-cd410b975fbd>:27: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
```

```
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/ResNet_{model_best_training_epoch[4]}.pth")
<ipython-input-21-cd410b975fbd>:31: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict =
torch.load(f"{folder_name}/SwinTransformer_{model_best_training_epoch[
5]}.pth")

class FeatureExtractor:
    def __init__(self, models, device='cuda'):
        self.models = models
        self.device = device
        # Access transforms directly from torchvision.models
        self.preprocess =
torchvision.models.ConvNeXt_Large_Weights.IMAGENET1K_V1.transforms() #
Changed line

    def extract_batch_features(self, inputs):
        features = [model(inputs).cpu().detach() for model in
self.models]
        return torch.cat(features, dim=1)

    def extract_dataset_features(self, dataloader, desc="Extracting
Features"):
        features, labels = [], []
```

```
        for inputs, targets in tqdm(dataloader, desc=desc):
            inputs = inputs.to(self.device)
            batch_features = self.extract_batch_features(inputs)
            features.append(batch_features)
            labels.append(targets)
        return torch.cat(features), torch.cat(labels)

class EnsembleModel(nn.Module):
    def __init__(self, input_size, config):
        super().__init__()
        self.layers = self._build_layers(input_size, config)

    def _build_layers(self, input_size, config):
        layers = []
        prev_size = input_size

        for size in config['hidden_sizes']:
            layers.extend([
                nn.Linear(prev_size, size),
                getattr(nn, config['activation'])(),
                nn.BatchNorm1d(size),
                nn.Dropout(config['dropout'])
            ])
            prev_size = size

        layers.append(nn.Linear(prev_size, config['num_classes']))
        return nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)

class ModelTrainer:
    def __init__(self, model, config, device='cuda'):
        self.model = model
        self.config = config
        self.device = device
        self.optimizer = optim.AdamW(
            model.parameters(),
            lr=config['lr'],
            weight_decay=config['weight_decay']
        )
        self.criterion = nn.CrossEntropyLoss(
            label_smoothing=config['label_smoothing']
        )
        self.scheduler = optim.lr_scheduler.ReduceLROnPlateau(
            self.optimizer,
            patience=config['patience'],
            factor=config['factor']
        )
```

```
    def train_epoch(self, dataloader):
        self.model.train()
        total_loss = 0
        correct = 0
        total = 0

        for inputs, targets in tqdm(dataloader, desc="Training"):
            inputs, targets = inputs.to(self.device),
targets.to(self.device)
            self.optimizer.zero_grad()

            outputs = self.model(inputs)
            loss = self.criterion(outputs, targets)
            loss.backward()

            if self.config['gradient_clipping']:
                nn.utils.clip_grad_norm_(self.model.parameters(), 1.0)

            self.optimizer.step()

            total_loss += loss.item()
            _, predicted = outputs.max(1)
            correct += predicted.eq(targets).sum().item()
            total += targets.size(0)

        return total_loss / len(dataloader), 100. * correct / total

    def validate(self, dataloader):
        self.model.eval()
        total_loss = 0
        correct = 0
        total = 0
        predictions = []
        true_labels = []

        with torch.no_grad():
            for inputs, targets in tqdm(dataloader,
desc="Validating"):
                inputs, targets = inputs.to(self.device),
targets.to(self.device)
                outputs = self.model(inputs)
                loss = self.criterion(outputs, targets)

                total_loss += loss.item()
                _, predicted = outputs.max(1)
                correct += predicted.eq(targets).sum().item()
                total += targets.size(0)

                predictions.extend(predicted.cpu().numpy())
                true_labels.extend(targets.cpu().numpy())
```

```
        return (
            total_loss / len(dataloader),
            100. * correct / total,
            predictions,
            true_labels
        )

def train_model(config, feature_extractor, train_loader, val_loader):
    # Extract features
    train_features, train_labels =
feature_extractor.extract_dataset_features(train_loader)
    val_features, val_labels =
feature_extractor.extract_dataset_features(val_loader)

    # Initialize model
    model = EnsembleModel(
        input_size=train_features.shape[1],
        config=config
    ).to(feature_extractor.device)

    # Initialize trainer
    trainer = ModelTrainer(model, config)

    # Training history
    history = {
        'train_loss': [], 'train_acc': [],
        'val_loss': [], 'val_acc': []
    }

    # Training loop
    best_val_acc = 0
    for epoch in range(config['epochs']):
        # Train
        train_loss, train_acc = trainer.train_epoch(
            DataLoader(
                TensorDataset(train_features, train_labels),
                batch_size=config['batch_size'],
                shuffle=True
            )
        )

        # Validate
        val_loss, val_acc, predictions, true_labels =
trainer.validate(
            DataLoader(
                TensorDataset(val_features, val_labels),
                batch_size=config['batch_size']
            )
        )
```

