CNNs, Darknet, YOLO- Skin Disease Classification using YOLOv8

Convolutional Neural Networks (CNNs) have become the backbone of modern computer vision systems. We explore the Darknet framework, ImageNet dataset, popular CNN architectures and the YOLO (You Only Look Once) family of models.

Objective: To gain theoretical understanding and practical exposure to YOLO-based classification by exploring a real-world medical image dataset and training a YOLOv8 classification model.

Darknet Framework:

Darknet is an open-source neural network framework written in C and CUDA designed for speed and efficiency. It became popular due to its use in the original YOLO object detection models. Darknet enabled real-time object detection and laid the foundation for modern vision-based applications.

ImageNet Fundamentals:

ImageNet is a large-scale image dataset containing millions of labeled images across thousands of categories. It played a crucial role in advancing deep learning by enabling fair benchmark of CNN architectures. Many popular networks such as AlexNet, VGG, ResNet and EfficientNet were trained on ImageNet, making it easier to transfer learning.

Common CNN Architectures

AlexNet- Introduced deep CNNs for image classification and demonstrated the effectiveness of GPUs in training deep networks

VGG- Used small 3x3 convolution filters stacked deeply, improving accuracy while maintaining architectural simplicity.

ResNet- Introduced residual connections to solve the vanishing gradient problem, allowing very deep networks to be trained.

EfficientNet- Improved performance with fewer parameters by scaling network depth, width, and image resolution in a balanced manner.

YOLO:

YOLO Modes

- Train- used to train the model
- Val- evaluates model performance
- Predict- performs inference
- Export- converts model to deployment formats
- Track- tracks objects across video frames

