Start coding or generate with AI.

Project Information

- Project id: 002
- Project title: Skin Lesion Classification Using Deep Learning
- Group name: hddddd
- Group members:
  - Ethan Jiang (zxxxxxxx)
  - Manojvradan Balaji Seshapriya (zxxxxxxx)
  - Sarathi Krishna (zxxxxxxx)
  - Yao Zhang (zxxxxxxx)
  - Zhuoran Liang (z5486350)

Import Packages

```
[ ]  import os
     import zipfile
     from collections import defaultdict
     import copy

     import pandas as pd
     import numpy as np
     import random

     import matplotlib.pyplot as plt
     from PIL import Image
     import seaborn as sns

     import torch
     import torch.nn as nn
     from torch.utils.data import WeightedRandomSampler, DataLoader, Dataset
     from torchvision import transforms
     from tqdm import tqdm
     import timm
     from sklearn.metrics import f1_score, accuracy_score, confusion_matrix
     from sklearn.model_selection import StratifiedKFold
     from sklearn.utils.class_weight import compute_class_weight
```

Data Engineering

Download the following files on ISIC Challenge Datasets 2019. (https://challenge.isic-archive.com/data/#2019) To run the data engineering section, it is necessary to put the notebook file into the same directory of the downloaded data files. The URLs are given as follows:

- https://isic-challenge-data.s3.amazonaws.com/2019/ISIC_2019_Training_Input.zip
- https://isic-challenge-data.s3.amazonaws.com/2019/ISIC_2019_Training_Metadata.csv
- https://isic-challenge-data.s3.amazonaws.com/2019/ISIC_2019_Training_GroundTruth.csv
- https://isic-challenge-data.s3.amazonaws.com/2019/ISIC_2019_Test_Input.zip
- https://isic-challenge-data.s3.amazonaws.com/2019/ISIC_2019_Test_Metadata.csv
- https://isic-challenge-data.s3.amazonaws.com/2019/ISIC_2019_Test_GroundTruth.csv

```
[ ]  # Unzip ISIC_2019_Training_Input.zip
     zip_path = "./ISIC_2019_Training_Input.zip"
     extract_to = "./"
     with zipfile.ZipFile(zip_path, 'r') as zip_ref:
         zip_ref.extractall(extract_to)

     # Unzip ISIC_2019_Test_Input.zip
     zip_path = "./ISIC_2019_Test_Input.zip"
     extract_to = "./"
     with zipfile.ZipFile(zip_path, 'r') as zip_ref:
         zip_ref.extractall(extract_to)
```

```
[ ]  # File paths
     img_path = "./ISIC_2019_Training_Input"
     groundtruth_file = "./ISIC_2019_Training_GroundTruth.csv"
     metadata_file = "./ISIC_2019_Training_Metadata.csv"
```

```
# Define column names
columns = ["image_id", "MEL", "NV", "BCC", "AK", "BKL", "DF", "VASC", "SCC", "UNK"]

# Load ground truth data
train_groundtruth = pd.read_csv(groundtruth_file, header=0, names=columns, dtype={"image_id": str})

# Add image path
train_groundtruth['image_path'] = train_groundtruth['image_id'].apply(lambda x: os.path.join(img_path, x + ".jpg"))
train_groundtruth['image_path'] = train_groundtruth['image_path'].apply(lambda x: x.replace("\\", "/"))

# Load metadata from the correct file
train_metadata = pd.read_csv(metadata_file, dtype={"image": str})

# Merge on image_id/image
train = pd.merge(train_groundtruth, train_metadata, left_on="image_id", right_on="image", how="left")
train = train.drop(columns=["image"])

# Save to CSV
train.to_csv("training_data.csv", index=False)
```

```
# File paths
test_img_path = "./ISIC_2019_Test_Input"
test_groundtruth_file = "./ISIC_2019_Test_GroundTruth.csv"
test_metadata_file = "./ISIC_2019_Test_Metadata.csv"

# Define column names
test_columns = ["image_id", "MEL", "NV", "BCC", "AK", "BKL", "DF", "VASC", "SCC", "UNK", "score_weight", "validation_weight"]

# Load ground truth data
train_groundtruth = pd.read_csv(test_groundtruth_file, header=0, names=test_columns)

# Add image path
train_groundtruth['image_path'] = train_groundtruth['image_id'].apply(lambda x: os.path.join(test_img_path, x + ".jpg"))
train_groundtruth['image_path'] = train_groundtruth['image_path'].apply(lambda x: x.replace("\\", "/"))

# Load metadata from the correct file
test_metadata = pd.read_csv(test_metadata_file, dtype={"image": str})

# Merge on image_id/image
test = pd.merge(train_groundtruth, test_metadata, left_on="image_id", right_on="image", how="left")
test = test.drop(columns=["image"])

# Save to CSV
test.to_csv("testing_data.csv", index=False)
```

Data Exploration

Data Wrangling

The data engineering section has already unzip the image folder and create csv files for training and testing dataset including the image paths, corresponding labels, and metadatas. In this section, train and test dataset will be imported from their file directories. In addition, this research ignore the UNK label because UNK means unknown, and there are 0 UNK label is contained in training dataset, which might have negative effect to the model convergence if UNK is counted as a class.

```
[ ] train = pd.read_csv("training_data.csv")
```

```
[ ] test = pd.read_csv("testing_data.csv")
```

```
[ ] # Data Munging and Aggregation
    label_cols = ["MEL", "NV", "BCC", "AK", "BKL", "DF", "VASC", "SCC"]
    num_classes = len(label_cols)

    train = train[train["UNK"] != 1].reset_index(drop=True)
    test = test[test["UNK"] != 1].reset_index(drop=True)

    train['labels'] = train[label_cols].idxmax(axis=1)
    train['labels_index'] = train['labels'].apply(lambda x: label_cols.index(x.split(';')[0]))
    test['labels'] = test[label_cols].idxmax(axis=1)
    test['labels_index'] = test['labels'].apply(lambda x: label_cols.index(x.split(';')[0]))
```