```
        # Update history
        history['train_loss'].append(train_loss)
        history['train_acc'].append(train_acc)
        history['val_loss'].append(val_loss)
        history['val_acc'].append(val_acc)

        # Save best model
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            torch.save({
                'model_state_dict': model.state_dict(),
                'config': config,
                'val_acc': val_acc
            }, 'best_ensemble_model.pth')

        print(f"Epoch {epoch+1}/{config['epochs']}")
        print(f"Train Loss: {train_loss:.4f}, Train Acc:
{train_acc:.2f}%")
        print(f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%")

        trainer.scheduler.step(val_loss)

    dv.plot_confusion_matrix(true_labels,predictions,class_names)

    history_df = pd.DataFrame(history)
    sns.lineplot(x=history_df.index, y='val_acc', data=history_df)
    plt.xlabel("Epoch")
    plt.ylabel("Validation Accuracy")
    plt.title("Validation Accuracy Over Epochs")
    plt.show()
    return model, history

def test_model(best_model):
    # Testing dataset accuracy
    best_model.eval()
    test_correct, test_total = 0, 0

    predictions = []
    true_labels = []

    test_dataset_path =
"/content/BdSLW41WordDatasetSplittedAugmented/test"
    test_dataset = datasets.ImageFolder(root=test_dataset_path,
transform=preprocess)
    test_loader = DataLoader(test_dataset, batch_size=32,
shuffle=False)

    with torch.no_grad():
        for inputs, targets in test_loader:
```

```
            inputs, targets = inputs.to("cuda"), targets.to("cuda")

            # Extract features for the test data using the same
fine_models
            batch_features = [model(inputs).cpu().detach() for model
in fine_models]
            features = torch.cat(batch_features, dim=1).to("cuda") #
Move features to cuda

            # Flatten features if necessary
            features = features.view(features.size(0), -1)

            # Now pass the extracted features to the best_model
            outputs = best_model(features)
            _, predicted = outputs.max(1)
            test_correct += (predicted == targets).sum().item()
            test_total += targets.size(0)

            predictions.extend(predicted.cpu().numpy())
            true_labels.extend(targets.cpu().numpy())

    test_accuracy = 100 * test_correct / test_total
    print(f"Testing Accuracy: {test_accuracy:.6f}%")

    dv.plot_confusion_matrix(torch.tensor(true_labels),
torch.tensor(predictions), class_names)

def predict_image(image_path, model, feature_extractor, class_names):
    # Load and preprocess image
    image = Image.open(image_path)
    image =
feature_extractor.preprocess(image).unsqueeze(0).to(feature_extractor.
device)

    # Extract features
    features = feature_extractor.extract_batch_features(image)

    # Get prediction
    model.eval()
    with torch.no_grad():
        outputs = model(features.to(feature_extractor.device))
        probabilities = torch.softmax(outputs, dim=1)
        pred_class = torch.argmax(probabilities, dim=1).item()
        confidence = probabilities[0][pred_class].item()

    return class_names[pred_class], confidence

# Dataset preparation
preprocess = models.ConvNeXt_Large_Weights.IMAGENET1K_V1.transforms()

train_dataset =
```

```
datasets.ImageFolder(root="/content/BdSLW41WordDatasetSplittedAugmente
d/train", transform=preprocess)
val_dataset =
datasets.ImageFolder(root="/content/BdSLW41WordDatasetSplittedAugmente
d/val", transform=preprocess)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

class_names = train_dataset.classes

# Function to extract features
def extract_features(models, dataloader):
    features, labels = [], []

    for inputs, targets in tqdm(dataloader, desc="Extracting
Features"):
        inputs = inputs.to("cuda")
        batch_features = [model(inputs).cpu().detach() for model in
models]
        features.append(torch.cat(batch_features, dim=1))
        labels.append(targets)

    return torch.cat(features), torch.cat(labels)

# Extract features
train_features, train_labels = extract_features(fine_models,
train_loader)
val_features, val_labels = extract_features(fine_models, val_loader)

# Flatten features if necessary
train_features = train_features.view(train_features.size(0), -1)
val_features = val_features.view(val_features.size(0), -1)
```

```
Extracting Features: 100%|██████████| 81/81 [00:57<00:00,  1.41it/s]
Extracting Features: 100%|██████████| 27/27 [00:18<00:00,  1.44it/s]
```

```
from torch.optim.lr_scheduler import ReduceLROnPlateau
# Define Neural Architecture Search (NAS) model
class NASModel(nn.Module):
    def __init__(self, input_size, hidden_size, activation,
num_classes, num_layers=16, dropout_prob=0.35):
        super(NASModel, self).__init__()
        layers = [
            nn.Linear(input_size, hidden_size),
            activation,
            nn.BatchNorm1d(hidden_size),
        ]

        for _ in range(num_layers - 1):
            layers.extend([
                nn.Linear(hidden_size, hidden_size),
```

