Topic 2 - Face Detection

Group 57

Jun 27, 2024

Group Members

- NGUYEN DUY THAI - 175906
- PHAM HOANG DUY - 220607
- NGUYEN HOANG VINH QUANG - 219130
- PHAN VAN TAN - 219200
- TRUONG MINH NGHIA - 164626

Overview

- Introduction to Face Detection
- Proposed Solution
- Architecture Model Evaluation
- Architecture
- Data Processing
- Dataset
- Environment Setup
- Evaluation Metrics
- Training Custom YOLOv10-L Detector
- Benchmark
- Experimental Results
- Comparison with Other Models
- Test Environment
- Test Results: Inference on Image
- Test Results: Inference on Video

Introduction to Face Detection

- **Definition**: Technology to identify and locate human faces in images and videos.
- **Importance**:
  - Foundation for facial recognition, emotion detection, and security systems.
  - Crucial in various applications like surveillance, user authentication, and personalized marketing.
- **How it Works**:
  - Uses algorithms and machine learning techniques.
  - Detects facial features and distinguishes them from other objects.
- **Advancements**:
  - Deep learning has enhanced accuracy and efficiency.
  - Modern systems are more robust and reliable.

Architecture Model Evaluation

Model	Pros	Cons
VGG16	- Simplicity - Strong Feature Extraction	- Computationally Intensive - Not Specialized for Detectio
ResNet50	- Residual Connections - High Accuracy - Scalability	- Complexity - Resource Intensive
YOLO	- Real-Time Performance - High Accuracy - Unified Architecture	- Complexity - Resource Intensive

Architecture Model Evaluation

Overview

VGG16, ResNet50, and YOLO are popular deep learning models used for image detection tasks, but they have different architectures, strengths, and use cases. Here is a comprehensive comparison of these models:

Architecture Model Evaluation

VGG16

Architecture

- **Type:** Convolutional Neural Network (CNN)
- **Depth:** 16 layers (13 convolutional layers and 3 fully connected layers)
- **Structure:** Sequential, uses small (3x3) convolution filters, followed by max-pooling layers

Pros

- **Simplicity:** Easy to implement and understand, making it a good starting point for beginners.
- **Strong Feature Extraction:** Effective at extracting features for image classification tasks due to its deep structure.

Architecture Model Evaluation

VGG16

Cons

- **Computationally Intensive:** Requires a lot of memory and computational power, especially for the fully connected layers.
- **Not Specialized for Detection:** Primarily designed for image classification; needs significant modifications to be used for object detection tasks.

Use Cases

- **Transfer Learning:** Commonly used as a feature extractor in transfer learning tasks.
- **Image Classification:** Suitable for tasks where the primary goal is to classify images.

Architecture Model Evaluation

ResNet50

Architecture

- **Type:** Convolutional Neural Network (CNN)
- **Depth:** 50 layers
- **Structure:** Uses residual blocks with skip connections to allow training of deeper networks

Pros

- **Residual Connections:** Helps in training very deep networks by mitigating the vanishing gradient problem.
- **High Accuracy:** Achieves better performance compared to VGG16 on many tasks.
- **Scalability:** Easier to scale to deeper networks (e.g., ResNet101, ResNet152).

Architecture Model Evaluation

ResNet50

Cons

- **Complexity:** More complex to implement and understand compared to VGG16.
- **Resource Intensive:** Requires substantial computational resources, especially for training.

Use Cases

- **Transfer Learning:** Widely used for transfer learning in various image classification and detection tasks.
- **Object Detection:** Often used as a backbone for object detection frameworks (e.g., Faster R-CNN).

Architecture Model Evaluation

YOLO (You Only Look Once)

Architecture

- **Type:** Convolutional Neural Network (CNN) designed for real-time object detection
- **Depth:** Varies (e.g., YOLOv3 has 53 convolutional layers in its backbone)
- **Structure:** Single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation

Architecture Model Evaluation

YOLO (You Only Look Once)

Pros

- **Real-Time Performance:** Extremely fast and suitable for real-time detection applications.
- **High Accuracy:** Provides a good balance between speed and accuracy for object detection tasks.
- **Unified Architecture:** End-to-end training directly on detection performance, making it highly efficient.

Cons

- **Complexity:** More challenging to implement and fine-tune compared to VGG16 and ResNet50.
- **Resource Intensive:** Requires substantial computational resources, especially for real-time applications.

Architecture Model Evaluation

YOLO (You Only Look Once)

Use Cases

- **Real-Time Object Detection:** Ideal for applications requiring real-time detection, such as video surveillance, autonomous driving, and interactive systems.
- **General Object Detection:** Suitable for various object detection tasks where speed and accuracy are both critical.

Transfer Learning

Knowledge Transfer

- During the explosion of deep learning, AI resources have become increasingly abundant.
- In parallel, more high-quality, accurate pretrained models have emerged, making it possible to find pretrained models for almost every domain.
- The theory of transfer learning was first experimented by Lorien Pratt in 1993 and later formalized as a mathematical theory in 1998.
- This theory realizes the idea of knowledge transfer between models, akin to knowledge transfer between humans.
- A model can leverage previously trained knowledge to improve its classification task.

Transfer Learning

Improve Accuracy and Save Training Costs

- For example, in the problem of classifying human face, training from scratch requires more epochs to achieve high accuracy.
- However, reusing pretrained models reduces the number of training epochs needed to achieve the expected accuracy, and the resulting accuracy can be even higher than without transfer learning.

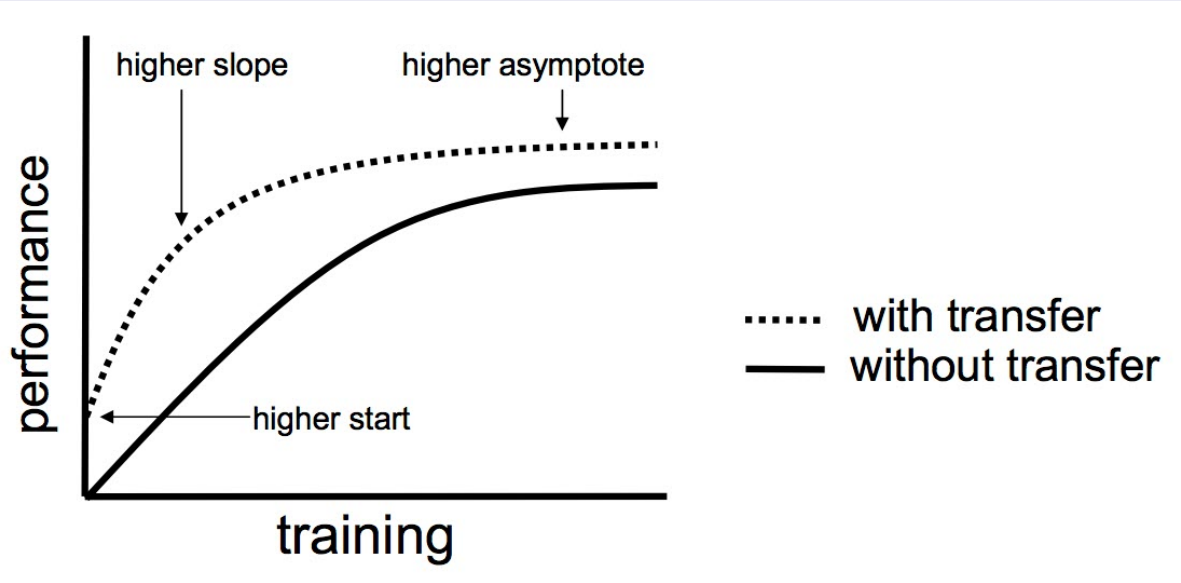


Figure 1: Diagram comparing model performance before and after applying transfer learning

Transfer Learning

Using transfer learning offers three main advantages:

- A better starting point (higher initial accuracy).
- Faster increase in accuracy (higher slope).
- Higher optimal accuracy (higher asymptote).