Exploratory Data Analysis

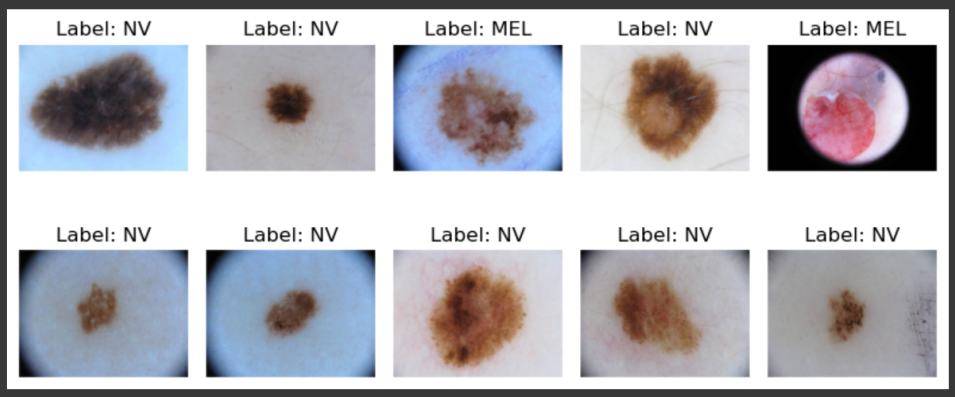
There are 25331 samples in dataset separated by 8 classes in training dataset, but the number of NV samples account for half of the total number of samples, which implies us a data imbanlance which will happen on model training. (The training models are more like to predict a testing sample to the label which has more samples because of the imbalance of class)

```
[ ] class_counts = train["labels"].value_counts()
    print(class_counts)
    print(f"total samples: {train.shape[0]}")
    print(f"total classes: {len(class_counts)}")
```

```
labels
NV      12875
MEL      4522
BCC      3323
BKL      2624
AK        867
SCC       628
VASC      253
DF        239
Name: count, dtype: int64
total samples: 25331
total classes: 8
```

In this data visualization section, the first ten samples have been visualized with their labels on the top. It is generally agreed that since this is a medical image of the skin, data augmentation in this case will have higher flexibility. (rotation, flip, and crop will not affect the the result of label) In subsequent research, data augmentation will be the main method to solve the data imbalance problem.

```
[ ] row_count = 2
    col_count = 5

    fig, axs = plt.subplots(row_count, col_count, figsize=(8, 4))
    axs = axs.flatten()

    for index in range(row_count * col_count):
        img = Image.open(train['image_path'][0:10].iloc[index])
        axs[index].imshow(img)
        axs[index].set_title(f"Label: {train['labels'][0:10].iloc[index]}")
        axs[index].axis("off")

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



Baseline models

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

```
NVIDIA GeForce RTX 3070 Laptop GPU
```

Preprocessing

```
[ ] class ImgDataset(Dataset):
        def __init__(self, image_paths, labels, transform=None):
            self.image_paths = image_paths
            self.labels = labels
            self.transform = transform

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

        def __getitem__(self, index):
            image = Image.open(self.image_paths[index]).convert("RGB")
            label = torch.tensor(self.labels[index], dtype=torch.long)

            if self.transform:
                image = self.transform(image)

            return image, label
```

```
[ ]  transform = {
         'train': transforms.Compose([
             transforms.Resize((224, 224)),
             transforms.ToTensor(),
             transforms.Normalize([0.485, 0.456, 0.406],
                                  [0.229, 0.224, 0.225]),
         ]),
         'test': transforms.Compose([
             transforms.Resize((224, 224)),
             transforms.ToTensor(),
             transforms.Normalize([0.485, 0.456, 0.406],
                                  [0.229, 0.224, 0.225]),
         ])
     }
```

```
[ ]  x_train = np.array([i for i in train['image_path']])
     y_train = np.array([i for i in train['labels_index']])

     x_test = np.array([i for i in test['image_path']])
     y_test = np.array([i for i in test['labels_index']])
```

```
[ ]  train_dataset = ImgDataset(x_train, y_train, transform=transform['train'])
     train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

```
[ ]  test_dataset = ImgDataset(x_test, y_test, transform=transform['test'])
     test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Evaluation

```
[ ]  def evaluate(model, test_loader):
         model.eval()
         y_pred_test = []
         y_true_test = []

         with torch.no_grad():
             testing = tqdm(test_loader, total=len(test_loader), leave=True, desc="Testing")
             for img, label in testing:

                 img = img.to(device)
                 label = label.to(device)

                 output = model(img)
                 preds = torch.argmax(output, dim=1).cpu().numpy()

                 y_pred_test.extend(preds)
                 y_true_test.extend(label.cpu().numpy())

         f1 = f1_score(y_true_test, y_pred_test, average="macro")
         acc = accuracy_score(y_true_test, y_pred_test)
         return acc, f1
```

ResNet

```
[ ]  # resnet50
     torch.cuda.empty_cache()
     model = timm.create_model('resnet50', pretrained=True, num_classes=len(label_cols))
     criterion = torch.nn.CrossEntropyLoss()
     optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
     model = model.to(device)
```

```
[ ]  best_acc = 0
     best_model = None
     patience = 3
     no_improve_epochs = 0
     epoch = 0

     while True:
         model.train()
         total_loss = 0
         epoch += 1
         training = tqdm(train_loader, total=len(train_loader), leave=True, desc=f'Epoch {epoch}')

         for img, label in training:
             img = img.to(device)
             y_true = label.to(device, non_blocking=True)

             optimizer.zero_grad()
             y_pred = model(img)
             loss = criterion(y_pred, y_true)

             loss.backward()
             optimizer.step()
```

```
            total_loss += loss.item()
            training.set_postfix_str(f'loss={loss.item():.5f}')

        # Check accuracy after each epoch
        acc, f1 = evaluate(model, test_loader)
        print(f"Epoch {epoch} — Accuracy: {acc:.5f} | F1 Score: {f1:.5f}", flush=True)

        # Stop when improvement is limited
        if acc > best_acc:
            best_acc = acc
            best_model = copy.deepcopy(model)
            no_improve_epochs = 0
        else:
            no_improve_epochs += 1
            if no_improve_epochs >= patience:
                print(f"No improvement in accuracy for {patience} consecutive epochs. Stopping training.")
                break
```