```
                activation,
                nn.BatchNorm1d(hidden_size),
                nn.Dropout(dropout_prob),
            ])

        layers.append(nn.Linear(hidden_size, num_classes))
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)


# Optuna objective function
def objective(trial):
    # Hyperparameter suggestions
    hidden_size = trial.suggest_int("hidden_size", 128, 1024,
step=128)
    activation = getattr(nn, trial.suggest_categorical("activation",
["ReLU", "LeakyReLU", "GELU", "ELU", "SELU"]))()
    lr = trial.suggest_loguniform("lr", 1e-4, 5e-4)
    num_layers = trial.suggest_int("num_layers", 16, 32)
    dropout_prob = trial.suggest_uniform("dropout_prob", 0.2, 0.5)
    label_smoothing = trial.suggest_float("label_smoothing", 0.01,
0.05)
    weight_decay = trial.suggest_loguniform("weight_decay", 1e-6, 1e-
5)
    patience = trial.suggest_int("patience", 1, 5)
    factor = trial.suggest_float("factor", 0.1, 0.5)
    gradient_clipping = trial.suggest_categorical("gradient_clipping",
[True, False])

    # Model initialization
    input_size = train_features.shape[1]
    model = NASModel(input_size, hidden_size, activation,
num_classes=36, num_layers=num_layers,
dropout_prob=dropout_prob).to("cuda")
    optimizer = optim.AdamW(model.parameters(), lr=lr,
weight_decay=weight_decay)
    criterion = nn.CrossEntropyLoss(label_smoothing=label_smoothing)
    scheduler = ReduceLROnPlateau(optimizer, mode="min",
patience=patience, factor=factor)

    # Training loop
    model.train()
    for epoch in range(3):  # Short training for NAS
        for inputs, targets in
DataLoader(TensorDataset(train_features, train_labels), batch_size=32,
shuffle=True):
            inputs, targets = inputs.to("cuda"), targets.to("cuda")
            optimizer.zero_grad()
```

```
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            if gradient_clipping:
                nn.utils.clip_grad_norm_(model.parameters(),
max_norm=1.0)
            optimizer.step()
        scheduler.step(loss.item())

    # Validation
    model.eval()
    val_correct, val_total = 0, 0
    with torch.no_grad():
        for inputs, targets in DataLoader(TensorDataset(val_features,
val_labels), batch_size=32, shuffle=False):
            inputs, targets = inputs.to("cuda"), targets.to("cuda")
            outputs = model(inputs)
            val_correct += (outputs.argmax(dim=1) ==
targets).sum().item()
            val_total += targets.size(0)

    return val_correct / val_total  # Validation accuracy

# Run Optuna optimization
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=50)
```

```
[I 2024-12-14 04:51:30,648] A new study created in memory with name:
no-name-45a1145c-9715-4282-915b-9f1e1af46cd6
<ipython-input-65-10a46f128744>:32: FutureWarning: suggest_loguniform
has been deprecated in v3.0.0. This feature will be removed in v6.0.0.
See https://github.com/optuna/optuna/releases/tag/v3.0.0. Use
suggest_float(..., log=True) instead.
  lr = trial.suggest_loguniform("lr", 1e-4, 5e-4)
<ipython-input-65-10a46f128744>:34: FutureWarning: suggest_uniform has
been deprecated in v3.0.0. This feature will be removed in v6.0.0. See
https://github.com/optuna/optuna/releases/tag/v3.0.0. Use
suggest_float instead.
  dropout_prob = trial.suggest_uniform("dropout_prob", 0.2, 0.5)
<ipython-input-65-10a46f128744>:36: FutureWarning: suggest_loguniform
has been deprecated in v3.0.0. This feature will be removed in v6.0.0.
See https://github.com/optuna/optuna/releases/tag/v3.0.0. Use
suggest_float(..., log=True) instead.
  weight_decay = trial.suggest_loguniform("weight_decay", 1e-6, 1e-5)
[I 2024-12-14 04:51:33,407] Trial 0 finished with value:
0.7398809523809524 and parameters: {'hidden_size': 384, 'activation':
'ReLU', 'lr': 0.00046615648374392996, 'num_layers': 16,
'dropout_prob': 0.31690595343724615, 'label_smoothing':
0.01805532139328512, 'weight_decay': 1.5248018293829383e-06,
'patience': 3, 'factor': 0.2581219995080742, 'gradient_clipping':
```

