Chanatip Kowsurat 6530075721

Notebook #1 Image classification with CNN

Input: 32x32 RGB images in 10 classes from the CIFAR-10 dataset
Output: Predicted class probabilities or labels for the 10 categories.

Hardware requirement: NVIDIA GPUs

Process

1. Preprocess
   1.1. Install and import libraries
   1.2. Load datasets from CIFAR-10
   1.3. Apply normalization and data augmentation

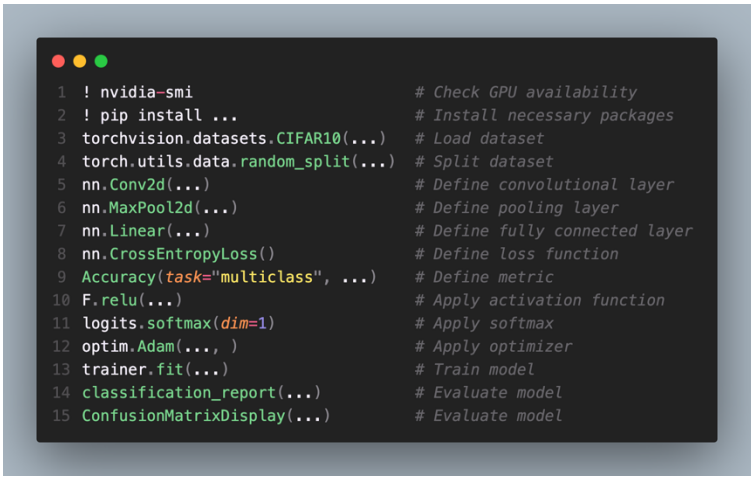
2. Create CNN model
   2.1. Pipeline
   - First convolutional layer has 3 input and 6 output channels with 5x5 kernel.
   - Pooling layer uses a 2x2 kernel with a stride of 2.
   - Second convolutional layer has 6 input and 16 output channels with 5x5 kernel.
   - First fully connected layer takes 400 input and outputs 120 units.
   - Second fully connected layer takes 120 input units and outputs 84 units.
   - Third fully connected layer takes 84 input units and outputs 10 units

   2.2. Summary
   - Activation function: ReLU, Softmax
   - Loss function: Cross-Entropy Loss
   - Optimizer: Adam
   - Total params: 62,006

3. Training
   - Batch size: 32
   - Epoch: 1
   - 60,000 images split into: 50,000 training images and 10,000 test images
   - Each class contains an equal number of images.

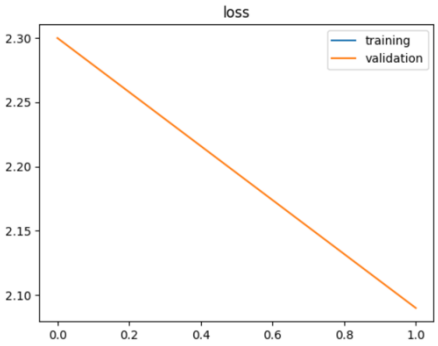
4. Evaluate the model
   - Accuracy, Confusion Matrix