```
Epoch 1: 100%|██████████| 792/792 [09:04<00:00,  1.46it/s, loss=1.10749]
Testing: 100%|██████████| 194/194 [02:16<00:00,  1.42it/s]Epoch 1 — Accuracy: 0.62332 | F1 Score: 0.28232

Epoch 2: 100%|██████████| 792/792 [08:51<00:00,  1.49it/s, loss=0.97197]
Testing: 100%|██████████| 194/194 [02:08<00:00,  1.51it/s]Epoch 2 — Accuracy: 0.64287 | F1 Score: 0.32595

Epoch 3: 100%|██████████| 792/792 [08:53<00:00,  1.48it/s, loss=0.53619]
Testing: 100%|██████████| 194/194 [02:13<00:00,  1.46it/s]Epoch 3 — Accuracy: 0.65159 | F1 Score: 0.40114

Epoch 4: 100%|██████████| 792/792 [09:07<00:00,  1.45it/s, loss=0.41359]
Testing: 100%|██████████| 194/194 [02:11<00:00,  1.47it/s]Epoch 4 — Accuracy: 0.66241 | F1 Score: 0.43457

Epoch 5: 100%|██████████| 792/792 [08:55<00:00,  1.48it/s, loss=0.26559]
Testing: 100%|██████████| 194/194 [02:11<00:00,  1.47it/s]Epoch 5 — Accuracy: 0.66031 | F1 Score: 0.46474

Epoch 6: 100%|██████████| 792/792 [08:55<00:00,  1.48it/s, loss=0.27272]
Testing: 100%|██████████| 194/194 [02:10<00:00,  1.49it/s]Epoch 6 — Accuracy: 0.65240 | F1 Score: 0.48175

Epoch 7: 100%|██████████| 792/792 [08:55<00:00,  1.48it/s, loss=0.03345]
Testing: 100%|██████████| 194/194 [02:12<00:00,  1.47it/s]Epoch 7 — Accuracy: 0.64723 | F1 Score: 0.47529
No improvement in accuracy for 3 consecutive epochs. Stopping training.
```

```
[ ]  torch.save(best_model, "baseline_resnet.pth")
```

EfficientNet

```
[ ]  # efficientnet_b3
     torch.cuda.empty_cache()
     model = timm.create_model('efficientnet_b3', pretrained=True, num_classes=len(label_cols))
     criterion = torch.nn.CrossEntropyLoss()
     optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
     model = model.to(device)
```

```
[ ]  best_acc = 0
     best_model = None
     patience = 3
     no_improve_epochs = 0
     epoch = 0

     while True:
         model.train()
         total_loss = 0
         epoch += 1
         training = tqdm(train_loader, total=len(train_loader), leave=True, desc=f'Epoch {epoch}')

         for img, label in training:
             img = img.to(device)
             y_true = label.to(device, non_blocking=True)

             optimizer.zero_grad()
             y_pred = model(img)
             loss = criterion(y_pred, y_true)

             loss.backward()
             optimizer.step()

             total_loss += loss.item()
             training.set_postfix_str(f'loss={loss.item():.5f}')

         # Check accuracy after each epoch
         acc, f1 = evaluate(model, test_loader)
         print(f"Epoch {epoch} — Accuracy: {acc:.5f} | F1 Score: {f1:.5f}", flush=True)

         # Stop when improvement is limited
         if acc > best_acc:
             best_acc = acc
             best_model = copy.deepcopy(model)
```

```
            no_improve_epochs = 0
        else:
            no_improve_epochs += 1
            if no_improve_epochs >= patience:
                print(f"No improvement in accuracy for {patience} consecutive epochs. Stopping training.")
                break
```

```
Epoch 1: 100%|██████████| 792/792 [09:19<00:00,  1.41it/s, loss=0.41969]
Testing: 100%|██████████| 194/194 [02:12<00:00,  1.46it/s]Epoch 1 — Accuracy: 0.64481 | F1 Score: 0.43671

Epoch 2: 100%|██████████| 792/792 [09:20<00:00,  1.41it/s, loss=0.50485]
Testing: 100%|██████████| 194/194 [02:11<00:00,  1.48it/s]Epoch 2 — Accuracy: 0.65450 | F1 Score: 0.50318

Epoch 3: 100%|██████████| 792/792 [09:23<00:00,  1.41it/s, loss=0.12762]
Testing: 100%|██████████| 194/194 [02:12<00:00,  1.46it/s]Epoch 3 — Accuracy: 0.65724 | F1 Score: 0.48747

Epoch 4: 100%|██████████| 792/792 [09:22<00:00,  1.41it/s, loss=0.30643]
Testing: 100%|██████████| 194/194 [02:12<00:00,  1.46it/s]Epoch 4 — Accuracy: 0.64675 | F1 Score: 0.46757

Epoch 5: 100%|██████████| 792/792 [09:22<00:00,  1.41it/s, loss=0.11532]
Testing: 100%|██████████| 194/194 [02:12<00:00,  1.47it/s]Epoch 5 — Accuracy: 0.66031 | F1 Score: 0.49895

Epoch 6: 100%|██████████| 792/792 [09:23<00:00,  1.41it/s, loss=0.04659]
Testing: 100%|██████████| 194/194 [02:12<00:00,  1.46it/s]Epoch 6 — Accuracy: 0.66161 | F1 Score: 0.48825

Epoch 7: 100%|██████████| 792/792 [09:21<00:00,  1.41it/s, loss=0.10248]
Testing: 100%|██████████| 194/194 [02:08<00:00,  1.51it/s]Epoch 7 — Accuracy: 0.66112 | F1 Score: 0.49917

Epoch 8: 100%|██████████| 792/792 [09:05<00:00,  1.45it/s, loss=0.00994]
Testing: 100%|██████████| 194/194 [02:07<00:00,  1.52it/s]Epoch 8 — Accuracy: 0.65385 | F1 Score: 0.50374

Epoch 9: 100%|██████████| 792/792 [09:06<00:00,  1.45it/s, loss=0.04569]
Testing: 100%|██████████| 194/194 [02:08<00:00,  1.52it/s]Epoch 9 — Accuracy: 0.66177 | F1 Score: 0.51247