```
False}. Best is trial 0 with value: 0.7398809523809524.
[I 2024-12-14 04:51:38,063] Trial 1 finished with value:
0.05952380952380952 and parameters: {'hidden_size': 128, 'activation':
'ELU', 'lr': 0.00022405697550568576, 'num_layers': 30, 'dropout_prob':
0.3636007495717135, 'label_smoothing': 0.013618549114570682,
'weight_decay': 2.7268053492015406e-06, 'patience': 5, 'factor':
0.4747701795157936, 'gradient_clipping': True}. Best is trial 0 with
value: 0.7398809523809524.
```

```
all_trials_sorted = sorted(study.trials, key=lambda t: t.value,
reverse=True)

# Retrieve top 10 trials
top_10_trials = all_trials_sorted[:10]

# Store top trials in a dictionary
top_trials_data = []
for i, trial in enumerate(top_10_trials):
    trial_data = {
        "rank": i + 1,
        "value": trial.value,  # Objective value (validation accuracy)
        "params": trial.params,  # Hyperparameters
    }
    top_trials_data.append(trial_data)

# Save top trials to a JSON file
with open("top_10_trials.json", "w") as f:
    json.dump(top_trials_data, f, indent=4)

# Print top trials
print("Top 10 Trials:")
for trial in top_trials_data:
    print(f"Rank: {trial['rank']}, Value: {trial['value']:.4f},
Params: {trial['params']}\n")
```

```
Top 10 Trials:
Rank: 1, Value: 0.7399, Params: {'hidden_size': 384, 'activation':
'ReLU', 'lr': 0.00046615648374392996, 'num_layers': 16,
'dropout_prob': 0.31690595343724615, 'label_smoothing':
0.01805532139328512, 'weight_decay': 1.5248018293829383e-06,
'patience': 3, 'factor': 0.2581219995080742, 'gradient_clipping':
False}

Rank: 2, Value: 0.0595, Params: {'hidden_size': 128, 'activation':
'ELU', 'lr': 0.00022405697550568576, 'num_layers': 30, 'dropout_prob':
0.3636007495717135, 'label_smoothing': 0.013618549114570682,
'weight_decay': 2.7268053492015406e-06, 'patience': 5, 'factor':
0.4747701795157936, 'gradient_clipping': True}
```

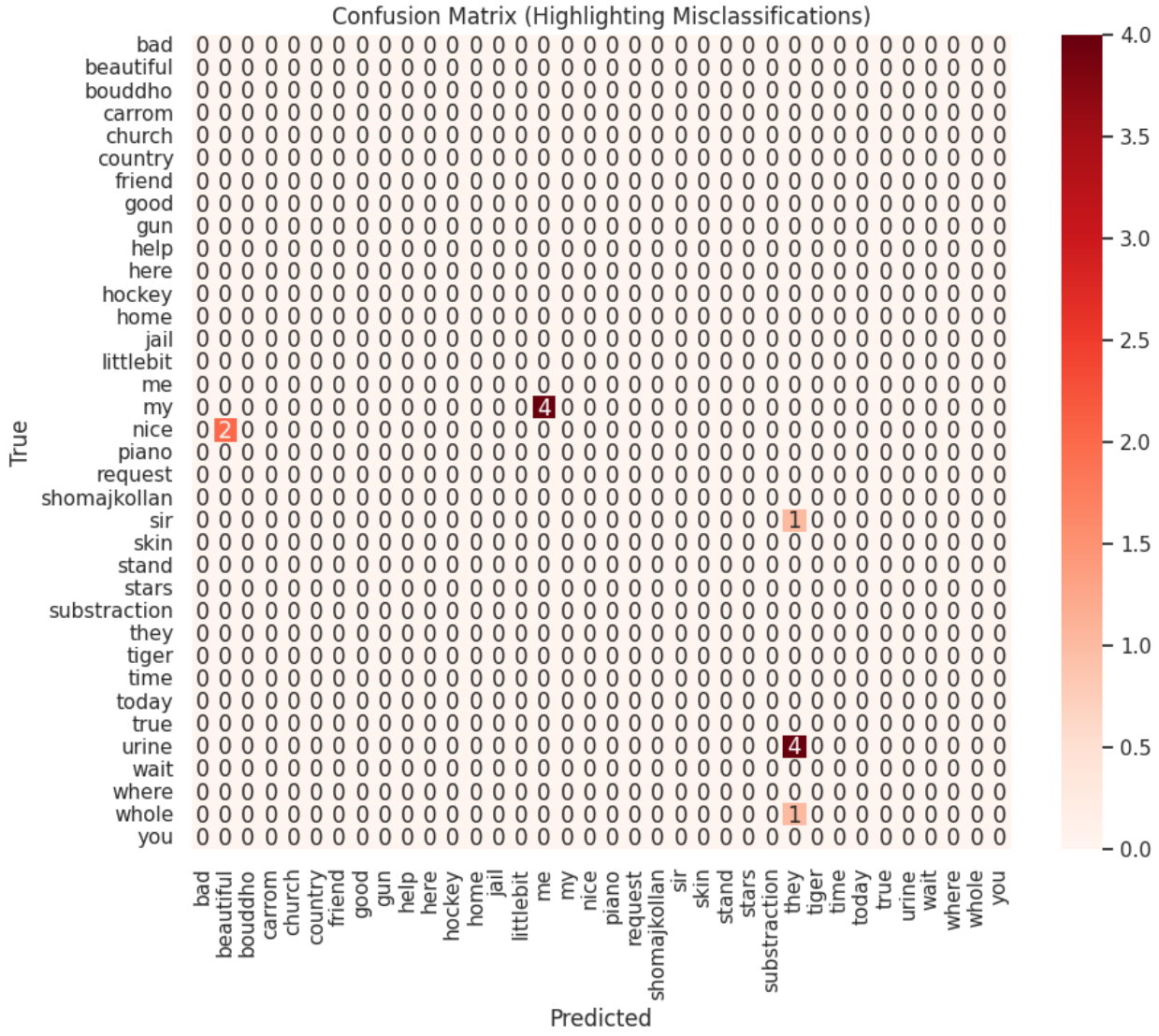
```python
if __name__ == "__main__":
    # Configuration
    config = {
        'hidden_sizes': [study.best_trial.params["hidden_size"]],
        'activation': study.best_trial.params["activation"],
        'dropout': study.best_trial.params["dropout_prob"],
        'num_classes': 36,
        'lr': study.best_trial.params["lr"],
        'weight_decay': study.best_trial.params["weight_decay"],
        'label_smoothing': study.best_trial.params["label_smoothing"],
        'patience': study.best_trial.params["patience"],
        'factor': study.best_trial.params["factor"],
        'gradient_clipping':
study.best_trial.params["gradient_clipping"],
        'epochs': 100,
        'batch_size': 16
    }