Efficient with Small Data

- In cases where the dataset is small and difficult to expand, models trained on such data alone struggle to predict well.
- Reusing knowledge from pretrained models with the same classification task helps the trained models predict better with new data by learning from both the training data and the previously learned data.

Transfer Learning

An Example of Transfer Learning

- The process of applying knowledge learned from a previous model to the current problem is called transfer learning.
- For instance, in the dog and cat classification problem, we have two labels to classify: dog and cat.
- Both labels appear in the ImageNet dataset. By leveraging a pretrained model on ImageNet, we can improve our model's performance on this task.

Proposed Solution

- **Model Selection**: Use YOLOv10-L, a state-of-the-art object detection model known for its speed and accuracy.
- **Data Collection**: Gather a dataset of diverse images containing faces, ensuring a balanced representation of different facial features.
- **Data Annotation**: Utilize the pre-labeled dataset downloaded from Kaggle, which includes about 32k images with clean labels in xywh (top-left x-coord, top-left y-coord, face-width, face-height) format for human face detection tasks.
- **Environment Setup**: Clone the YOLOv10-L repository and install dependencies in Google Colab for free GPU access.
- **Model Configuration**: Define the model architecture and configuration using a custom YAML file tailored for face detection.

Proposed Solution

- **Training**: Train the YOLOv10-L model on the annotated dataset, optimizing for accuracy and performance.
- **Evaluation**: Assess the model's performance using metrics such as mAP (mean Average Precision) and adjust parameters as needed.
- **Inference**: Test the trained model on new images to validate its face detection capabilities.
- **Deployment**: Export the trained model weights for use in real-world applications.

Architecture

- **YOLOv10 Detailed Structure**:
  - **Backbone**:
    - The backbone extracts features from images.
    - YOLOv10 uses an improved version of CSPNet (Cross Stage Partial Network) to improve the flow of information and use less computing power.
  - **Neck**:
    - The neck combines features from different backbone levels and passes them to the head.
    - It effectively mixes features from multiple scales(layers) using PAN (Path Aggregation Network) layers.
  - **One-to-Many Head**:
    - During training, this head makes several predictions for each object. This gives the model lots of information(features), helping it learn better.
  - **One-to-One Head**:
    - During inference, this head makes a single best prediction for each object. This removes the need for NMS (Non-Maximum Suppression), making the process faster.

Architecture

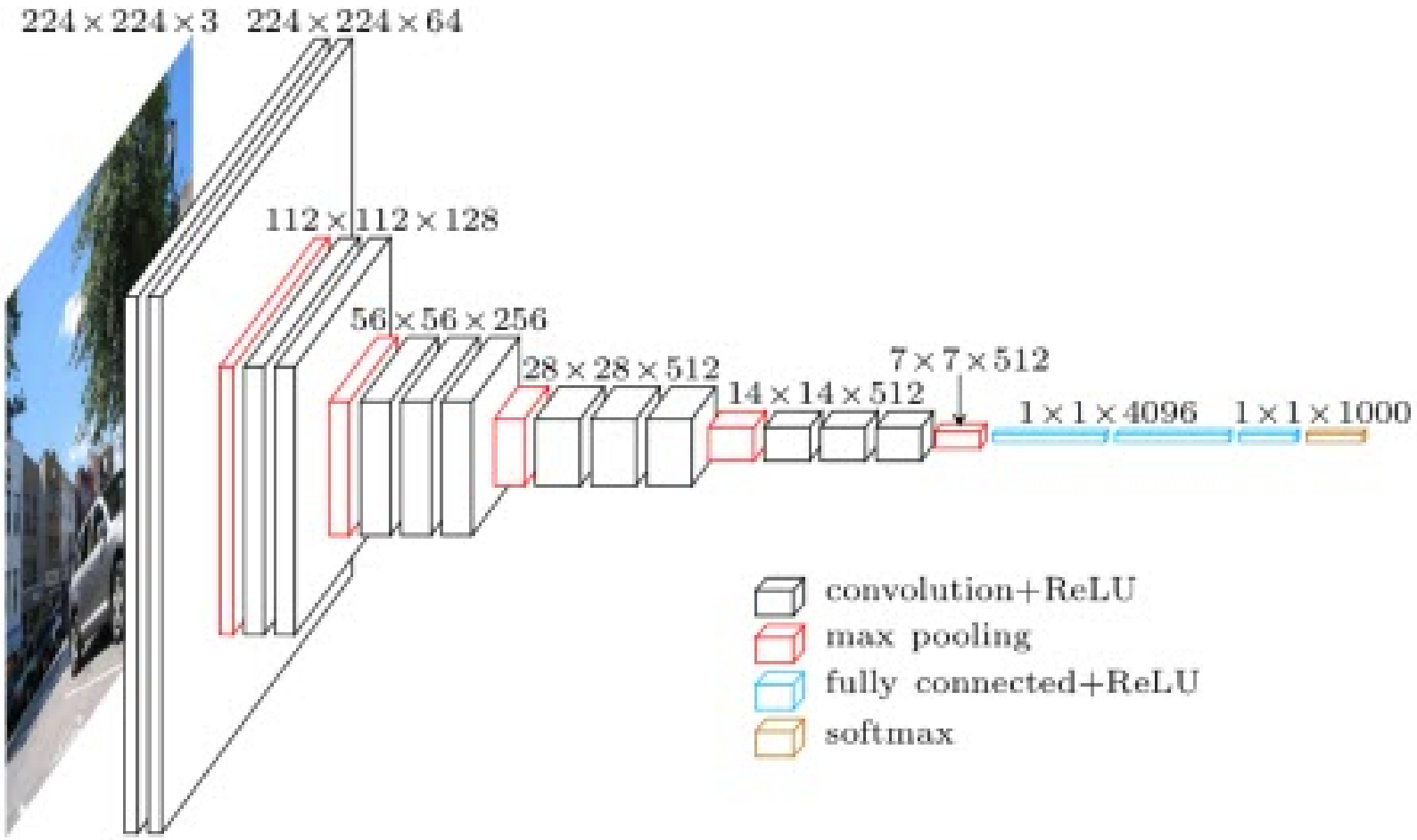


Figure 2: YOLO architecture

Architecture

Convolutional layers in CNNs (Convolutional Neural Networks) like YOLO are used to extract features from the input image. The filters (kernels) slide over the input image, performing element-wise multiplication and summing the results to produce feature maps. The size of the filter determines the receptive field, which is the region of the input image that influences a particular feature in the output feature map.

- **Larger filters** (like 7x7) capture more spatial information but with less detail.
- **Smaller filters** (like 3x3) capture finer details and can stack multiple layers to capture complex patterns.

The depth of the filters (number of filters) determines how many different features can be extracted at each spatial location. More filters allow the network to learn more features, making the network more powerful but also increasing the computational complexity.