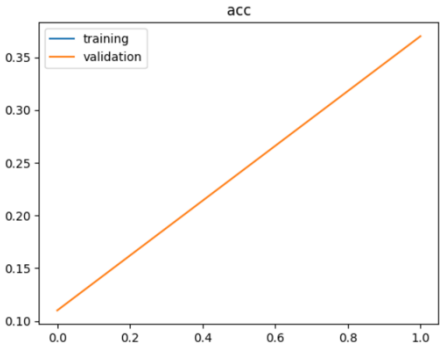
Important commands



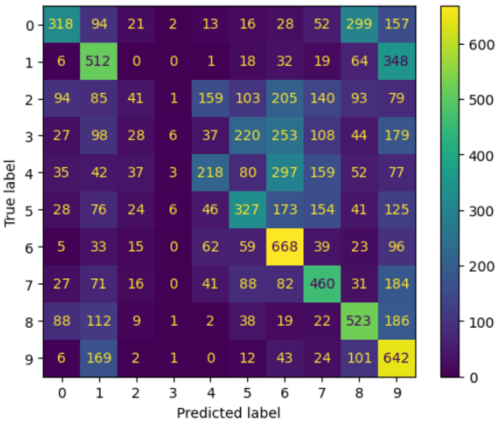
```
! nvidia-smi                          # Check GPU availability
! pip install ...                     # Install necessary packages
torchvision.datasets.CIFAR10(...)     # Load dataset
torch.utils.data.random_split(...)    # Split dataset
nn.Conv2d(...)                        # Define convolutional layer
nn.MaxPool2d(...)                     # Define pooling layer
nn.Linear(...)                        # Define fully connected layer
nn.CrossEntropyLoss()                 # Define loss function
Accuracy(task="multiclass", ...)      # Define metric
F.relu(...)                           # Apply activation function
logits.softmax(dim=1)                 # Apply softmax
optim.Adam(..., )                     # Apply optimizer
trainer.fit(...)                      # Train model
classification_report(...)            # Evaluate model
ConfusionMatrixDisplay(...)           # Evaluate model
```

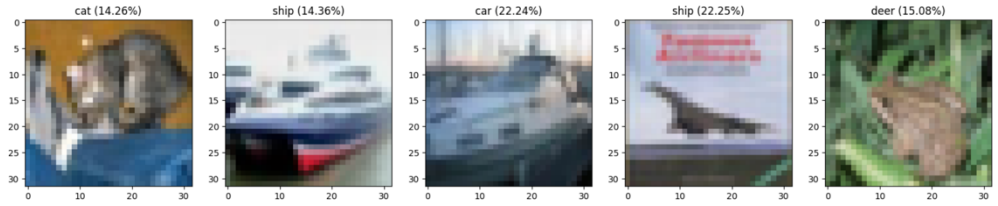
Results






```
testing ...
100% 313/313 [00:01<00:00, 174.13it/s]
testing loss: 2.085
              precision    recall  f1-score   support

           0     0.5016    0.3180    0.3892      1000
           1     0.3963    0.5120    0.4468      1000
           2     0.2124    0.0410    0.0687      1000
           3     0.3000    0.0060    0.0118      1000
           4     0.3765    0.2180    0.2761      1000
           5     0.3403    0.3270    0.3335      1000
           6     0.3711    0.6680    0.4771      1000
           7     0.3908    0.4600    0.4226      1000
           8     0.4115    0.5230    0.4606      1000
           9     0.3097    0.6420    0.4178      1000

    accuracy                         0.3715     10000
   macro avg     0.3610    0.3715    0.3304     10000
weighted avg     0.3610    0.3715    0.3304     10000
```





Notebook #2 Image classification with EfficientNet

Input: Image dataset with 10 animal classes
Output: Predicted class labels

Hardware requirement: NVIDIA GPUs

Process

1. Preprocess
   1.1. Install and import libraries
   1.2. Load datasets from GitHub repository
   1.3. Image augmentation ex. Rotation, Horizontal and Vertical flip
   1.4. Normalize images
   1.5. Split data into 1,400 training images, 300 validation images and 300 test images

2. Training Setup
   - Model: EficientNetV2
   - Loss function: Cross-Entropy function
   - Optimizer: Adam
   - Learning rate: 0.001
   - Batch size: 32
   - Epoch: 5
   - Trainable parameters: 20.2 M