    # Initialize feature extractor with your models
    feature_extractor = FeatureExtractor(fine_models)

    # Train model
    model, history = train_model(config, feature_extractor,
train_loader, val_loader)

    torch.save({'model_state_dict':
model.state_dict()},'/content/NAS_model.pth')
```

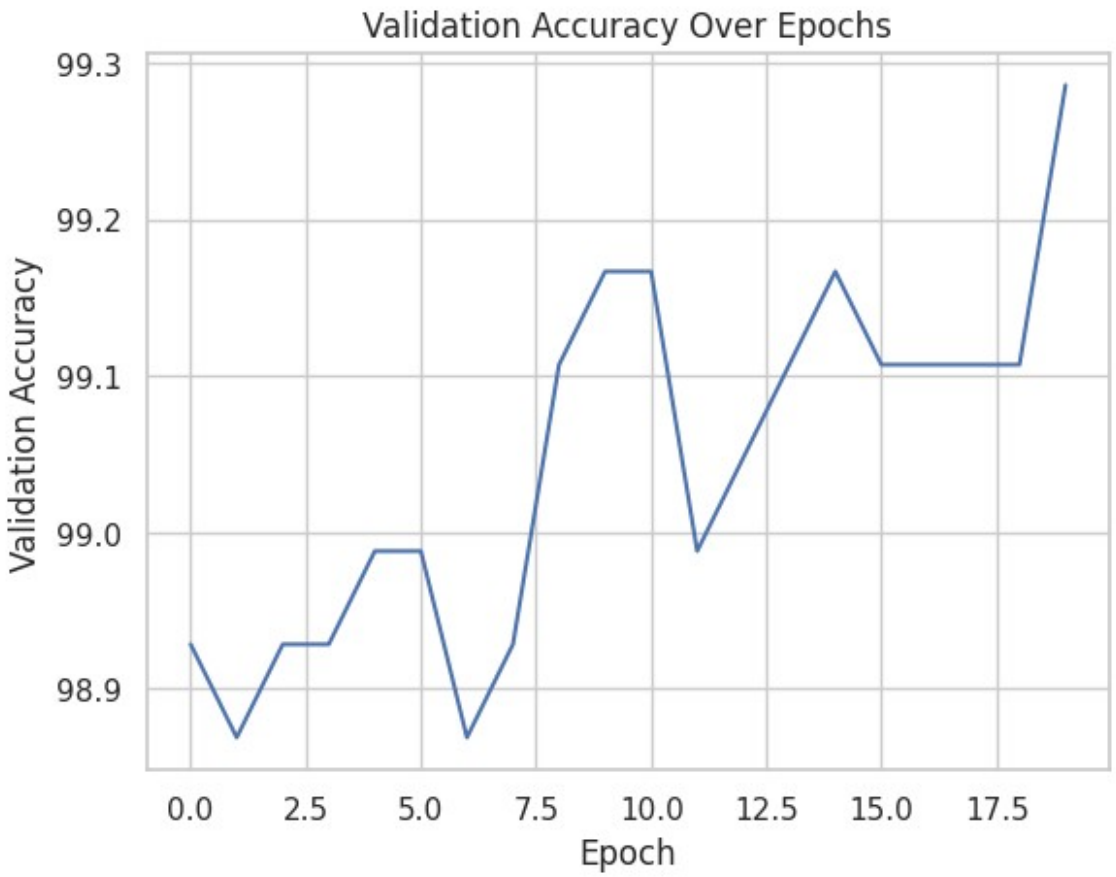
```
Extracting Features: 100%|██████████| 81/81 [01:00<00:00,  1.35it/s]
Extracting Features: 100%|██████████| 27/27 [00:20<00:00,  1.33it/s]
Training: 100%|██████████| 321/321 [00:00<00:00, 611.28it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1470.42it/s]