Epoch 10: 100%|██████████| 792/792 [09:05<00:00,  1.45it/s, loss=0.39708]
Testing: 100%|██████████| 194/194 [02:08<00:00,  1.51it/s]Epoch 10 — Accuracy: 0.67017 | F1 Score: 0.51052

Epoch 11: 100%|██████████| 792/792 [09:06<00:00,  1.45it/s, loss=0.03938]
Testing: 100%|██████████| 194/194 [02:08<00:00,  1.51it/s]Epoch 11 — Accuracy: 0.66193 | F1 Score: 0.52264

Epoch 12: 100%|██████████| 792/792 [09:06<00:00,  1.45it/s, loss=0.01484]
Testing: 100%|██████████| 194/194 [02:08<00:00,  1.50it/s]Epoch 12 — Accuracy: 0.65159 | F1 Score: 0.48956

Epoch 13: 100%|██████████| 792/792 [09:07<00:00,  1.45it/s, loss=0.00976]
Testing: 100%|██████████| 194/194 [02:08<00:00,  1.51it/s]Epoch 13 — Accuracy: 0.66484 | F1 Score: 0.50442
No improvement in accuracy for 3 consecutive epochs. Stopping training.
```

```
[ ]  torch.save(best_model, "baseline_efficientnet.pth")
```

Vision Transformer

```
[ ]  # vit_base_patch16_224
     torch.cuda.empty_cache()
     model = timm.create_model('vit_base_patch16_224', pretrained=True, num_classes=len(label_cols))
     criterion = torch.nn.CrossEntropyLoss()
     optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
     model = model.to(device)
```

```
[ ]  best_acc = 0
     best_model = None
     patience = 3
     no_improve_epochs = 0
     epoch = 0

     while True:
         model.train()
         total_loss = 0
         epoch += 1
         training = tqdm(train_loader, total=len(train_loader), leave=True, desc=f'Epoch {epoch}')

         for img, label in training:
             img = img.to(device)
             y_true = label.to(device, non_blocking=True)

             optimizer.zero_grad()
             y_pred = model(img)
             loss = criterion(y_pred, y_true)

             loss.backward()
             optimizer.step()

             total_loss += loss.item()
             training.set_postfix_str(f'loss={loss.item():.5f}')

         # Check accuracy after each epoch
         acc, f1 = evaluate(model, test_loader)
         print(f"Epoch {epoch} — Accuracy: {acc:.5f} | F1 Score: {f1:.5f}", flush=True)
```

```
        # Stop when improvement is limited
        if acc > best_acc:
            best_acc = acc
            best_model = copy.deepcopy(model)
            no_improve_epochs = 0
        else:
            no_improve_epochs += 1
            if no_improve_epochs >= patience:
                print(f"No improvement in accuracy for {patience} consecutive epochs. Stopping training.")
                break
```

```
Epoch 1: 100%|██████████| 792/792 [15:28<00:00,  1.17s/it, loss=0.43290]
Testing: 100%|██████████| 194/194 [02:31<00:00,  1.28it/s]Epoch 1 — Accuracy: 0.65628 | F1 Score: 0.40340

Epoch 2: 100%|██████████| 792/792 [15:19<00:00,  1.16s/it, loss=0.73639]
Testing: 100%|██████████| 194/194 [02:30<00:00,  1.29it/s]Epoch 2 — Accuracy: 0.63318 | F1 Score: 0.45130

Epoch 3: 100%|██████████| 792/792 [15:18<00:00,  1.16s/it, loss=1.33762]
Testing: 100%|██████████| 194/194 [02:31<00:00,  1.28it/s]Epoch 3 — Accuracy: 0.63382 | F1 Score: 0.47467

Epoch 4: 100%|██████████| 792/792 [15:27<00:00,  1.17s/it, loss=0.46302]
Testing: 100%|██████████| 194/194 [02:33<00:00,  1.27it/s]Epoch 4 — Accuracy: 0.66193 | F1 Score: 0.45635

Epoch 5: 100%|██████████| 792/792 [15:34<00:00,  1.18s/it, loss=0.12554]
Testing: 100%|██████████| 194/194 [02:33<00:00,  1.26it/s]Epoch 5 — Accuracy: 0.65498 | F1 Score: 0.46413

Epoch 6: 100%|██████████| 792/792 [15:35<00:00,  1.18s/it, loss=0.07793]
Testing: 100%|██████████| 194/194 [02:34<00:00,  1.26it/s]Epoch 6 — Accuracy: 0.64998 | F1 Score: 0.48019

Epoch 7: 100%|██████████| 792/792 [15:39<00:00,  1.19s/it, loss=0.07699]
Testing: 100%|██████████| 194/194 [02:34<00:00,  1.26it/s]Epoch 7 — Accuracy: 0.64400 | F1 Score: 0.47042
No improvement in accuracy for 3 consecutive epochs. Stopping training.
```

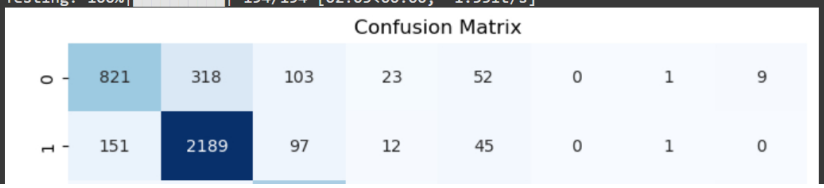
```
[ ]  torch.save(best_model, "baseline_vit.pth")
```

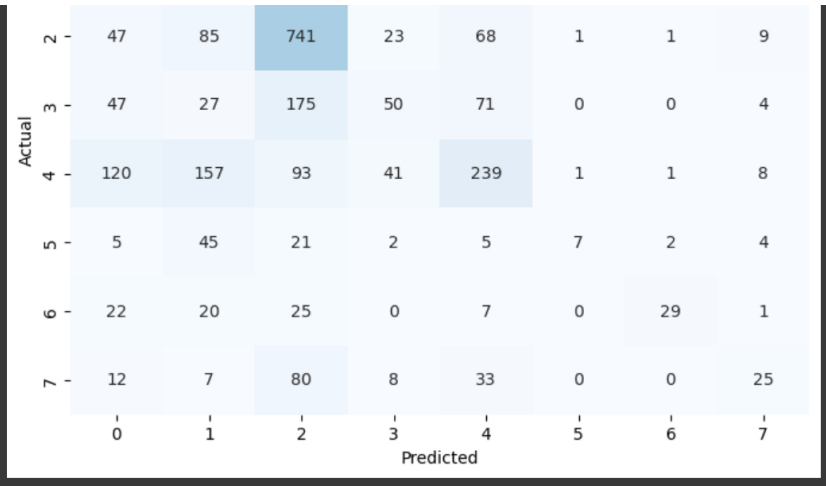
Result Analysis