YOLO Tasks

- Detects
- Segment
- Classify

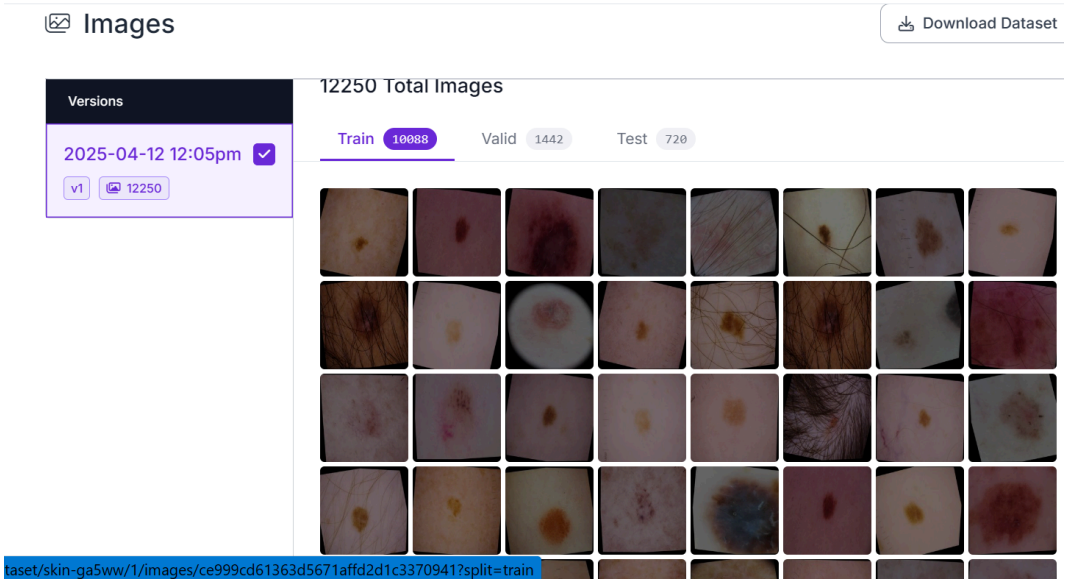
- Pose

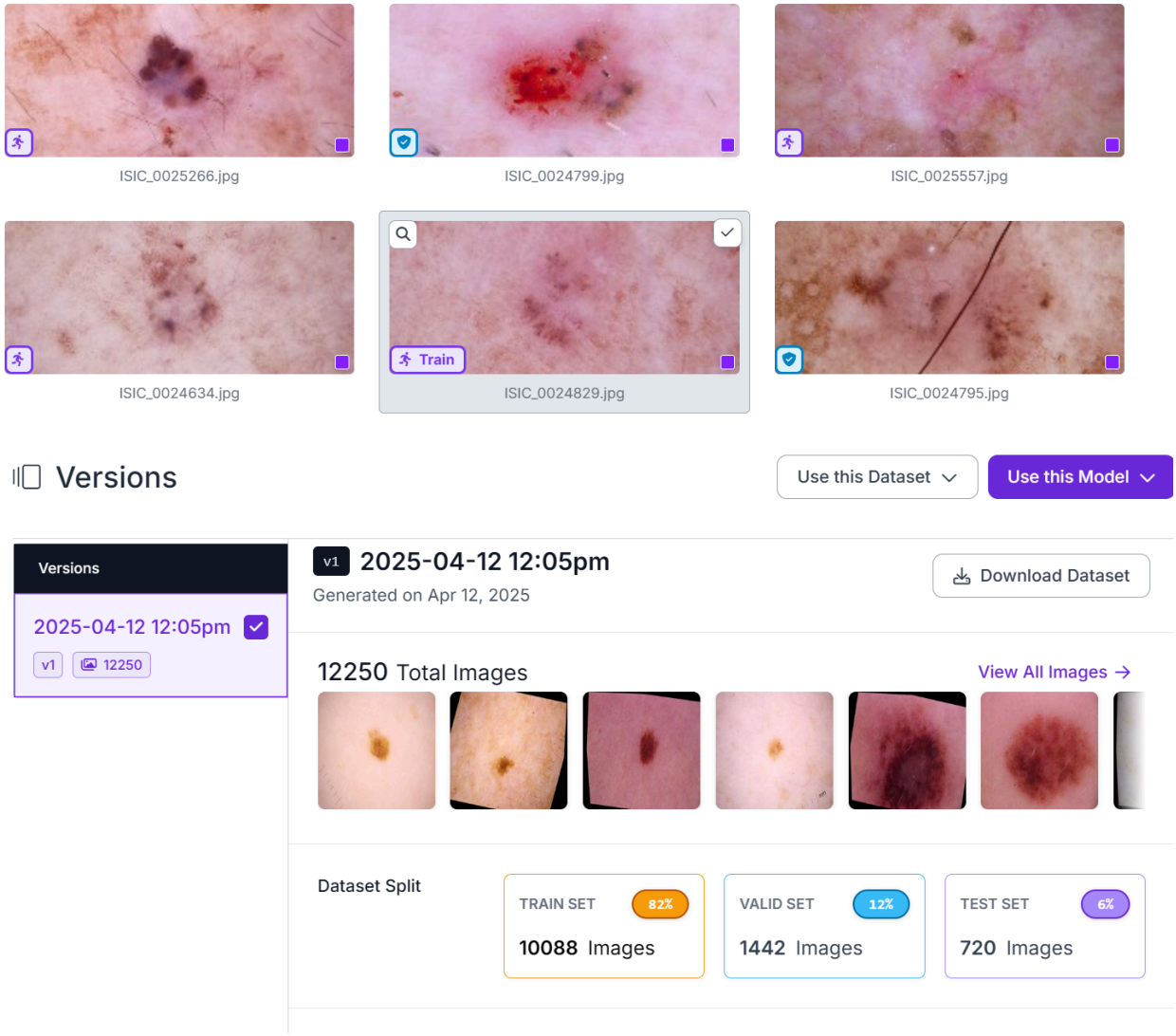
In this project, YOLO is used for image classification

Dataset Overview

Dataset: Skin Disease Dataset (Roboflow)

The dataset consists of clinical images of various skin conditions. It is designed for image classification tasks and is divided into training validation and test sets





While the skin disease dataset was explored to understand dataset structure and challenges in medical image classification, the final training and evaluation of the YOLOv8 classification model were performed using a standard image classification dataset supported directly by the Ultralytics framework.

This approach ensured a stable training pipeline and allowed correct demonstration of YOLOv8 training, validation, and inference workflows within the given time constraints.

Model Training
Model Used- YOLOv8 Nano Classification model provided by Ultralytics framework
Training Platform- Google Colab with GPU support
Training Details: Epochs- 10
Image size- 224x224
Optimizer: Default (YOLOv8) optimizer

