Supplementary material

to Poultry Perfection - Comparison of Computer Vision Models to Detect and Classify Poultry Products in a Production Setting

Applied Food Research

```json
{
  "images": [
    {
      "id": 1,
      "file_name": "example.jpg",
      "width": 640,
      "height": 640
    }
  ],
  "annotations": [
    {
      "id": 1,
      "image_id": 1,
      "category_id": 1,
      "bbox": [100, 200, 50, 60],
      "iscrowd": 0
    }
  ],
  "categories": [
    {
      "id": 1,
      "name": "imperfect product"
    }
  ]
}
```

Listing 1: Structure of a `JSON` file in COCO format. The file contains the image name plus a unique ID for the image, plus other information such as the image's dimensions, the class ID, and all the bounding box information. One COCO label file usually contains the information for all images in the dataset.

```
# The first number
    represents the class
    ID. The rest is bbox
    information
1 0.5 0.5 0.1 0.2
0 0.8 0.5 0.1 0.2
```

Listing 2: This label file depicts the structure of the YOLO label files. One such file exists for each image. The `txt` file contains the class ID and the bounding box information. It is essential that the name of the label file matches with the corresponding image name.

Figure A1: Label Studio interface with a labeled picture.

```python
# Define transformations
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    A.ToGray(p=0.2),
    A.ChannelShuffle(p=0.2),
], bbox_params=A.BboxParams(format='yolo', label_fields=['
    class_labels'], min_area=0.1, min_visibility=0.1))
```

Listing 3: This code snippet from the augmentation demonstrates which transformations are performed during the augmentation process. In this case, a horizontal flip is applied, the brightness is adjusted randomly, the images are changed to gray, and the channels are shuffled. To introduce randomness, probabilities were applied, which dictate the chance of a certain transformation being applied.

```
#Option 1: Traing from scratch
model = YOLO('yolov8x.yaml')
#Option 2: Training with pretrained weights
model = YOLO('yolov8x.pt')

# Training command
results = model.train(
    data=path/to/data.yaml,
    batch=16, #Set batch-size
    epochs=100, #Set epochs
    imgsz=640 #Set image-size
)
```

Listing 4: This listing shows how training is initiated for YOLOv8x. The model can either be trained from scratch or with pretrained weights. Additional parameters can be specified in the training command.

```python
# Start timer
start = datetime.now()

# Define model and data path
model = YOLO("yolo12x.pt")
data_path = r"/home/foodinformatics/Projects/Daniel_KI-Workingstation/YOLO/Yolo_Format/data.yaml"

# Run hyperparameter tuning
results = model.tune(
    data=data_path,
    epochs=100,
    iterations=100,
    optimizer="AdamW",
    plots=False,
    save=False,
    val=False
)

# End timer
end = datetime.now()
duration = end - start

# Write runtime info to file
runtime_str = f"Total tuning runtime: {duration}\nStarted at: {start}\nEnded at: {end}\n"
with open('tuning_runtime_log.txt', 'w') as f:
    f.write(runtime_str)
```

Listing 5: This listing shows how tuning is initiated for YOLOv12x.