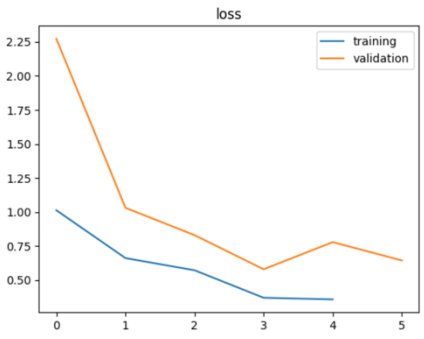
3. Training

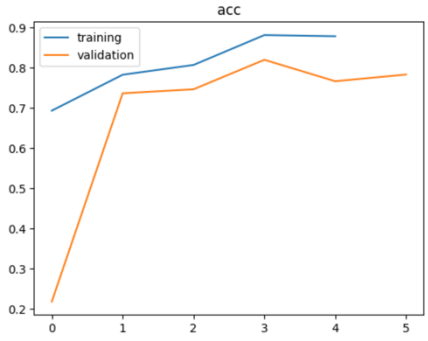
4. Evaluate the model
   - Accuracy, Confusion Matrix

Important Commands

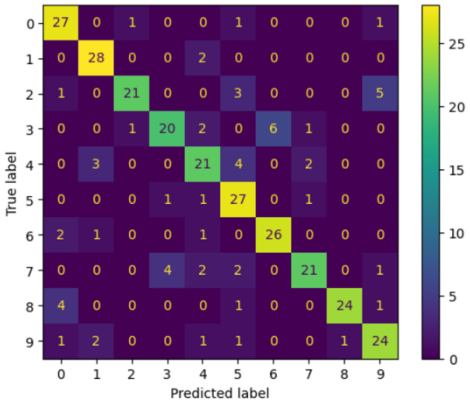
```
! nvidia-smi                          # Check GPU availability
! pip install ...                     # Install necessary packages
! wget <...>                          # Download dataset
torchvision.transforms.Compose([...]) # Apply transformations
models.efficientnet_v2_s(...)         # Load model
nn.CrossEntropyLoss()                 # Define loss function
Accuracy(task="multiclass", ...)      # Define metric
logits.softmax(...)                   # Apply softmax
torch.optim.Adam(...)                 # Apply optimizer
trainer.fit(model, ...)               # Train model
trainer.test(model, ...)              # Evaluate model
```

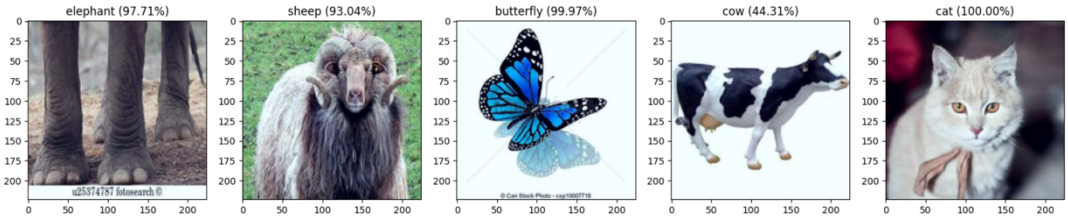
Results






```
testing ...
100%|████████████████████| 10/10 [00:53<00:00,  4.30s/it]
testing loss: 0.7089
              precision    recall  f1-score   support

           0     0.7714    0.9000    0.8308        30
           1     0.8235    0.9333    0.8750        30
           2     0.9130    0.7000    0.7925        30
           3     0.8000    0.6667    0.7273        30
           4     0.7000    0.7000    0.7000        30
           5     0.6923    0.9000    0.7826        30
           6     0.8125    0.8667    0.8387        30
           7     0.8400    0.7000    0.7636        30
           8     0.9600    0.8000    0.8727        30
           9     0.7500    0.8000    0.7742        30

    accuracy                         0.7967       300
   macro avg     0.8063    0.7967    0.7957       300
weighted avg     0.8063    0.7967    0.7957       300
```





Notebook #3 Object detection with yolov8 (basic)

Input: Images in 20 classes from the Pascal-VOC dataset.
Output: Predicted class labels

Hardware requirement: NVIDIA GPUs

Process:

1. Preprocess
   1.1. Import packages from Ultralytics
   1.2. Checks software and hardware configuration

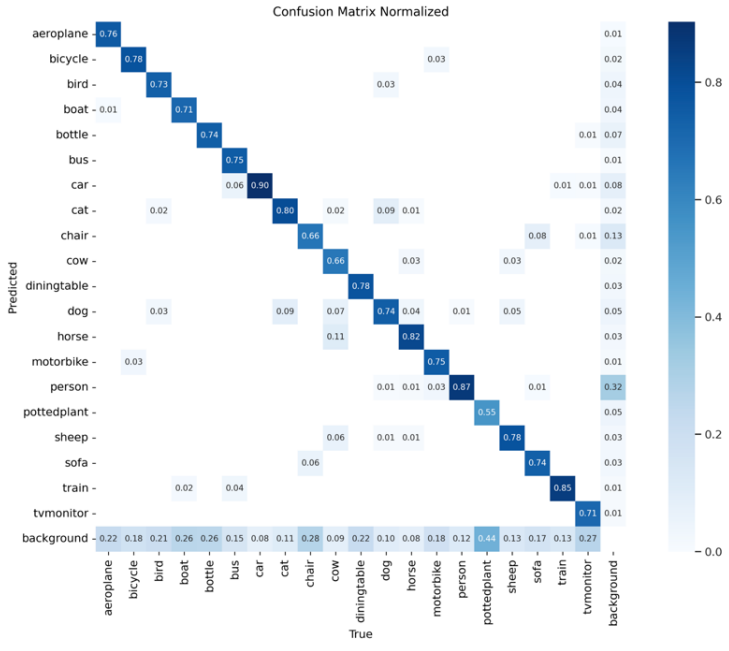
2. Train
   - Model: YOLOv8
   - Images of a size 640x640 with 20 classes
   - Epoch: 3
   - Learning rate: 0.0004
   - Optimizer: Adam
   - Parameters: 3 M