```
[ ]  def result(model, test_loader, class_names=None, plot=True):
        model.eval()
        y_pred_test = []
        y_true_test = []

        with torch.no_grad():
            testing = tqdm(test_loader, total=len(test_loader), leave=True, desc="Testing")
            for img, label in testing:
                img = img.to(device)
                label = label.to(device)

                output = model(img)
                preds = torch.argmax(output, dim=1).cpu().numpy()

                y_pred_test.extend(preds)
                y_true_test.extend(label.cpu().numpy())

        acc = accuracy_score(y_true_test, y_pred_test)
        f1 = f1_score(y_true_test, y_pred_test, average="macro")
        cm = confusion_matrix(y_true_test, y_pred_test)

        if plot:
            plt.figure(figsize=(8, 6))
            sns.heatmap(
                cm, annot=True, fmt="d", cmap="Blues", cbar=False,
                xticklabels=class_names if class_names else True,
                yticklabels=class_names if class_names else True
            )
            plt.xlabel("Predicted")
            plt.ylabel("Actual")
            plt.title("Confusion Matrix")
            plt.show()

        return acc, f1, cm
```

```
[ ]  model = torch.load('baseline_resnet.pth', weights_only=False)
     model.eval()
     acc, f1, cm = result(model, test_loader, class_names=None, plot=True)
```

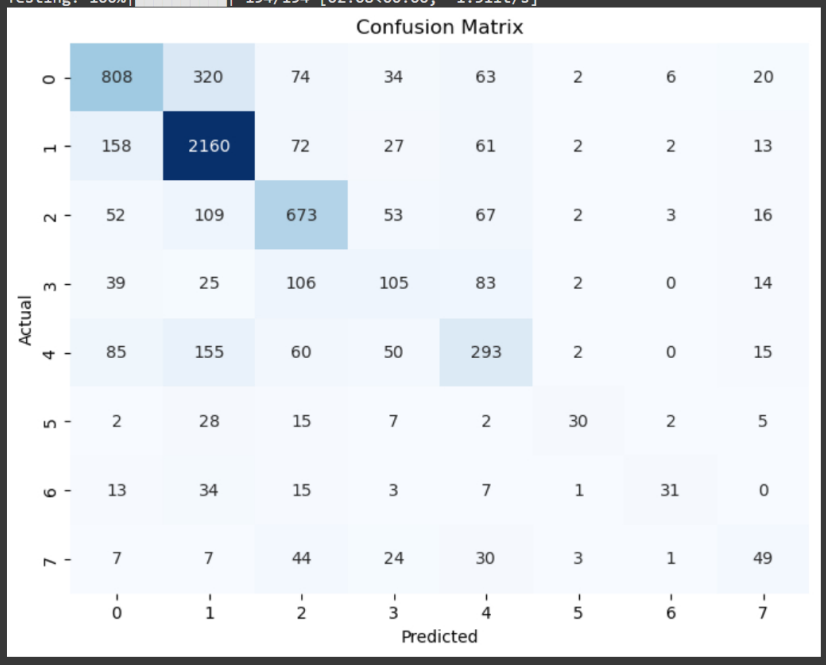
```
Testing: 100%|██████████| 194/194 [02:05<00:00,  1.55it/s]
```





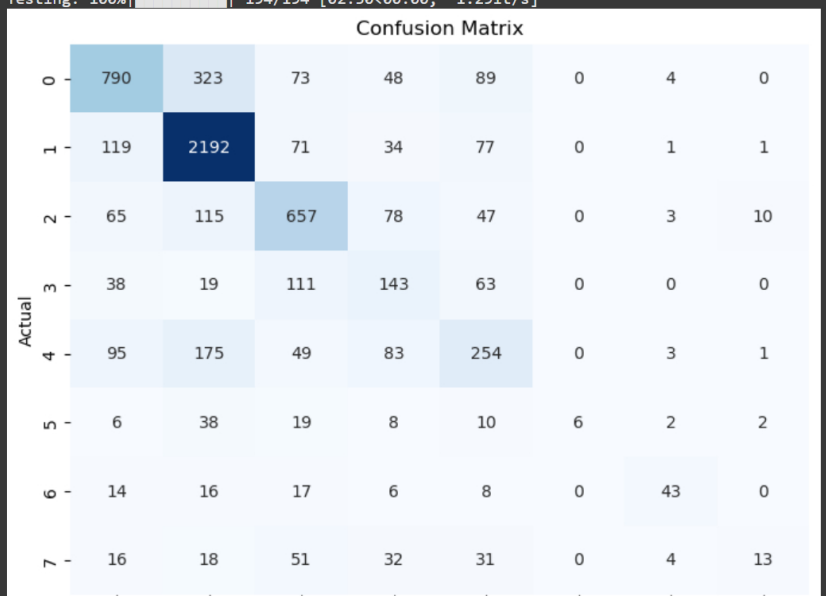

```
[ ]  model = torch.load('baseline_efficientnet.pth', weights_only=False)
     model.eval()
     acc, f1, cm = result(model, test_loader, class_names=None, plot=True)
```

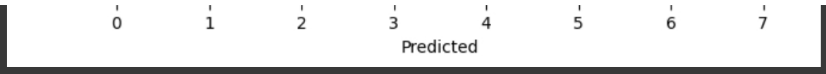
```
Testing: 100%| | 194/194 [02:08<00:00,  1.51it/s]
```