Architecture

- **YOLOv10-L model configuration file (yolov10l.yaml)**:

```yaml
# Parameters
nc: 80 # number of classes
scales: # model compound scaling constants, i.e. 'model=yo
  # [depth, width, max_channels]
  l: [1.00, 1.00, 512] # YOLOv8l summary: 365 layers, 4369

# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 3, C2f, [128, True]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
```

Architecture

- **YOLOv10-L model configuration file (yolov10l.yaml)**:

```
  - [-1, 6, C2f, [256, True]]
  - [-1, 1, SCDown, [512, 3, 2]] # 5-P4/16
  - [-1, 6, C2f, [512, True]]
  - [-1, 1, SCDown, [1024, 3, 2]] # 7-P5/32
  - [-1, 3, C2fCIB, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9
  - [-1, 1, PSA, [1024]] # 10

# YOLOv8.0n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 3, C2fCIB, [512, True]] # 13
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
```

Architecture

- **YOLOv10-L model configuration file (yolov10l.yaml)**:

```
- [[-1, 4], 1, Concat, [1]] # cat backbone P3
- [-1, 3, C2f, [256]] # 16 (P3/8-small)

- [-1, 1, Conv, [256, 3, 2]]
- [[-1, 13], 1, Concat, [1]] # cat head P4
- [-1, 3, C2fCIB, [512, True]] # 19 (P4/16-medium)

- [-1, 1, SCDown, [512, 3, 2]]
- [[-1, 10], 1, Concat, [1]] # cat head P5
- [-1, 3, C2fCIB, [1024, True]] # 22 (P5/32-large)

- [[16, 19, 22], 1, v10Detect, [nc]] # Detect(P3, P4, P5
```

Data Processing

- **Preprocessing**:
  - **Normalization**: Adjust image pixel values to a common scale to improve model performance.
  - **Augmentation**: Apply techniques such as rotation, flipping, and scaling to increase dataset diversity.
  - **Annotation**: Utilize the pre-labeled dataset downloaded from Kaggle, which includes about 32k images with clean labels.
  - **Dataset Preparation**: Ensure balanced representation of various facial features and expressions.

Dataset

- **Source**:
  - We used the Face Detection Dataset from Kaggle.
  - This dataset is specifically curated for training and testing face detection models.
- **Dataset Composition**:
  - **Training Set**: 26,300 images with annotated face locations.
  - **Validation Set**: 6,500 images with similar annotations.
- **Annotations**:
  - Each image comes with corresponding labels indicating face positions using bounding boxes.

Dataset

- **Preparation**:
  - Downloaded and extracted the dataset using a simple helper script.
  - Ensured the removal of duplicate images and corresponding labels.
- **Directory Structure**:
  - Organized as follows:

```
train
    images
    labels
validation
    images
    labels
test
    images
face-detect-datase.yaml
```

Dataset

- **YAML Configuration**:
  - Defined paths for training and validation data in a `face-detect-datase.yaml`.
  - Included class names and counts for model reference.
  - **Training Data Path**: `/content/drive/MyDrive/face-detection-project/merged/images/train`
  - **Validation Data Path**: `/content/drive/MyDrive/face-detection-project/merged/images/validation`
  - **Class Names**: `['face']`
  - **Number of Classes (nc)**: `1`

Environment Setup

- **GPU Status Check**:
  - Firstly we ensure the availability and readiness of the GPU for processing and running the YOLOv10 model.
  - Command:
    `!nvidia-smi`

```
[1]  !nvidia-smi

Sun Jun 30 15:33:10 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA L4                      Off | 00000000:00:03.0 Off |                    0 |
| N/A   44C    P8              12W /  72W |      1MiB / 23034MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Figure 3: GPU Status

Environment Setup

- **Mounting Google Drive**:
  - Next, we need to navigate to folder where the dataset is stored
  - Command:

```
from google.colab import drive
drive.mount('/content/drive')
```

- **Display Current Directory**:
  - Store current working directory's path in the HOME variable for reference.
  - Command:

```
import os
HOME = os.getcwd()
print(HOME)
```

- **Install YOLOv10**:
  - To set up the environment for running the YOLOv10 model, we need to install and clone the YOLOv10 repository. These installations are crucial for ensuring that all necessary tools and libraries are available for the model to function correctly.

Environment Setup

- **Install YOLOv10**:
  - Command:
    ```
    !pip install git+https://github.com/THU-MIG/yolov10.git
    ```
- **Download YOLOv10-L Pre-trained Weights**:
  - This step involves creating a directory for storing the model weights and downloading the pre-trained weights for YOLOv10-L model. These weights are essential for initializing the model and can be used for both inference and fine-tuning.
  - Command:
    ```
    !mkdir -p {HOME}/weights
    !wget -P {HOME}/weights -q https://github.com/jameslahm
    /yolov10/releases/download/v1.0/yolov10l.pt
    ```

Evaluation Metrics

- **Average Precision (AP)**:
  - Measures precision and recall at various thresholds.
  - Calculates the weighted mean of precisions achieved at each threshold.
  - Provides a comprehensive view of model performance across different confidence levels.
- **Mean Average Precision (mAP)**:
  - Combines AP scores over multiple IoU thresholds (e.g., 0.5 to 0.95).
  - Averages AP across all classes in the dataset.
  - Offers a comprehensive metric for overall model performance comparison.
- **AP@0.5**:
  - Measures precision and recall with a fixed Intersection over Union (IoU) threshold of 0.5.
  - Indicates how well the model distinguishes true positives from false positives.

Evaluation Metrics

- **P Curve (Precision Curve)**:
  - Shows the precision at various confidence levels.
  - Helps in understanding how precision changes with different confidence thresholds.
- **PR Curve (Precision-Recall Curve)**:
  - Plots precision against recall for different threshold values.
  - Provides insight into the trade-off between precision and recall.
- **R Curve (Recall Curve)**:
  - Displays recall at various confidence levels.
  - Useful for understanding how recall varies with different confidence threshold
- **Confusion Matrix**:
  - This cell displays the confusion matrix generated during training.
  - The confusion matrix helps in visualizing the performance of the model by showing the accuracy of its predictions.

Training Custom YOLOv10-L Detector

- **Model Summary**: YOLOv10-L with 628 layers, 25,766,870 parameters, and 127.2 GFLOPs.
- **Training**:
  - Total Epochs: 50
  - Batch Size: 8
  - Image Size: 640x640 pixels
  - Data Augmentation:
    - **Color Augmentation**: Adjustments in hue (hsv_h=0.02), saturation (hsv_s=0.8), and value (hsv_v=0.5).
    - **Geometric Transformations**: Includes rotation (degrees=5), translation (translate=0.2), scaling (scale=0.6), and shearing (shear=2).
    - **Perspective Transform**: Small adjustments with perspective (perspective=0.001).
    - **Flipping**: Both vertical flipping (flipud=0.1) and horizontal flipping (fliplr=0.6).

Data Augmentation

- **Training**:
  - Data Augmentation:
    - **Mosaic**: Combines four images into one (mosaic=1.0).
    - **MixUp**: Merges two images into one (mixup=0.2).
    - **Copy-Paste**: Pastes objects from one image into another (copy_paste=0.1).
    - **Auto Augment**: Uses RandAugment strategy for automatic augmentation.
    - **Erasing**: Randomly erases parts of images (erasing=0.5).
    - **Cropping**: Applies cropping with a fraction of 1.0 (crop_fraction=1.0).

Data Augmentation

Figure 4: Data Augmentation

Data Augmentation

Figure 5: Data Augmentation

Benchmark

- **Training Results**:
  - Initial Epoch GPU Memory: 17.2G
  - Final Epoch GPU Memory: 14.7G
  - Total Training Time: 13.777 hours
- **Validation Results**:
  - Precision (P): 0.861
  - Recall (R): 0.669
  - mAP@0.5: 0.735
  - mAP@0.5:0.95: 0.42
- **Inference Speed**:
  - Preprocess: 0.2ms per image
  - Inference: 28.1ms per image
  - Postprocess: 0.1ms per image
- **Model Performance**: Efficient detection and robust accuracy, suitable for real-time face detection applications

Benchmark

- Command:

```
!yolo task=detect mode=train epochs=50 batch=12
imgsz=640 plots=True model='/content/drive/MyDrive/
face-detection-project/yolov10/weights/yolov10l.pt'
data='/content/drive/MyDrive/face-detection-project/
merged/face-detect-datase.yaml' project='/content/
drive/MyDrive/face-detection-project/runs/detect/
train4/weights' name='train4' augment=True
hsv_h=0.02 hsv_s=0.8 hsv_v=0.5 degrees=5 translate=0.2
scale=0.6 shear=2 perspective=0.001 flipud=0.1
fliplr=0.6 mosaic=1.0 mixup=0.2 copy_paste=0.1
auto_augment=randaugment erasing=0.5 crop_fraction=1.0
```

Benchmark

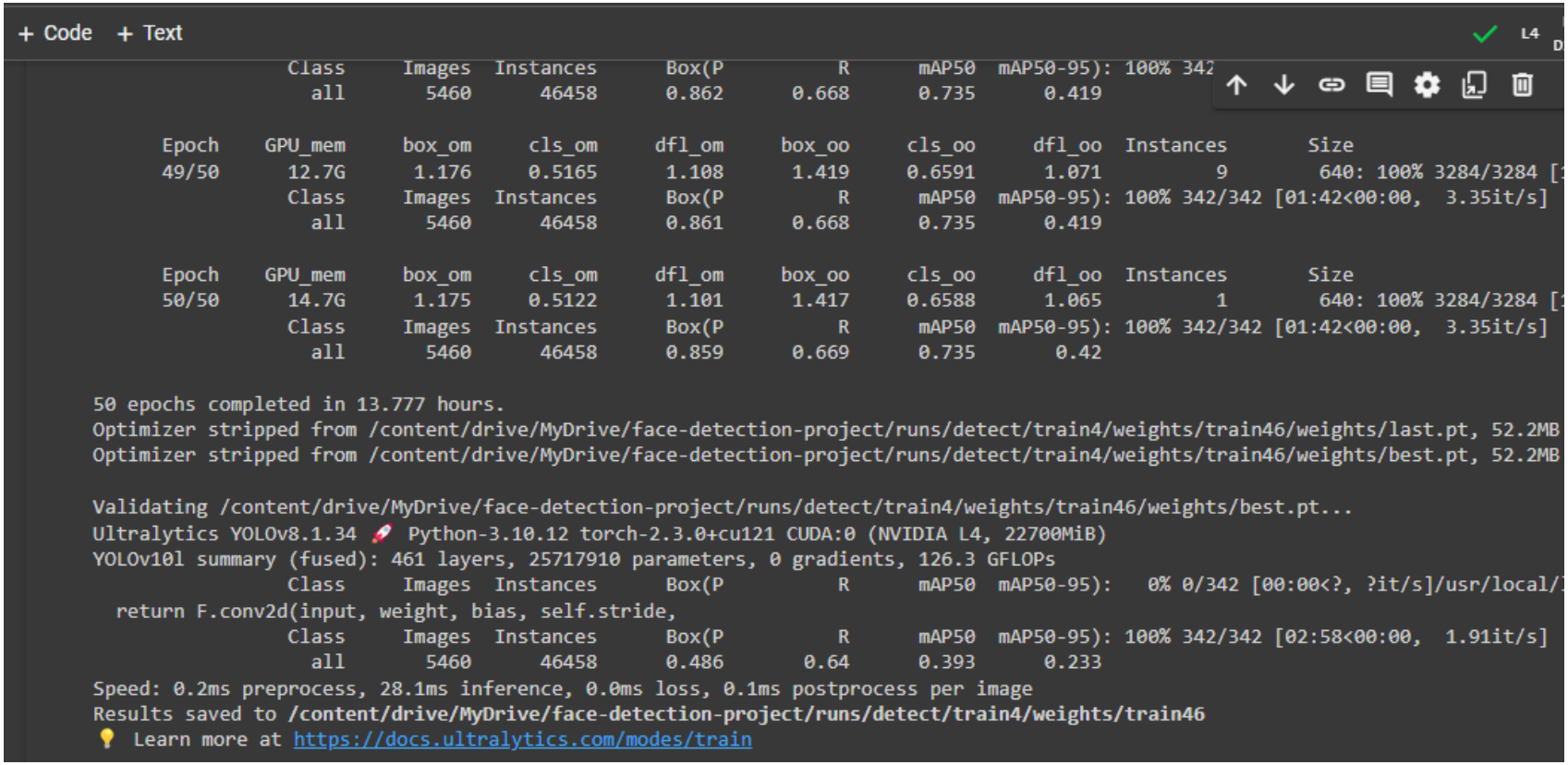


Figure 6: Train model

Benchmark

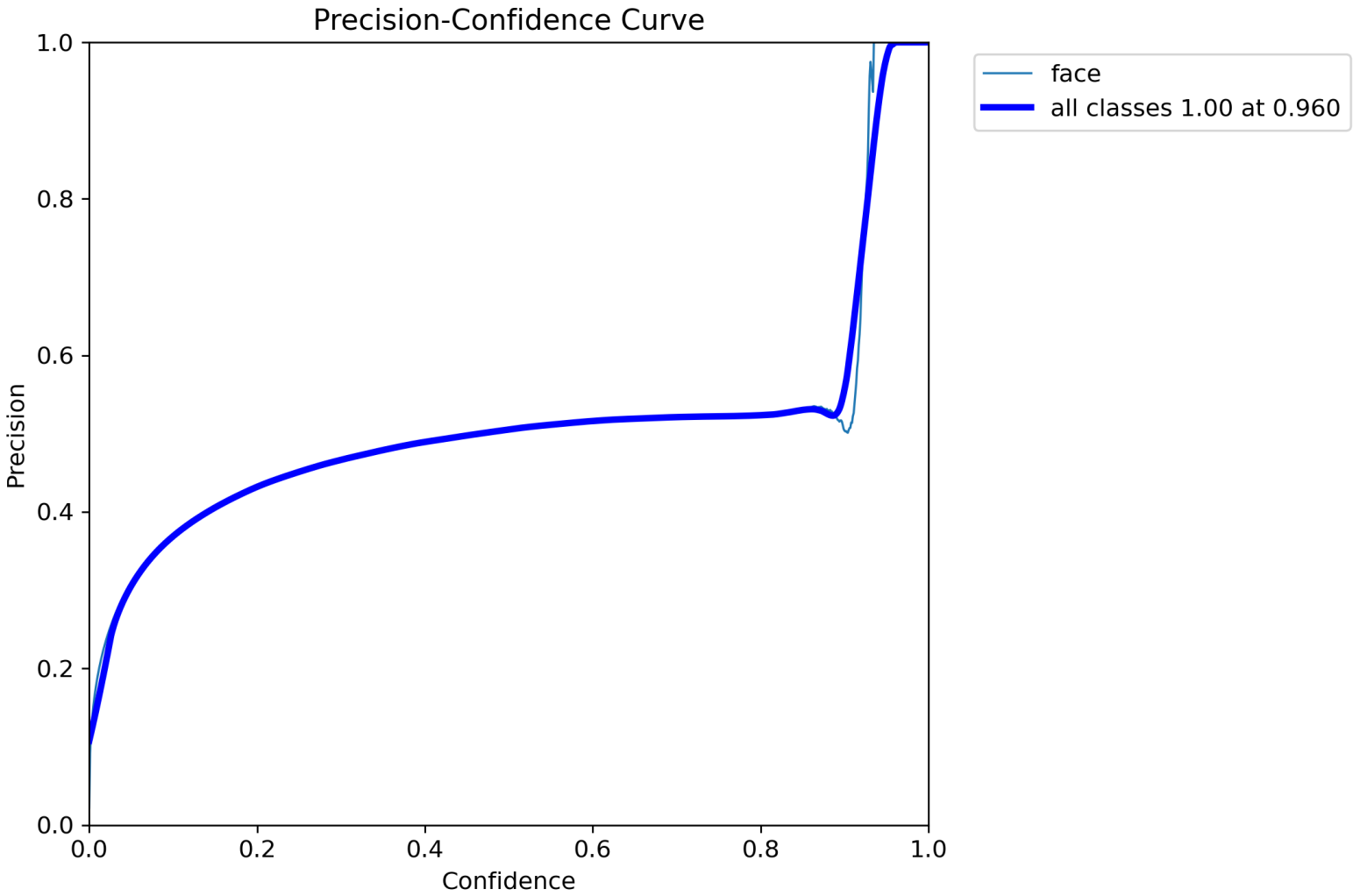


Figure 7: P curve

Benchmark

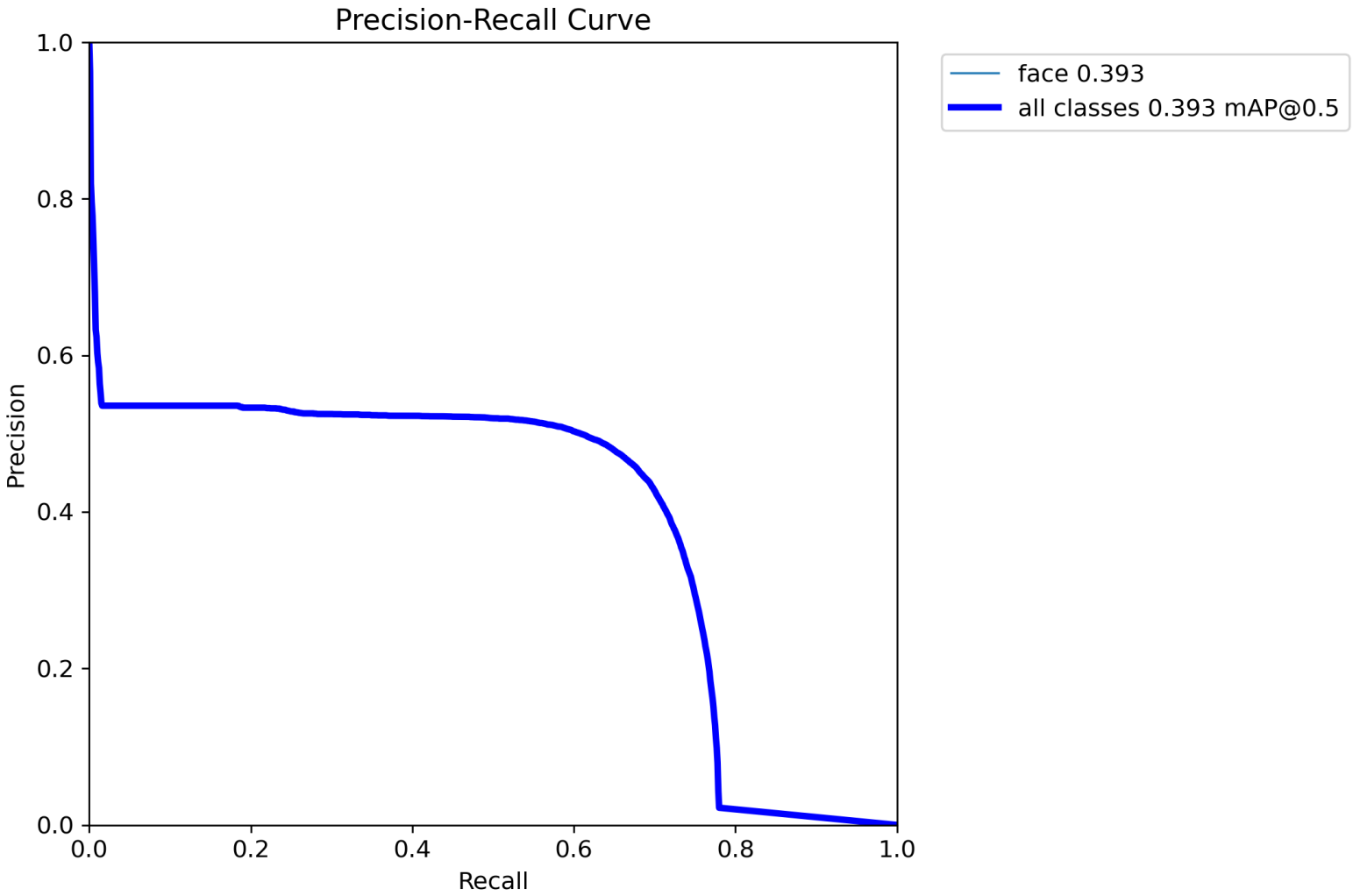


Figure 8: PR curve

Benchmark

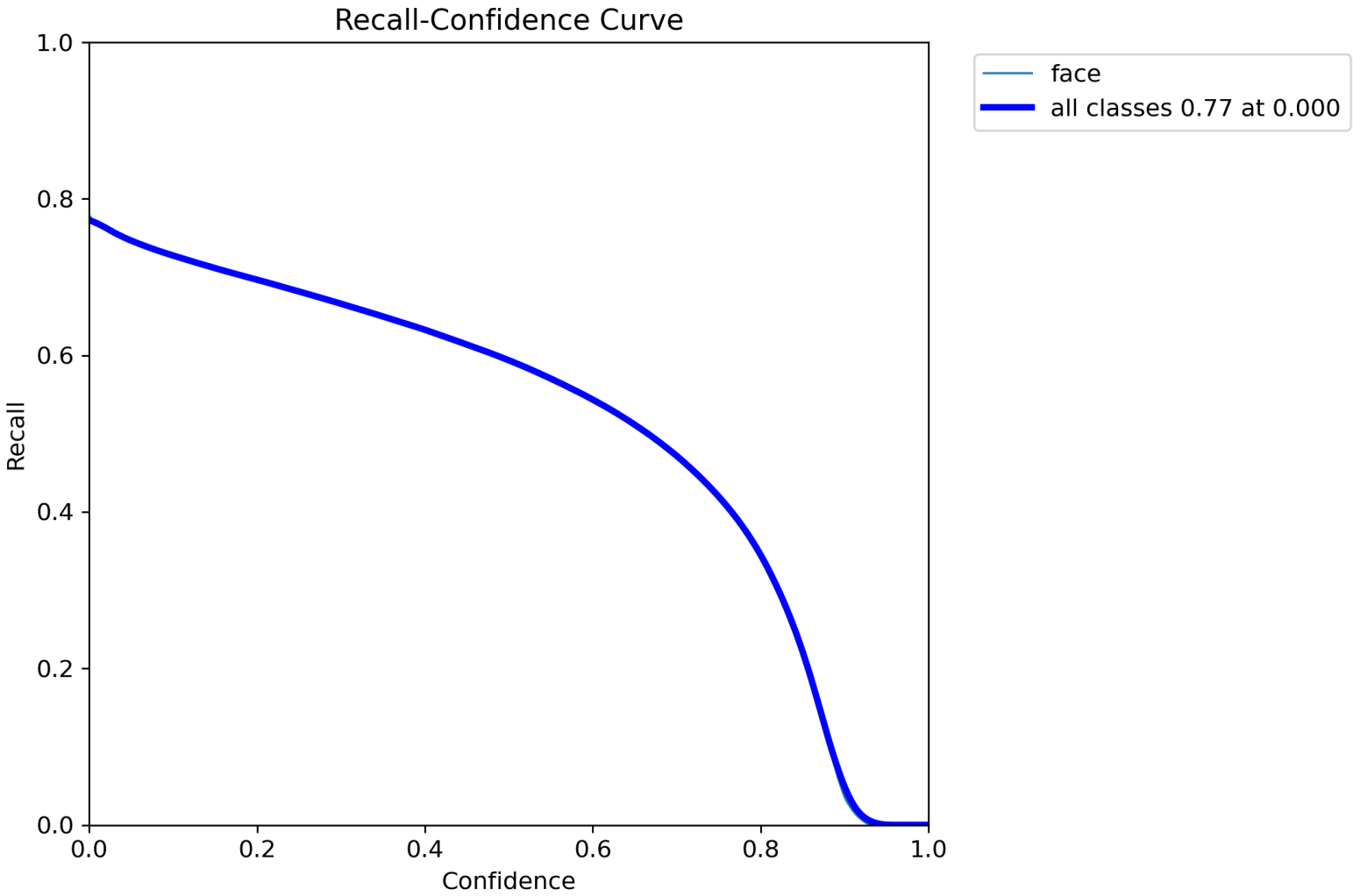