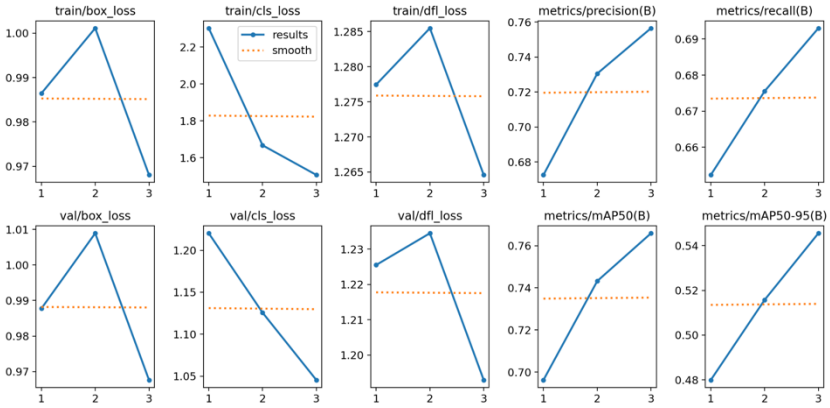
3. Evaluate model

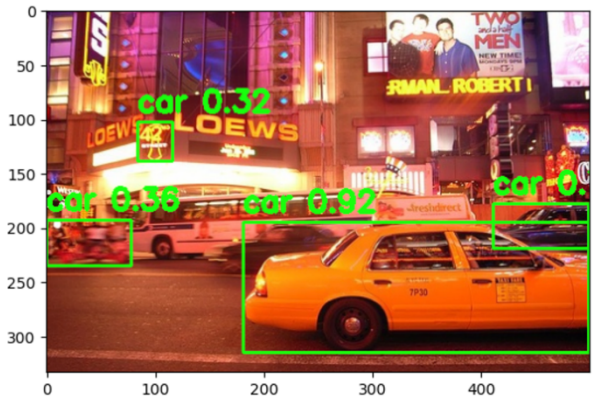
Important commands

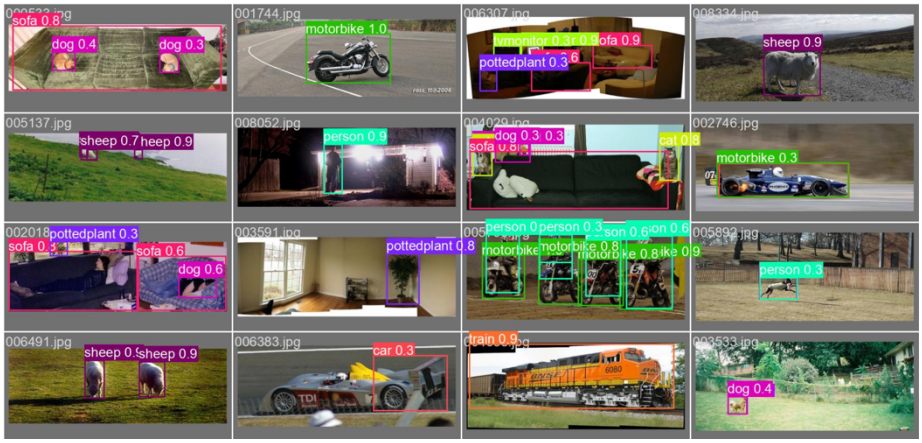
```
%pip install ultralytics                                          # Install Ultralytics
!nvidia-smi                                                       # Check GPU availability

# For CLI usage
!yolo train data=VOC.yaml model=yolov8n.pt epochs=3 imgsz=640 device=0  # Train model
!yolo predict model=yolov8n.pt source='image.jpg' device=0        # Predict image

# For Python usage
model = YOLO(...)                                                 # load a pretrained model
model.train(...)                                                  # Train model
model.val()                                                       # Validate model
model.predict(...)                                                # Predict image
```

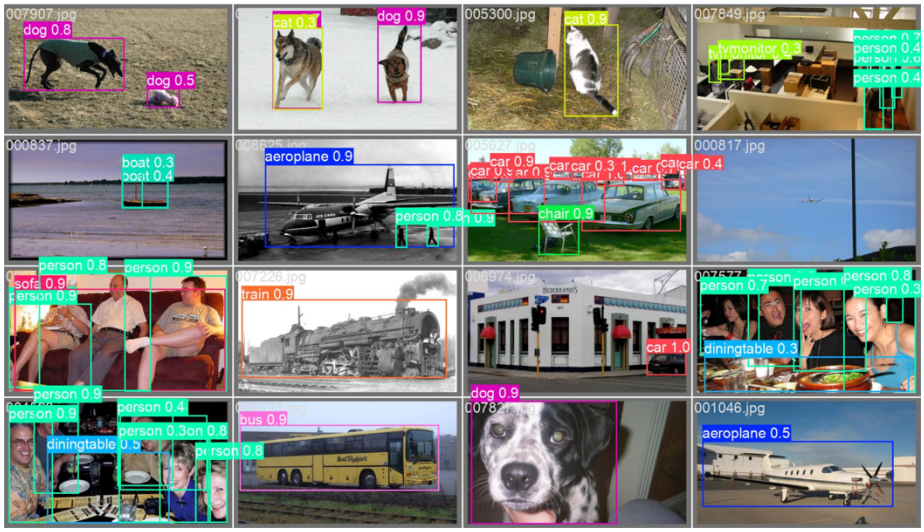
Results




train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)


car 0.32
car 0.36
car 0.92


sofa 0.8
dog 0.4
dog 0.3
motorbike 1.0
tvmonitor 0.3
sofa 0.9
pottedplant 0.3
sheep 0.9
sheep 0.7
person 0.9
dog 0.3
cat 0.8
motorbike 0.3
pottedplant 0.3
sofa 0.6
dog 0.6
pottedplant 0.8
person 0.3
sheep 0.9
car 0.3
train 0.9
dog 0.4


dog 0.8
dog 0.5
cat 0.3
dog 0.9
cat 0.9
tvmonitor 0.3
person 0.4
boat 0.3
aeroplane 0.9
person 0.8
car 0.9
chair 0.9
person 0.9
train 0.9
car 1.0
diningtable 0.3
bus 0.9
aeroplane 0.5