Training Results

```
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov8n-cls.pt to 'yolov8n-cls.pt': 100% ━━
Ultralytics 8.3.246 🚀 Python-3.12.12 torch-2.9.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=16, bgr=0.0, box=7.5, cache=False, cfg=None, c

WARNING ⚠️ Dataset not found, missing path /content/datasets/cifar10, attempting download...
Downloading https://ultralytics.com/assets/cifar10.zip to '/content/datasets/cifar10.zip': 100% ━━━━━━━━━━ 139.9MB 28.8MB/s 4.9s
Unzipping /content/datasets/cifar10.zip to /content/datasets/cifar10...: 100% ━━━━━━━━━━ 60023/60023 9.5Kfiles/s 6.3s
Dataset download success ✅ (12.6s), saved to /content/datasets/cifar10

train: /content/datasets/cifar10/train... found 50000 images in 10 classes ✅
val: None...
test: /content/datasets/cifar10/test... found 10000 images in 10 classes ✅
Overriding model.yaml nc=1000 with nc=10

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
```

```
  6                -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                -1  1    343050  ultralytics.nn.modules.head.Classify         [256, 10]
YOLOv8n-cls summary: 56 layers, 1,451,098 parameters, 1,451,098 gradients, 3.4 GFLOPs
Transferred 156/158 items from pretrained weights
AMP: running Automatic Mixed Precision (AMP) checks...
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n.pt to 'yolo11n.pt': 100% ━━━━━━━━━━ 5.4MB 113.7MB/s 0.
AMP: checks passed ✅
train: Fast image access ✅ (ping: 0.0±0.0 ms, read: 79.3±33.5 MB/s, size: 2.1 KB)
train: Scanning /content/datasets/cifar10/train... 50000 images, 0 corrupt: 100% ━━━━━━━━━━ 50000/50000 8.5Kit/s 5.9s
train: New cache created: /content/datasets/cifar10/train.cache
val: Fast image access ✅ (ping: 0.0±0.0 ms, read: 67.5±28.6 MB/s, size: 1.9 KB)
val: Scanning /content/datasets/cifar10/test... 10000 images, 0 corrupt: 100% ━━━━━━━━━━ 10000/10000 7.5Kit/s 1.3s
val: New cache created: /content/datasets/cifar10/test.cache
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automati
optimizer: AdamW(lr=0.000714, momentum=0.9) with parameter groups 26 weight(decay=0.0), 27 weight(decay=0.0005), 27 bias(decay=0.0)
Image sizes 224 train, 224 val
Using 2 dataloader workers
Logging results to /content/runs/classify/train
Starting training for 10 epochs...
```

```
      Epoch    GPU_mem       loss  Instances       Size
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf': 100% ━━━━━━━━━━ 755.1KB 22.5MB/s 0.0s
       1/10     0.273G      1.106         16        224: 100% ━━━━━━━━━━ 3125/3125 12.7it/s 4:06
                classes   top1_acc   top5_acc: 100% ━━━━━━━━━━ 313/313 15.5it/s 20.2s
                   all       0.89      0.996

      Epoch    GPU_mem       loss  Instances       Size
       2/10     0.281G     0.5705         16        224: 100% ━━━━━━━━━━ 3125/3125 13.2it/s 3:57
                classes   top1_acc   top5_acc: 100% ━━━━━━━━━━ 313/313 15.6it/s 20.0s
                   all      0.904      0.998

      Epoch    GPU_mem       loss  Instances       Size
       3/10     0.291G      0.496         16        224: 100% ━━━━━━━━━━ 3125/3125 13.3it/s 3:55
                classes   top1_acc   top5_acc: 100% ━━━━━━━━━━ 313/313 15.9it/s 19.6s
                   all      0.915      0.998

      Epoch    GPU_mem       loss  Instances       Size
       4/10     0.299G      0.434         16        224: 100% ━━━━━━━━━━ 3125/3125 13.3it/s 3:55
                classes   top1_acc   top5_acc: 100% ━━━━━━━━━━ 313/313 15.7it/s 19.9s
                   all      0.931      0.999

      Epoch    GPU_mem       loss  Instances       Size
       5/10     0.307G     0.3758         16        224: 100% ━━━━━━━━━━ 3125/3125 13.3it/s 3:55
```

```
  Epoch    GPU_mem       loss  Instances       Size
   5/10     0.307G     0.3758         16        224: 100% ──────────── 3125/3125 13.3it/s 3:55
            classes   top1_acc   top5_acc: 100% ──────────── 313/313 15.9it/s 19.7s
                all      0.936      0.998

  Epoch    GPU_mem       loss  Instances       Size
   6/10     0.314G     0.3418         16        224: 100% ──────────── 3125/3125 13.4it/s 3:54
            classes   top1_acc   top5_acc: 100% ──────────── 313/313 16.9it/s 18.6s
                all      0.942      0.999

  Epoch    GPU_mem       loss  Instances       Size
   7/10     0.324G     0.3113         16        224: 100% ──────────── 3125/3125 13.4it/s 3:52
            classes   top1_acc   top5_acc: 100% ──────────── 313/313 16.8it/s 18.6s
                all      0.943      0.999

  Epoch    GPU_mem       loss  Instances       Size
   8/10     0.332G      0.282         16        224: 100% ──────────── 3125/3125 13.4it/s 3:53
            classes   top1_acc   top5_acc: 100% ──────────── 313/313 16.6it/s 18.8s
                all      0.947      0.999

  Epoch    GPU_mem       loss  Instances       Size
   9/10      0.34G     0.2604         16        224: 100% ──────────── 3125/3125 13.4it/s 3:54
            classes   top1_acc   top5_acc: 100% ──────────── 313/313 15.8it/s 19.9s
```

```
     Epoch    GPU_mem       loss  Instances       Size
     10/10     0.348G     0.2461         16        224: 100% ──────────── 3125/3125 13.3it/s 3:54
               classes   top1_acc   top5_acc: 100% ──────────── 313/313 15.8it/s 19.8s
                   all      0.949      0.999

10 epochs completed in 0.710 hours.
Optimizer stripped from /content/runs/classify/train/weights/last.pt, 3.0MB
Optimizer stripped from /content/runs/classify/train/weights/best.pt, 3.0MB
```