```
[ ]  model = torch.load('baseline_vit.pth', weights_only=False)
     model.eval()
     acc, f1, cm = result(model, test_loader, class_names=None, plot=True)
```

```
Testing: 100%| | 194/194 [02:30<00:00,  1.29it/s]
```





Data augmentation

```
[ ]  class ImgDataset(Dataset):
        def __init__(self, image_paths, labels, metadata, transform=None):
            self.image_paths = image_paths
            self.labels = labels
            self.metadata = metadata
            self.transform = transform

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

        def __getitem__(self, index):
            # Load image
            image = Image.open(self.image_paths[index]).convert("RGB")
            label = torch.tensor(self.labels[index], dtype=torch.long)

            # Get metadata (age, sex, location)
            meta = torch.tensor(self.metadata[index], dtype=torch.float)

            # Apply transformations on the image
            if self.transform:
                image = self.transform(image)
            return image, meta, label
```

```
[ ]  transform = {
       'train': transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(30),
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406],
                                 [0.229, 0.224, 0.225]),
        ]),
        'test': transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406],
                                 [0.229, 0.224, 0.225]),
        ])
    }
```

Balanced training and testing dataset with metadata

```
[ ]  # Create a mapping of class indices to paths and metadata
    class_to_paths = defaultdict(list)
    class_to_metadata = defaultdict(list)

    # Adjust for the missing/NaN columns (sex, age, anatom_site_general)
    for i, row in train.iterrows():
        class_idx = row['labels_index']
        image_path = row['image_path']

        # Handle sex column (1 for male, 0 for female, -1 for missing)
        sex = row['sex']
        if sex == "male":
            sex_value = 1
        elif sex == "female":
            sex_value = 0
        else:
            sex_value = -1  # For missing or unknown

        # Handle age_approx (0 for missing, otherwise use the actual value)
        age_value = row['age_approx'] if pd.notna(row['age_approx']) else 0

        # Handle anatom_site_general (One-hot encoding for locations)
        location = row['anatom_site_general']
        locations = ["anterior torso", "head/neck", "lower extremity", "upper extremity", "posterior torso", "oral_genital", "lateral torso", "palms/soles"
        location_vector = [1 if location == loc else 0 for loc in locations]

        # Store the metadata for each class
        metadata = [age_value, sex_value] + location_vector  # Combined metadata

        # Map metadata for each class index
        class_to_paths[class_idx].append(image_path)
        class_to_metadata[class_idx].append(metadata)

    # Balancing logic for paths, labels, and metadata
    UPPER_THRESHOLD = 5000
```

```
    LOWER_THRESHOLD = 1500
    train_images = []
    train_labels = []
    train_metadata = []

    for class_idx, paths in class_to_paths.items():
        current_count = len(paths)

        # Get the corresponding metadata for this class
        class_metadata = class_to_metadata[class_idx]

        # Apply balancing logic for image paths and metadata
        if current_count > UPPER_THRESHOLD:
            indices = random.sample(range(current_count), UPPER_THRESHOLD)
            sampled_paths = [paths[i] for i in indices]
            sampled_metadata = [class_metadata[i] for i in indices]
        elif current_count < LOWER_THRESHOLD:
            multiplier = LOWER_THRESHOLD // current_count
            remainder = LOWER_THRESHOLD % current_count

            sampled_paths = paths * multiplier
            sampled_metadata = class_metadata * multiplier

            remainder_indices = random.choices(range(current_count), k=remainder)
            sampled_paths += [paths[i] for i in remainder_indices]
            sampled_metadata += [class_metadata[i] for i in remainder_indices]
        else:
            sampled_paths = paths
            sampled_metadata = class_metadata

        # Append the sampled paths, labels, and metadata
        train_images.extend(sampled_paths)
        train_labels.extend([class_idx] * len(sampled_paths))
        train_metadata.extend(sampled_metadata)
```

```
[ ] locations = ["anterior torso", "head/neck", "lower extremity", "upper extremity",
                 "posterior torso", "oral_genital", "lateral torso", "palms/soles"]

    test_images = np.array(test['image_path'].tolist())
    test_labels = np.array(test['labels_index'].tolist())

    test_metadata = []
    for _, row in test.iterrows():
        sex = row['sex']
        if sex == "male":
            sex_value = 1
        elif sex == "female":
            sex_value = 0
        else:
            sex_value = -1

        age_value = row['age_approx'] if pd.notna(row['age_approx']) else 0

        location = row['anatom_site_general']
        location_vector = [1 if location == loc else 0 for loc in locations]

        metadata = [age_value, sex_value] + location_vector
        test_metadata.append(metadata)
```

```
[ ] train_dataset = ImgDataset(train_images, train_labels, train_metadata, transform=transform['train'])
    train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
```

```
[ ] test_dataset = ImgDataset(test_images, test_labels, test_metadata, transform=transform["test"])
    test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

Fusion Model

```
[ ] device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(torch.cuda.get_device_name())
    torch.cuda.empty_cache()
```

```
NVIDIA GeForce RTX 3070 Laptop GPU
```

```
[ ] def Evaluate(model):
        model.eval()
        y_pred_test = []
        y_true_test = []
        top2_correct = 0

        with torch.no_grad():
            testing = tqdm(test_loader, total=len(test_loader), leave=True, desc="Testing")
            for img, metadata, label in testing:
                img = img.to(device)
```

```
                label = label.to(device)
                metadata = metadata.to(device)
                output = model(img, metadata)

                preds = torch.argmax(output, dim=1).cpu().numpy()
                y_pred_test.extend(preds)
                y_true_test.extend(label.cpu().numpy())

                top2_preds = torch.topk(output, k=2, dim=1).indices.cpu().numpy()
                label_np = label.cpu().numpy()
                for i in range(len(label_np)):
                    if label_np[i] in top2_preds[i]:
                        top2_correct += 1

        f1_class = f1_score(y_true_test, y_pred_test, average=None)
        f1_macro = f1_score(y_true_test, y_pred_test, average="macro")
        acc = accuracy_score(y_true_test, y_pred_test)
        top2_acc = top2_correct / len(y_true_test)

        return acc, f1_class, top2_acc, f1_macro
```