Notebook #4 Object detection with yolov8 (advanced)

Input: Pascal VOC datasets of 20 object classes
Output: Trained YOLOv8 object detection model with bounding box predictions

Hardware requirement: NVIDIA GPUs

Process:

1. Preprocess
   1.1. Install and import packages
   1.2. Download and extract Pascal VOC dataset
   1.3. Define .yml file structure for YOLO
   1.4. Convert VOC XML annotations to YOLO format

2. Setup model
   - Model: YOLOv8
   - Epoch: 3
   - Learning rate: 0.0004
   - Optimizer: Adam
   - Parameters: 3 M
   - 10,000 images split into 8,000 training images and 2,000 test images

3. Train
   3.1. Load YOLOv8 model
   3.2. Train with dataset and monitor loss

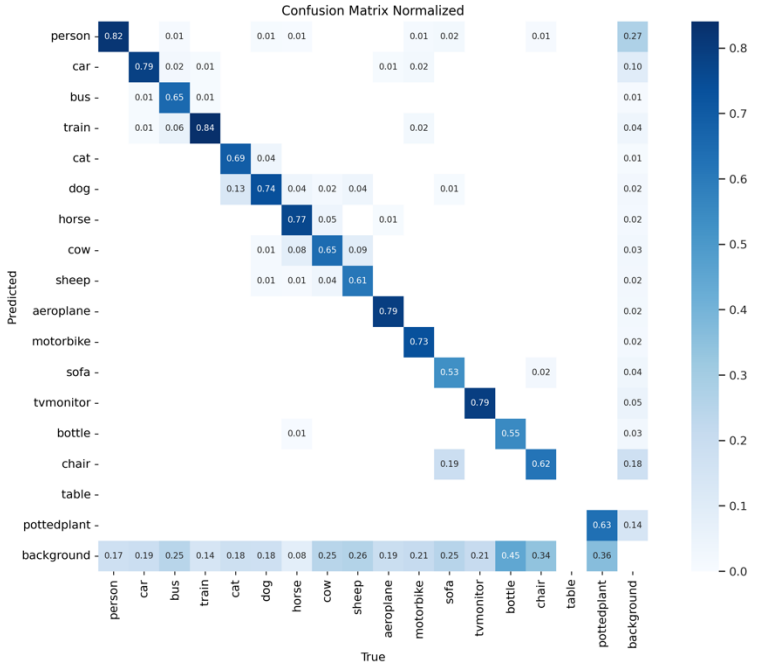
4. Evaluate
   4.1. Validate performance with mAP

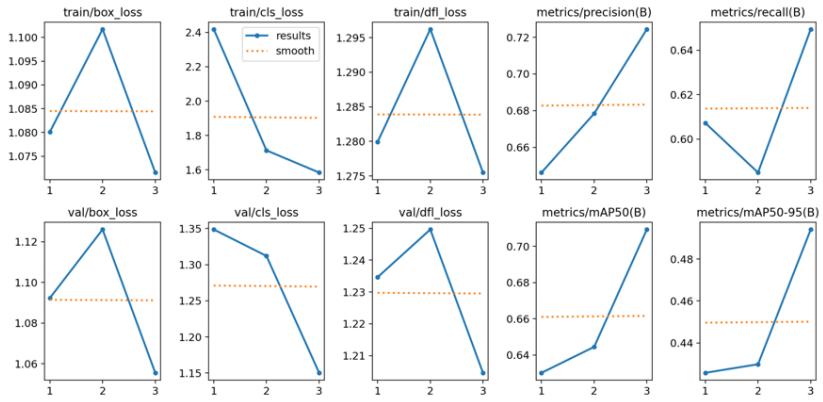
5. Test
   5.1. Test detection on sample images