Epoch 1/20
Train Loss: 0.3626, Train Acc: 96.82%
Val Loss: 0.1924, Val Acc: 98.93%

Training: 100%|██████████| 321/321 [00:00<00:00, 648.79it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1564.92it/s]

Epoch 2/20
Train Loss: 0.1623, Train Acc: 100.00%
Val Loss: 0.1941, Val Acc: 98.87%

Training: 100%|██████████| 321/321 [00:00<00:00, 649.73it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1421.87it/s]

Epoch 3/20
Train Loss: 0.1611, Train Acc: 100.00%
Val Loss: 0.1905, Val Acc: 98.93%
```

```
Training: 100%|██████████| 321/321 [00:00<00:00, 636.94it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1500.62it/s]

Epoch 4/20
Train Loss: 0.1603, Train Acc: 100.00%
Val Loss: 0.1879, Val Acc: 98.93%

Training: 100%|██████████| 321/321 [00:00<00:00, 626.59it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1488.87it/s]

Epoch 5/20
Train Loss: 0.1597, Train Acc: 100.00%
Val Loss: 0.1898, Val Acc: 98.99%

Training: 100%|██████████| 321/321 [00:00<00:00, 624.84it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1416.12it/s]

Epoch 6/20
Train Loss: 0.1591, Train Acc: 100.00%
Val Loss: 0.1877, Val Acc: 98.99%

Training: 100%|██████████| 321/321 [00:00<00:00, 616.13it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1482.56it/s]

Epoch 7/20
Train Loss: 0.1590, Train Acc: 100.00%
Val Loss: 0.1910, Val Acc: 98.87%

Training: 100%|██████████| 321/321 [00:00<00:00, 634.10it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1479.84it/s]

Epoch 8/20
Train Loss: 0.1583, Train Acc: 100.00%
Val Loss: 0.1907, Val Acc: 98.93%

Training: 100%|██████████| 321/321 [00:00<00:00, 665.54it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1429.40it/s]

Epoch 9/20
Train Loss: 0.1584, Train Acc: 100.00%
Val Loss: 0.1869, Val Acc: 99.11%

Training: 100%|██████████| 321/321 [00:00<00:00, 580.58it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1392.43it/s]

Epoch 10/20
Train Loss: 0.1579, Train Acc: 100.00%
Val Loss: 0.1854, Val Acc: 99.17%

Training: 100%|██████████| 321/321 [00:00<00:00, 626.21it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1525.65it/s]
```

```
Epoch 11/20
Train Loss: 0.1577, Train Acc: 100.00%
Val Loss: 0.1851, Val Acc: 99.17%

Training: 100%|██████████| 321/321 [00:00<00:00, 606.38it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1401.28it/s]

Epoch 12/20
Train Loss: 0.1575, Train Acc: 100.00%
Val Loss: 0.1856, Val Acc: 98.99%

Training: 100%|██████████| 321/321 [00:00<00:00, 602.10it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1375.48it/s]

Epoch 13/20
Train Loss: 0.1571, Train Acc: 100.00%
Val Loss: 0.1844, Val Acc: 99.05%

Training: 100%|██████████| 321/321 [00:00<00:00, 603.26it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1251.99it/s]

Epoch 14/20
Train Loss: 0.1573, Train Acc: 100.00%
Val Loss: 0.1838, Val Acc: 99.11%

Training: 100%|██████████| 321/321 [00:00<00:00, 576.44it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1371.63it/s]

Epoch 15/20
Train Loss: 0.1569, Train Acc: 100.00%
Val Loss: 0.1826, Val Acc: 99.17%

Training: 100%|██████████| 321/321 [00:00<00:00, 578.26it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1204.89it/s]

Epoch 16/20
Train Loss: 0.1572, Train Acc: 100.00%
Val Loss: 0.1839, Val Acc: 99.11%

Training: 100%|██████████| 321/321 [00:00<00:00, 591.42it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1276.69it/s]