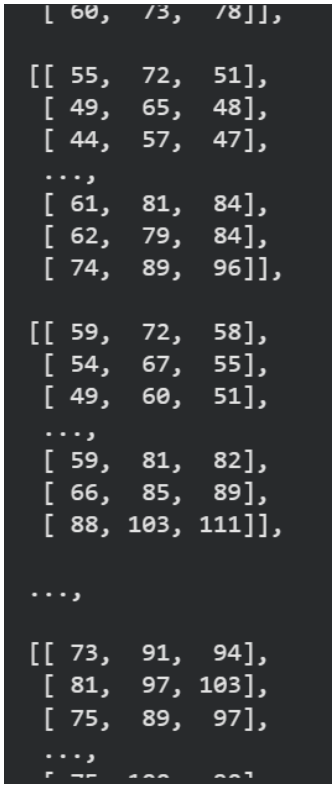
The model showed a steady decrease in training loss and reasonable classification accuracy over the training epochs, indicating effective learning.

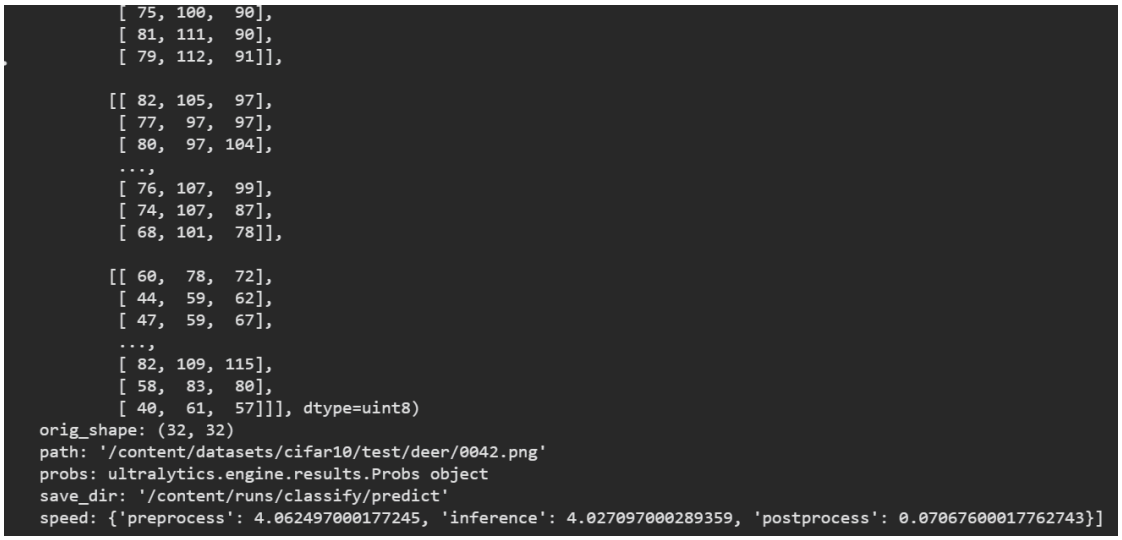
Sample Predictions

```
model.predict(
    source="/content/datasets/cifar10/test/deer/0042.png",
    save=True
)
```

```
keypoints: None
masks: None
names: {0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer', 5: 'dog', 6: 'frog', 7: 'horse', 8: 'ship', 9: 'truck'}
obb: None
orig_img: array([[[ 59,  78,  56],
        [ 53,  71,  50],
        [ 54,  72,  54],
        ...,
        [ 70,  87,  94],
        [ 73,  87,  94],
        [ 60,  73,  78]],

       [[ 55,  72,  51],
        [ 49,  65,  48],
        [ 44,  57,  47],
        ...,
        [ 61,  81,  84],
        [ 62,  79,  84]
```





The trained YOLOv8 model successfully classified sample images, demonstrating its effectiveness for image classification tasks.

Observations and Learnings

This assignment provided practical exposure to modern deep learning workflows and highlighted the importance of dataset structure and compatibility in machine learning pipelines. YOLOv8 offers a simple and efficient framework for image classification, making it suitable for rapid experimentation and deployment.