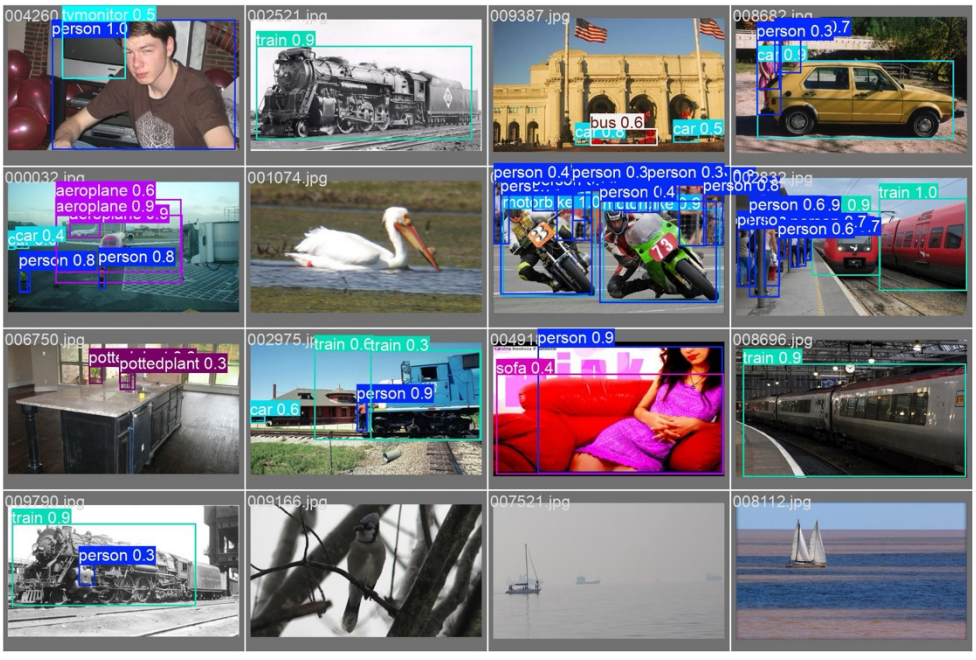
Important commands

```
%pip install ...          # Install necessary packages
ultralytics.checks()      # Check dependencies
download_file(...)        # Download dataset
extract_tar(...)          # Extract dataset
train_test_split(...)     # Split dataset
model = YOLO(...)         # Load model
model.train(...)          # Train model
results = model(...)      # Predict image
```

Results


Confusion Matrix Normalized
Predicted
True
person
car
bus
train
cat
dog
horse
cow
sheep
aeroplane
motorbike
sofa
tvmonitor
bottle
chair
table
pottedplant
background


train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)


004260.jpg tvmonitor 0.5 person 1.0
002521.jpg train 0.9
009387.jpg bus 0.6 car 0.5
008682.jpg person 0.3
000032.jpg aeroplane 0.6 aeroplane 0.9 car 0.4 person 0.8 person 0.8
001074.jpg
person 0.4 person 0.3
002832.jpg train 1.0 person 0.6 person 0.6
006750.jpg pottedplant 0.3
002975.jpg train 0.6 train 0.3 car 0.6 person 0.9
sofa 0.4
008696.jpg train 0.9
009790.jpg train 0.9 person 0.3
009166.jpg
007521.jpg
008112.jpg

Notebook #5 Semantic segmentation with DeepLabV3

Input: Image from CamSeq dataset with a size of 960x720
Output: Segmentation mask

Hardware requirement: NVIDIA GPUs

Process:

1. Preprocess
   1.1. Install and import packages
   1.2. Download 100 image datasets from CamSeq
   1.3. Load colormap from label_color.txt
   1.4. Create Data loader
   1.5. Apply transformations (resize, normalize).
   1.6. Split 101 images into 70 training images, 15 validation images and 16 test images