Epoch 17/20
Train Loss: 0.1570, Train Acc: 100.00%
Val Loss: 0.1846, Val Acc: 99.11%

Training: 100%|██████████| 321/321 [00:00<00:00, 585.85it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1297.69it/s]

Epoch 18/20
Train Loss: 0.1573, Train Acc: 100.00%
Val Loss: 0.1840, Val Acc: 99.11%
```

```
Training: 100%|██████████| 321/321 [00:00<00:00, 602.42it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1365.25it/s]

Epoch 19/20
Train Loss: 0.1565, Train Acc: 100.00%
Val Loss: 0.1829, Val Acc: 99.11%

Training: 100%|██████████| 321/321 [00:00<00:00, 597.40it/s]
Validating: 100%|██████████| 105/105 [00:00<00:00, 1273.85it/s]

Epoch 20/20
Train Loss: 0.1558, Train Acc: 100.00%
Val Loss: 0.1819, Val Acc: 99.29%
```



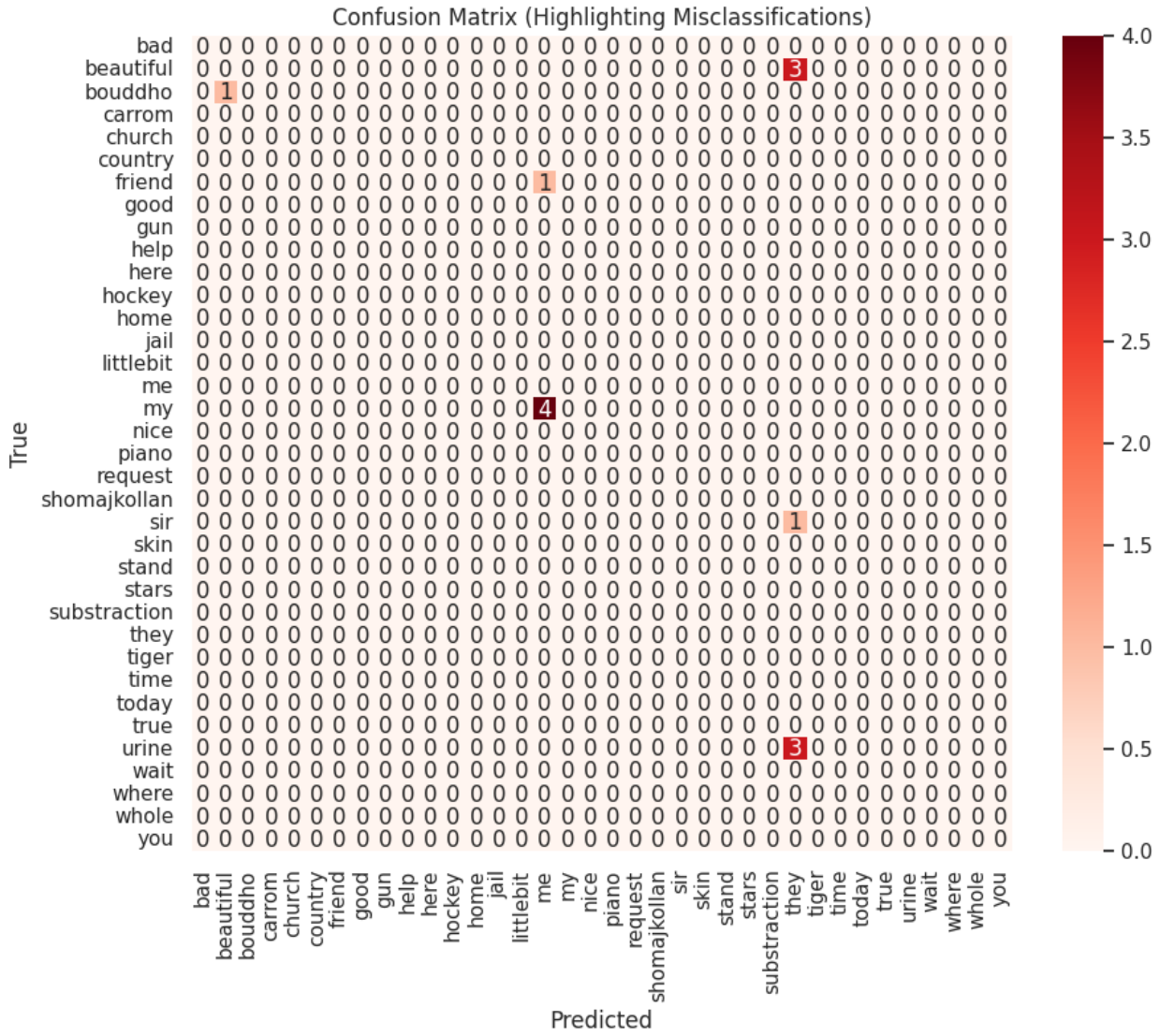