Figure 9: R curve

Benchmark

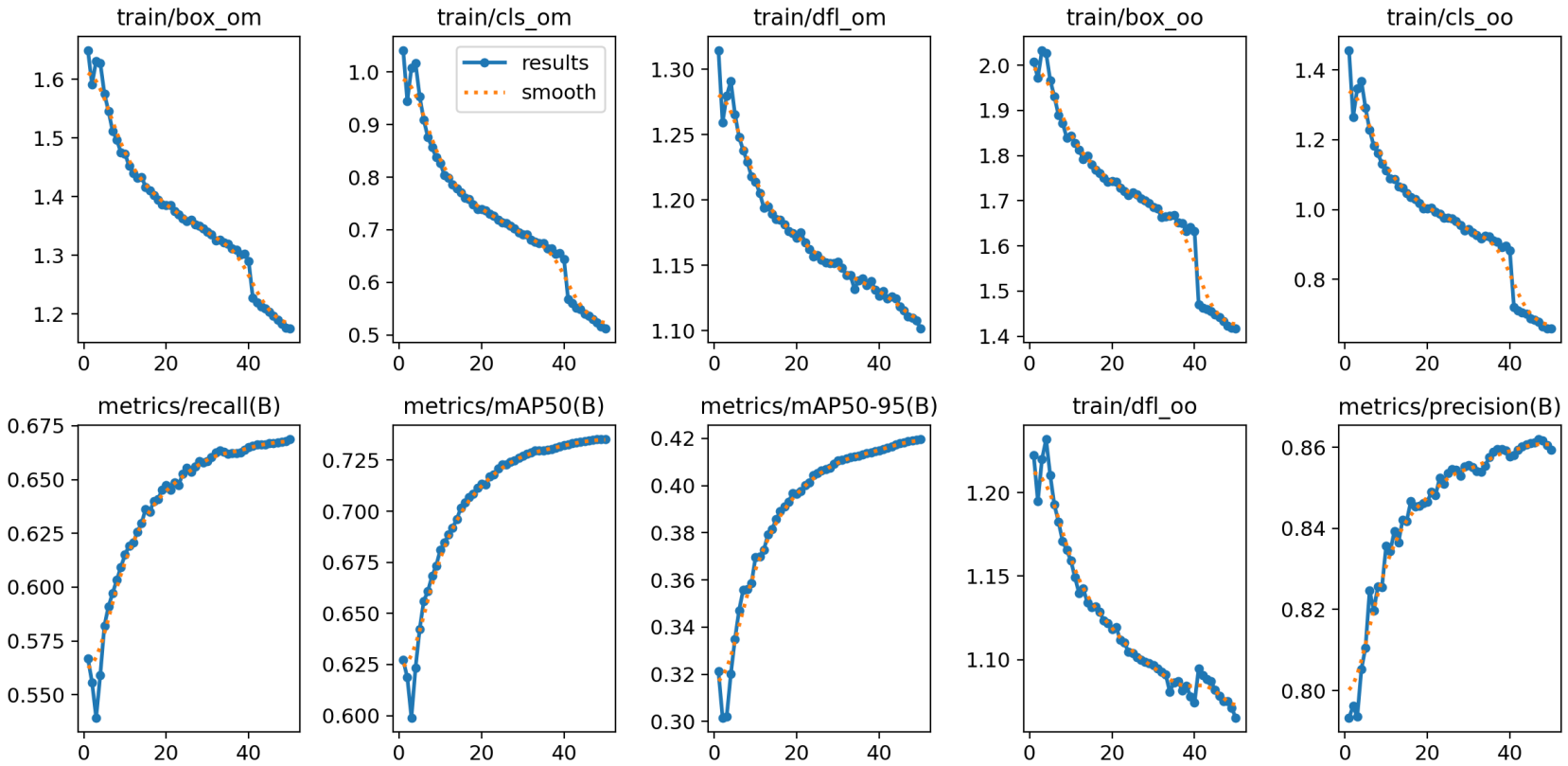


Figure 10: Train model result

Benchmark

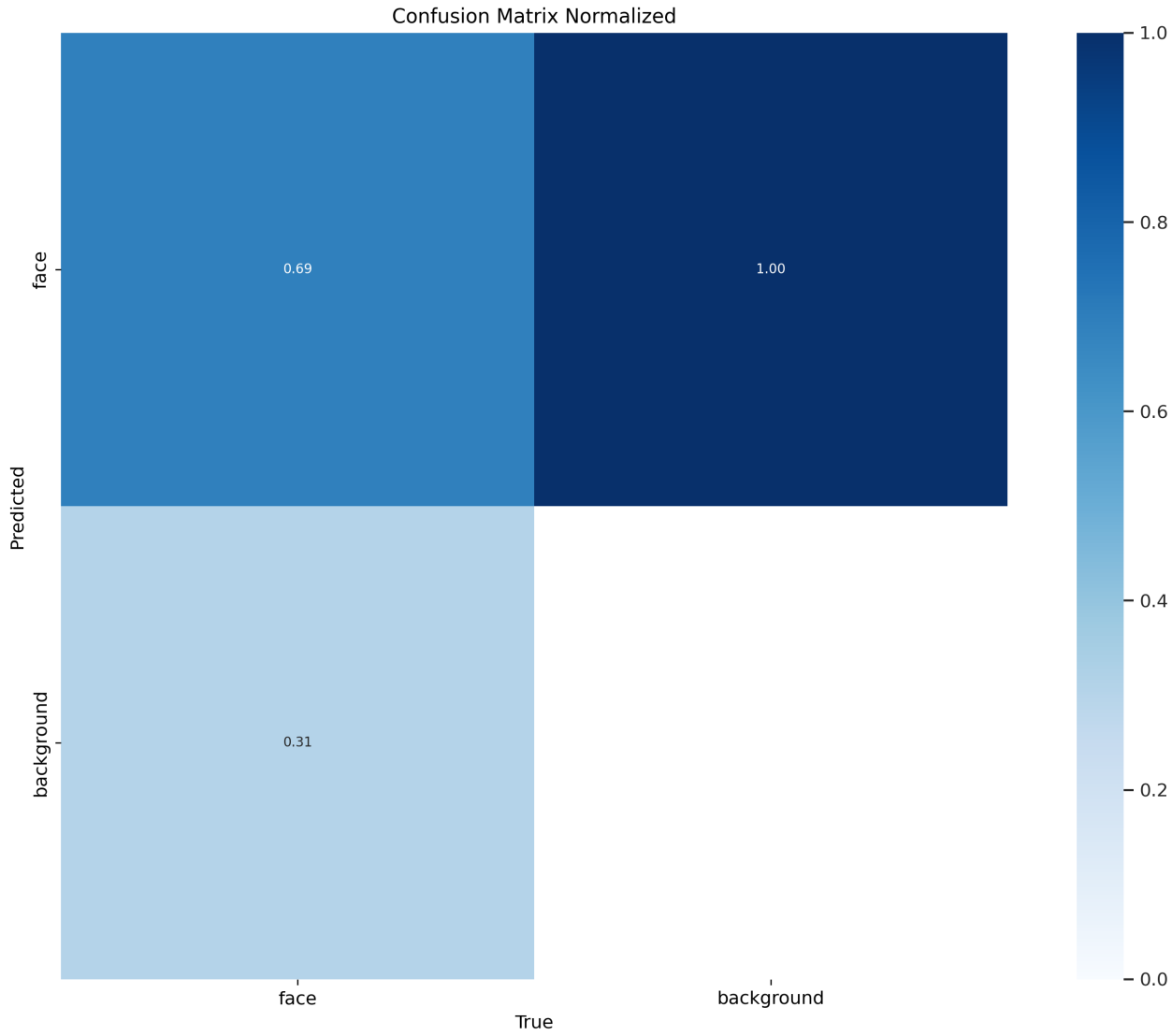


Figure 11: Confusion Matrix

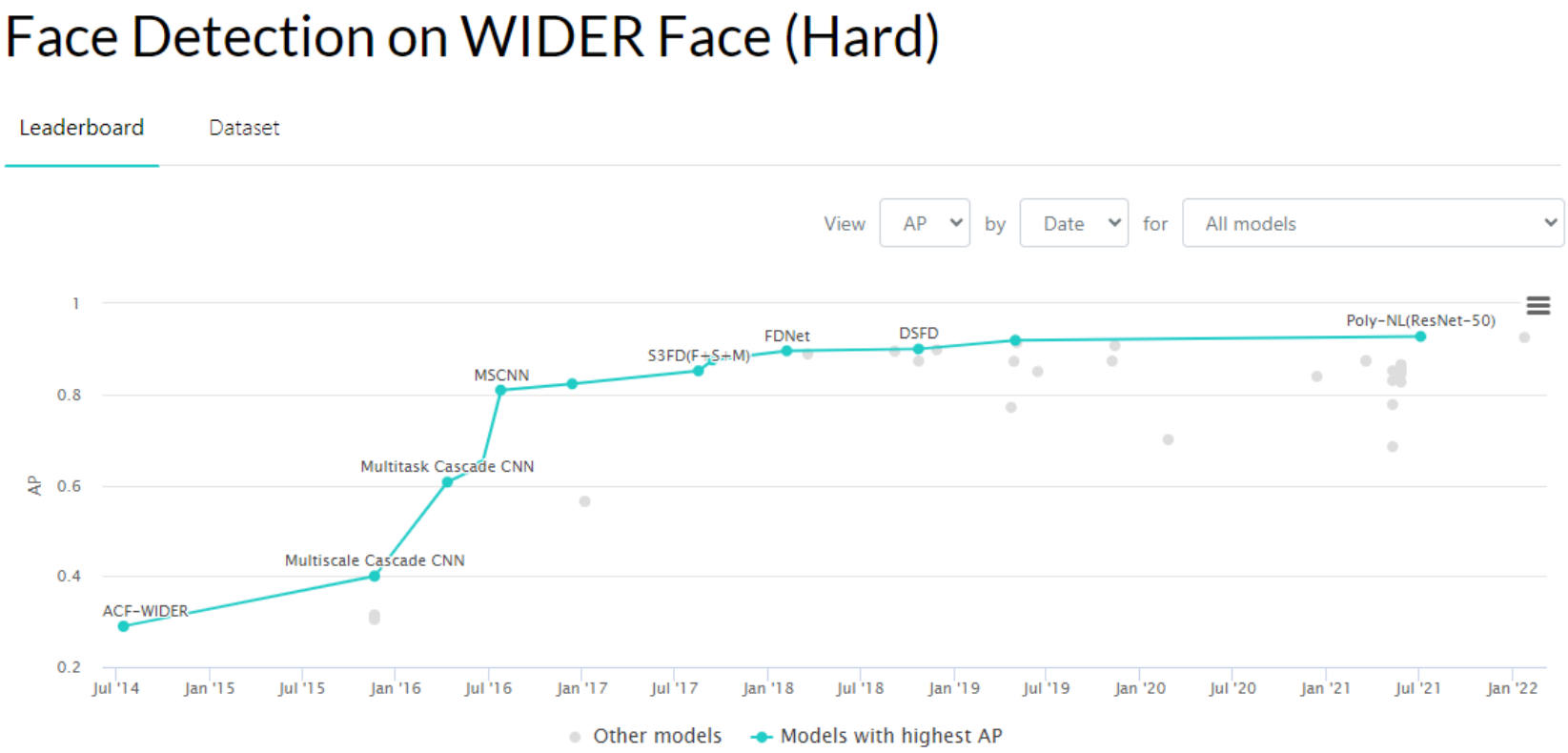
Comparison with Other SOTA Models

- The average precision (mAP@0.5) of our detector was 73.5%, while the mAP@0.5:0.95 reached 41.9%.

Detector	**mAP**	**Params(M)**	**GFLOPS**
Poly-NL(ResNet-50)	0.928	23.5	5.6
DSFD	0.900	120.0	259.55
FDNet	0.896	45.0	10.5
S3FD(F,S+M)	0.852	37.0	9.2
MSCNN	0.809	20.1	6.3
Our detector	**0.735**	**25.7**	**127**
Multitask Cascade CNN	0.607	25.4	7.8
Multiscale Cascade CNN	0.400	30.2	8.5
ACF-WIDER	0.29	15.6	4.1

Table 2: Comparison of various face detectors.

Comparison with Other SOTA Models



Refer: Face Detection on SOTA.

Test Results: Inference on Image

- **Inference Details**:
  - Model: YOLOv10l
  - Parameters: 46 layers, 25,717,910 parameters, 126.3 GFLOPs
  - Inference speed: 158.4ms per image
  - Command:
    ```
    !yolo task=detect mode=predict conf=0.25 save=True
    model="/content/drive/MyDrive/face-detection-project/
    runs/detect/train4/weights/train46/weights/best.pt"
    source="/content/drive/MyDrive/face-detection-project/
    test/12_Group_Group_12_Group_Group_12_2.jpg"
    project="/content/drive/MyDrive/face-detection-project/
    runs/detect/predict" name="prediction_results"
    ```
  - Results saved to:
    `/content/drive/MyDrive/face-detection-project/runs/detect/predict/prediction_results4`

Test Results: Inference on Image

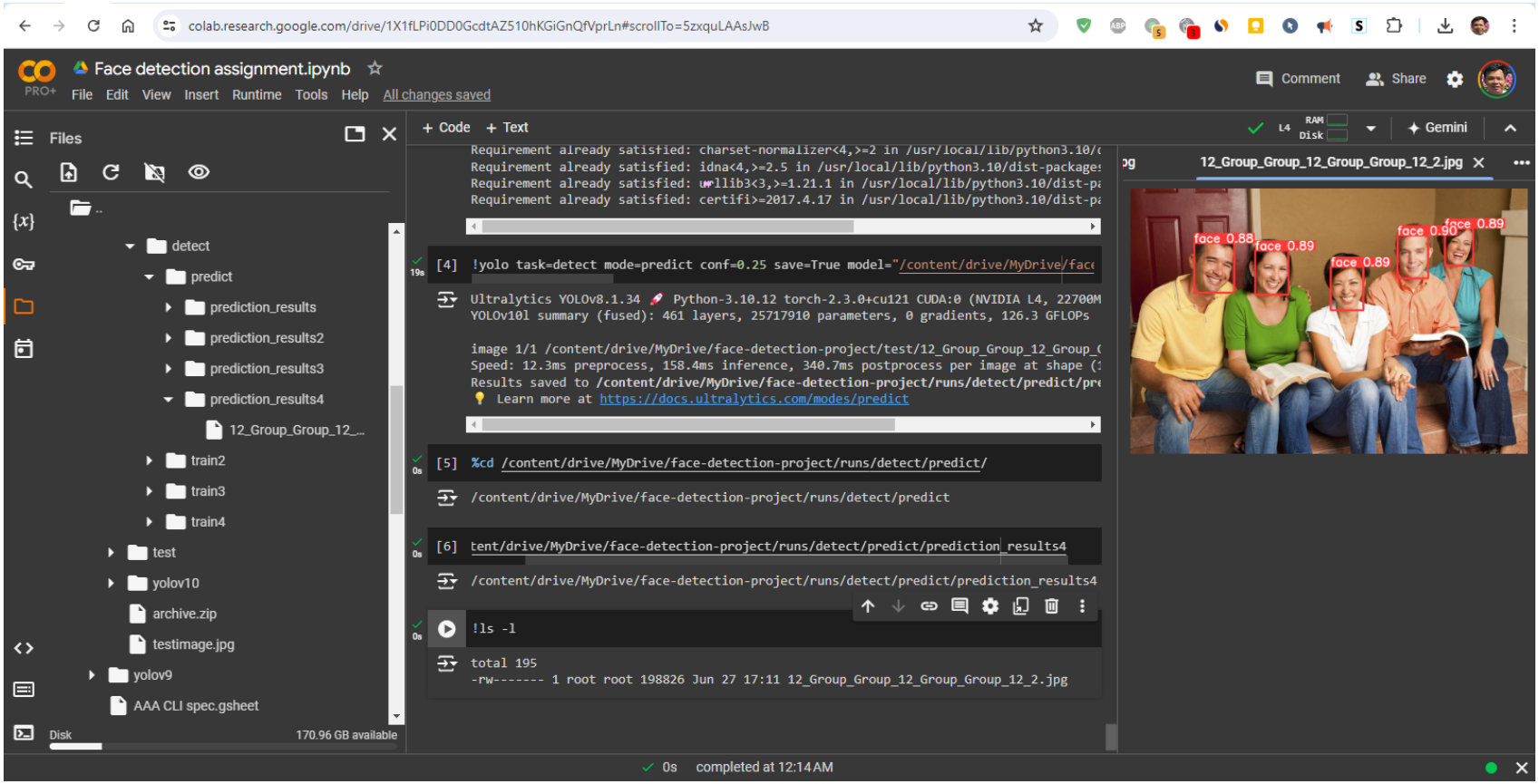


Figure 12: Google Colab workspace

Test Results: Inference on Image

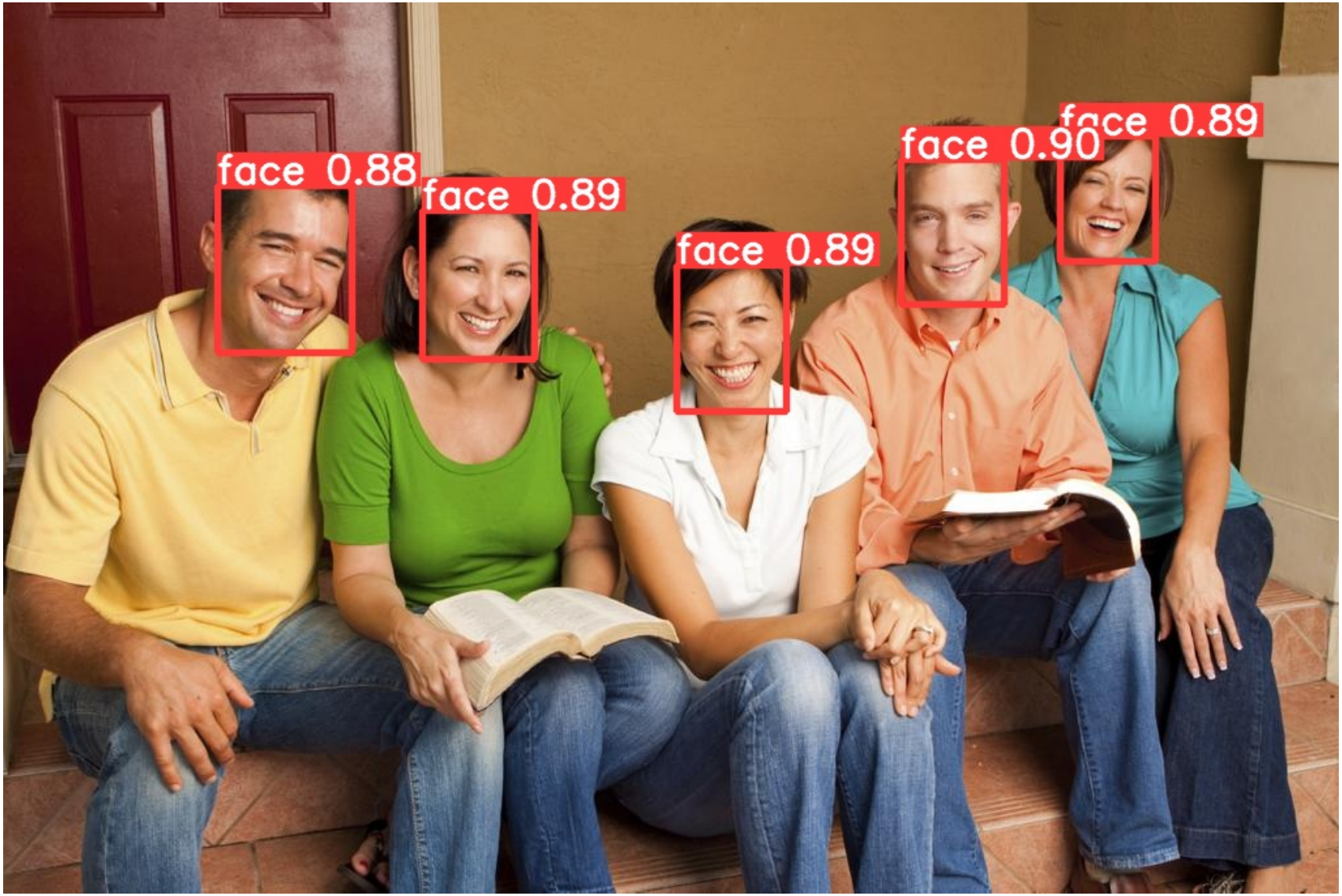


Figure 13: Inference on image

Test Results: Inference on Image

- **Inference Details**:
  - Model: YOLOv10l
  - Parameters: 461 layers, 25,717,910 parameters, 126.3 GFLOPs
  - Inference speed: 155.5ms per image
  - Command:
    ```
    !yolo task=detect mode=predict conf=0.25 save=True
    model="/content/drive/MyDrive/face-detection-project/
    runs/detect/train4/weights/train46/weights/best.pt"
    source="/content/drive/MyDrive/face-detection-project/
    test/runs/AOS-group.png"
    project="/content/drive/MyDrive/face-detection-project/
    runs/detect/predict" name="prediction_results"
    ```
  - Results saved to:
    `/content/drive/MyDrive/face-detection-project/runs/detect/predict/prediction_results5`

Test Results: Inference on Image

Figure 14: Inference on image

Test Results: Inference on Video

- Command:

```
!yolo task=detect mode=predict conf=0.25 save=True
model="/content/drive/MyDrive/face-detection-project
/runs/detect/train4/weights/train46/weights/best.pt"
source="/content/drive/MyDrive/face-detection-project
/test/WALK-NEW-YORK-City-USA-vlog.mp4" project="/conter
drive/MyDrive/face-detection-project/runs/detect/predic
name="prediction_results"
```

- Log:

```
Streaming output truncated to the last 5000 lines.
video 1/1 (frame 76799/81795) /content/drive/MyDrive/
face-detection-project/test/WALK-NEW-YORK-City-USA-vlog
 384x640 1 face, 16.6ms
```

Test Results: Inference on Video

- Log:

```
video 1/1 (frame 76800/81795) /content/drive/MyDrive/
face-detection-project/test/WALK-NEW-YORK-City-USA-vlog
 384x640 2 faces, 16.8ms
video 1/1 (frame 77973/81795) /content/drive/MyDrive/
face-detection-project/test/WALK-NEW-YORK-City-USA-vlog
384x640 (no detections), 16.7ms
...
Speed: 2.3ms preprocess, 17.3ms inference, 1.1ms
postprocess per image at shape (1, 3, 384, 640)
Results saved to /content/drive/MyDrive/face-detection-
project/runs/detect/predict/prediction_results6
```

- Play the inference result video:
https://www.youtube.com/watch?v=HwbmiKk6k3I

Test Results: Inference on Video

Figure 15: Inference on video

Conclusion

- Face detection is a vital technology
- Wide range of applications
- Project aims to contribute to this field

Questions?

- Open for any questions or discussions

References

- Dataset: Face Detection Dataset
- Information: Train set - 26,300 images, Test set - 6,500 images
- Evaluation Metric: AP, AP@0.5