[ ]
```
class MetaMLP(nn.Module):
    def __init__(self, input_dim=10, output_dim=32):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.BatchNorm1d(64),
            nn.Linear(64, output_dim),
            nn.ReLU()
        )

    def forward(self, x):
        return self.model(x)
```

[ ]
```
class FusionModel(nn.Module):
    def __init__(self, num_classes=8, metadata_dim=10, meta_out_dim=32):
        super().__init__()

        self.model = timm.create_model('vit_base_patch16_224', pretrained=True)
        self.model.head = nn.Identity()
        model_out_dim = self.model.num_features

        self.meta_mlp = MetaMLP(input_dim=metadata_dim, output_dim=meta_out_dim)

        self.classifier = nn.Sequential(
            nn.Linear(model_out_dim + meta_out_dim, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, num_classes)
        )

    def forward(self, img, metadata):
        img_feat = self.model(img)                      # [B, 768]
        meta_feat = self.meta_mlp(metadata)             # [B, 32]
        fused = torch.cat([img_feat, meta_feat], dim=1)  # [B, 800]
        out = self.classifier(fused)                    # [B, num_classes]
        return out
```

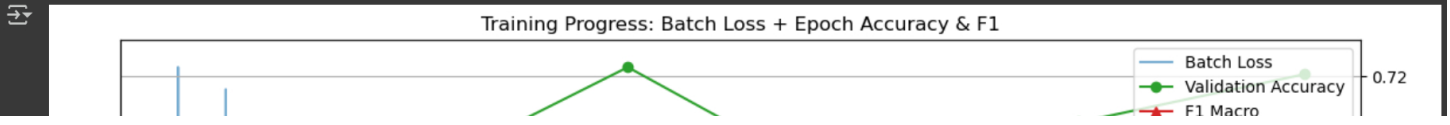
[ ]
```
torch.cuda.empty_cache()
model = FusionModel(num_classes=8)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)
model = model.to(device)
```