```
import pandas as pd
import torch.nn.functional as F
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

#load model
checkpoint= torch.load("/content/NAS_model.pth")
model.load_state_dict(checkpoint['model_state_dict'])
model.eval()

test_model(model)

# Function to preprocess the image
def preprocess_image(image_path):
    transform =
models.ConvNeXt_Large_Weights.IMAGENET1K_V1.transforms()
    image = Image.open(image_path).convert('RGB')
    return transform(image).unsqueeze(0).to("cuda")

# Function to predict and visualize results
def predict_and_visualize(image_path, model, class_names, top_k=5,
feature_extractor=None):
```

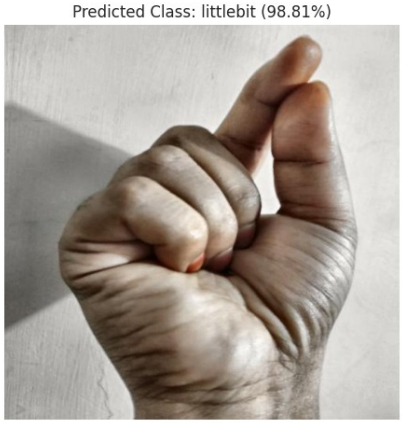
```
    # Preprocess the image
    input_tensor = preprocess_image(image_path)

    # Extract Features using the feature_extractor used during
training
    features = [m(input_tensor).cpu().detach() for m in
feature_extractor.models]
    features = torch.cat(features, dim=1).to("cuda")
    features = features.view(features.size(0), -1)  # Flatten if
necessary

    # Predict the class probabilities
    with torch.no_grad():
        outputs = model(features)
        probs = F.softmax(outputs, dim=1)[0].cpu().numpy()

    # Get the predicted class and top K predictions
    predicted_index = np.argmax(probs)
    predicted_class = class_names[predicted_index]
    top_probs, top_indices = torch.topk(torch.tensor(probs), k=top_k)
    print("\nPredictions:")
    # Load and visualize the image
    img = Image.open(image_path)
    plt.figure(figsize=(15, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(img)
    plt.axis('off')
    plt.title(f"Predicted Class: {predicted_class}
({probs[predicted_index] * 100:.2f}%)")

    # Plot the probability distribution
    plt.subplot(1, 2, 2)
    plt.bar(class_names, probs)
    plt.xticks(rotation=45, ha='right')
    plt.xlabel('Class Name')
    plt.ylabel('Probability')
    plt.title('Prediction Probability Distribution')
    plt.tight_layout()
    plt.show()

    # Print the top K predictions
    print("\nTop {} Predictions:".format(top_k))
    for i in range(top_k):
        print(f"{class_names[top_indices[i]]}: {top_probs[i].item() *
100:.2f}%")

# Predict and visualize
image_path =
'/content/BdSLW41WordDatasetSplittedAugmented/test/littlebit/littlebit
_2_aug_2.jpg'
```

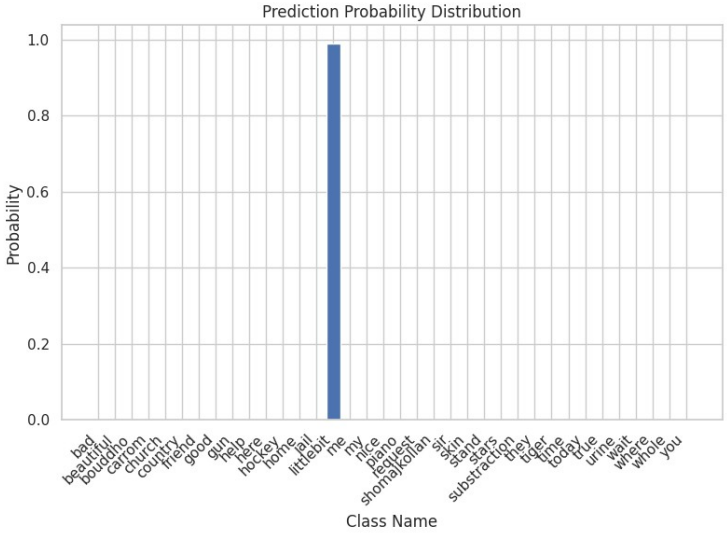
```
predict_and_visualize(image_path, model, train_dataset.classes,
feature_extractor=feature_extractor)
```

```
<ipython-input-68-f72db4419379>:8: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  checkpoint= torch.load("/content/NAS_model.pth")

Testing Accuracy: 98.689516%
```




```
Predictions:
```






```
Top 5 Predictions:
littlebit: 98.81%
stars: 0.05%
substraction: 0.05%
skin: 0.05%
today: 0.04%
```