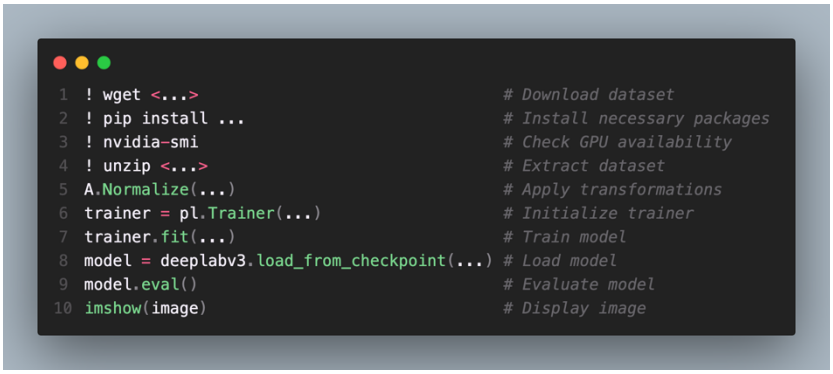
2. Setup model
   - Model: DeepLabV3
   - Trainable parameters: 42 M
   - Epoch: 2
   - Training Metrics: Cross-Entropy, accuracy and IoU.

3. Train model

4. Evaluate model
   - Compute IoU

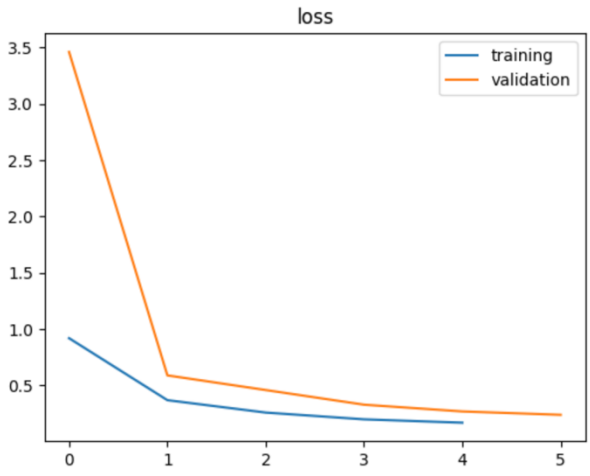
5. Visualize results
   - Test model with sample image

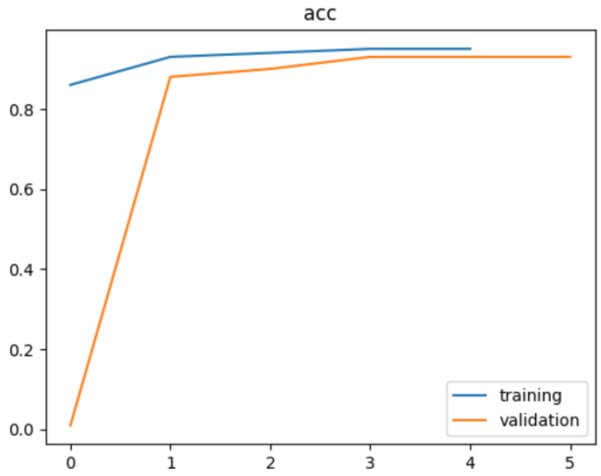
Important commands



```
! wget <...>                                  # Download dataset
! pip install ...                             # Install necessary packages
! nvidia-smi                                  # Check GPU availability
! unzip <...>                                 # Extract dataset
A.Normalize(...)                              # Apply transformations
trainer = pl.Trainer(...)                     # Initialize trainer
trainer.fit(...)                              # Train model
model = deeplabv3.load_from_checkpoint(...)   # Load model
model.eval()                                  # Evaluate model
imshow(image)                                 # Display image
```

Results:






```
100% ████████████████████ 8/8 [00:16<00:00,  2.08s/it]
average_iou_per_class: tensor([   nan, 0.0000, 0.6595,    nan, 0.7270, 0.0045,    nan, 0.3903, 0.0297,
        0.7489, 0.0631,    nan, 0.0000,    nan, 0.1779,    nan, 0.0212, 0.8938,
           nan, 0.3790, 0.0000, 0.7314,    nan,    nan, 0.4691,    nan, 0.5835,
        0.4388,    nan, 0.0012, 0.0679, 0.5700])
overall_average_iou: 0.33127713203430176
```