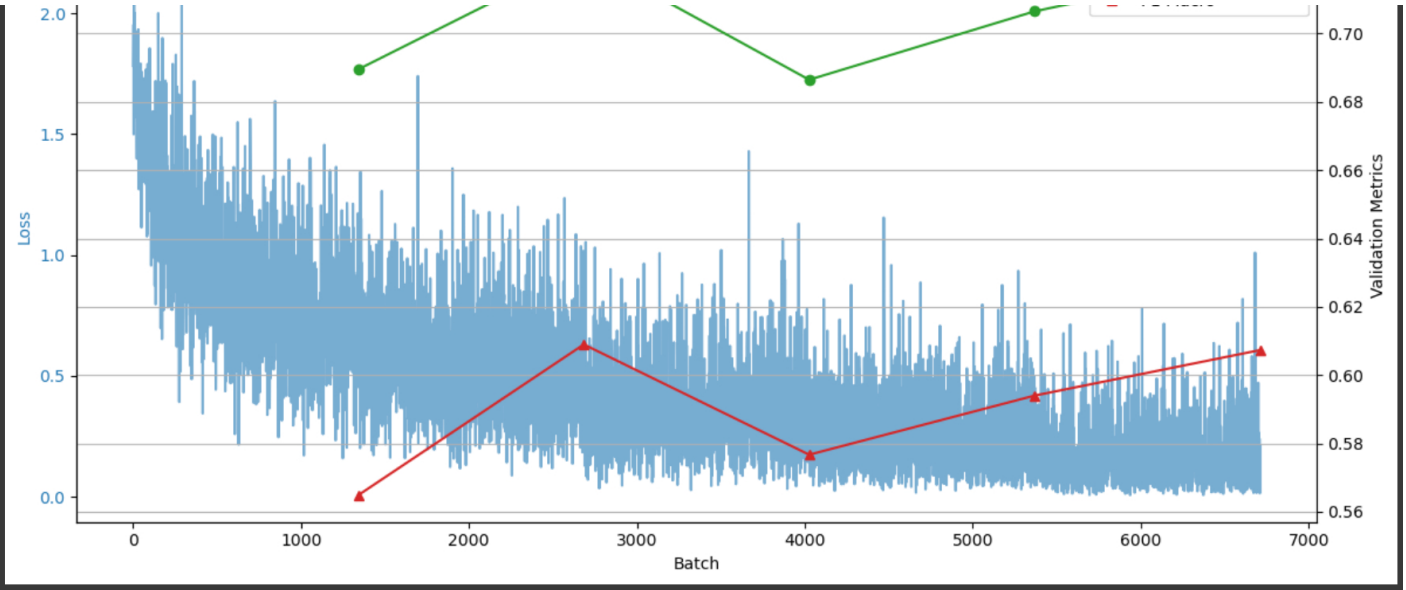
```
best_acc = 0
best_model = None
patience = 3
no_improve_epochs = 0
epoch = 0

# Training curve data
batch_loss_list = []
acc_per_epoch = []
f1_macro_per_epoch = []
batch_indices = []

batch_index = 0

while True:
    model.train()
    epoch += 1

    training = tqdm(train_loader, total=len(train_loader), leave=True, desc=f'Epoch {epoch}')

    for img, metadata, label in training:
```

```
            img = img.to(device)
            metadata = metadata.to(device)
            y_true = label.to(device, non_blocking=True)

            optimizer.zero_grad()
            y_pred = model(img, metadata)
            loss = criterion(y_pred, y_true)
            loss.backward()
            optimizer.step()

            # Record batch loss and batch index
            batch_index += 1
            batch_loss_list.append(loss.item())
            batch_indices.append(batch_index)

            training.set_postfix_str(f'Batch {batch_index} | loss={loss.item():.5f}')

        # Validation per epoch
        acc, f1_class, top2, f1_macro = Evaluate(model)
        acc_per_epoch.append(acc)
        f1_macro_per_epoch.append(f1_macro)

        print(f"Epoch {epoch} — Accuracy: {acc:.5f} | Top 2 Accuracy: {top2:.5f} | F1 Score (macro): {f1_macro:.5f}")
        print(" | ".join([f"{label_cols[i]}: {f1:.4f}" for i, f1 in enumerate(f1_class)]))

        # Early stopping
        if acc > best_acc:
            best_acc = acc
            best_model = model
            no_improve_epochs = 0
        else:
            no_improve_epochs += 1
            if no_improve_epochs >= patience:
                print(f"No improvement in accuracy for {patience} consecutive epochs. Stopping training.")
                break
```

```
Epoch 1: 100%|██████████| 1342/1342 [15:19<00:00,  1.46it/s, Batch 1342 | loss=0.45371]
Testing: 100%|██████████| 387/387 [02:44<00:00,  2.36it/s]
Epoch 1 — Accuracy: 0.68955 | Top 2 Accuracy: 0.85705 | F1 Score (macro): 0.56487
MEL: 0.6739 | NV: 0.8273 | BCC: 0.6974 | AK: 0.4677 | BKL: 0.4635 | DF: 0.5000 | VASC: 0.5778 | SCC: 0.3113
Epoch 2: 100%|██████████| 1342/1342 [15:40<00:00,  1.43it/s, Batch 2684 | loss=0.45298]
Testing: 100%|██████████| 387/387 [02:34<00:00,  2.51it/s]
Epoch 2 — Accuracy: 0.72266 | Top 2 Accuracy: 0.88112 | F1 Score (macro): 0.60897
MEL: 0.7038 | NV: 0.8497 | BCC: 0.7168 | AK: 0.4527 | BKL: 0.4881 | DF: 0.6585 | VASC: 0.6036 | SCC: 0.3986
Epoch 3: 100%|██████████| 1342/1342 [15:40<00:00,  1.43it/s, Batch 4026 | loss=0.19437]
Testing: 100%|██████████| 387/387 [02:41<00:00,  2.40it/s]
Epoch 3 — Accuracy: 0.68648 | Top 2 Accuracy: 0.87611 | F1 Score (macro): 0.57669
MEL: 0.6768 | NV: 0.8026 | BCC: 0.7250 | AK: 0.4703 | BKL: 0.3872 | DF: 0.5139 | VASC: 0.6154 | SCC: 0.4223
Epoch 4: 100%|██████████| 1342/1342 [15:51<00:00,  1.41it/s, Batch 5368 | loss=0.09198]
Testing: 100%|██████████| 387/387 [02:41<00:00,  2.39it/s]
Epoch 4 — Accuracy: 0.70651 | Top 2 Accuracy: 0.86900 | F1 Score (macro): 0.59400
MEL: 0.7109 | NV: 0.8307 | BCC: 0.7159 | AK: 0.4085 | BKL: 0.5057 | DF: 0.5868 | VASC: 0.6203 | SCC: 0.3731
Epoch 5: 100%|██████████| 1342/1342 [16:02<00:00,  1.39it/s, Batch 6710 | loss=0.21336]
Testing: 100%|██████████| 387/387 [02:41<00:00,  2.40it/s]Epoch 5 — Accuracy: 0.72056 | Top 2 Accuracy: 0.88128 | F1 Score (macro): 0.60731
MEL: 0.7097 | NV: 0.8404 | BCC: 0.7207 | AK: 0.4408 | BKL: 0.4901 | DF: 0.5957 | VASC: 0.6552 | SCC: 0.4059
No improvement in accuracy for 3 consecutive epochs. Stopping training.
```

```
[ ]  fig, ax1 = plt.subplots(figsize=(12, 6))

    # Batch-wise loss (left y-axis)
    ax1.plot(batch_indices, batch_loss_list, label='Batch Loss', color='tab:blue', alpha=0.6)
    ax1.set_xlabel('Batch')
    ax1.set_ylabel('Loss', color='tab:blue')
    ax1.tick_params(axis='y', labelcolor='tab:blue')

    # Epoch-wise accuracy and F1 (right y-axis)
    ax2 = ax1.twinx()
    ax2.plot([i * len(train_loader) for i in range(1, len(acc_per_epoch)+1)],
             acc_per_epoch, label='Validation Accuracy', color='tab:green', marker='o')
    ax2.plot([i * len(train_loader) for i in range(1, len(f1_macro_per_epoch)+1)],
             f1_macro_per_epoch, label='F1 Macro', color='tab:red', marker='^')
    ax2.set_ylabel('Validation Metrics', color='black')
    ax2.tick_params(axis='y', labelcolor='black')

    # Combine legends from both axes
    lines, labels = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax2.legend(lines + lines2, labels + labels2, loc='upper right')

    plt.title('Training Progress: Batch Loss + Epoch Accuracy & F1')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```






```
[ ]  torch.save(best_model, "fusion_model.pth")
```

Result Analysis of Fusion Model

```
[ ]  def result(model, test_loader, class_names=None, plot=True):
         model.eval()
         y_pred_test = []
         y_true_test = []

         with torch.no_grad():
             testing = tqdm(test_loader, total=len(test_loader), leave=True, desc="Testing")
             for img, metadata, label in testing:
                 img = img.to(device)
                 label = label.to(device)
                 metadata = metadata.to(device)

                 output = model(img, metadata)
                 preds = torch.argmax(output, dim=1).cpu().numpy()

                 y_pred_test.extend(preds)
                 y_true_test.extend(label.cpu().numpy())

         acc = accuracy_score(y_true_test, y_pred_test)
         f1 = f1_score(y_true_test, y_pred_test, average="macro")
         cm = confusion_matrix(y_true_test, y_pred_test)

         if plot:
             plt.figure(figsize=(8, 6))
             sns.heatmap(
                 cm, annot=True, fmt="d", cmap="Blues", cbar=False,
                 xticklabels=class_names if class_names else True,
                 yticklabels=class_names if class_names else True
             )
             plt.xlabel("Predicted")
             plt.ylabel("Actual")
             plt.title("Confusion Matrix")
             plt.show()

         return acc, f1, cm
```

```
[ ]  model = torch.load('fusion_model.pth', weights_only=False)
     model.eval()
     acc, f1, cm = result(model, test_loader, class_names=None, plot=True)
```

```
Testing: 100%|██████████| 387/387 [02:34<00:00,  2.50it/s]
```