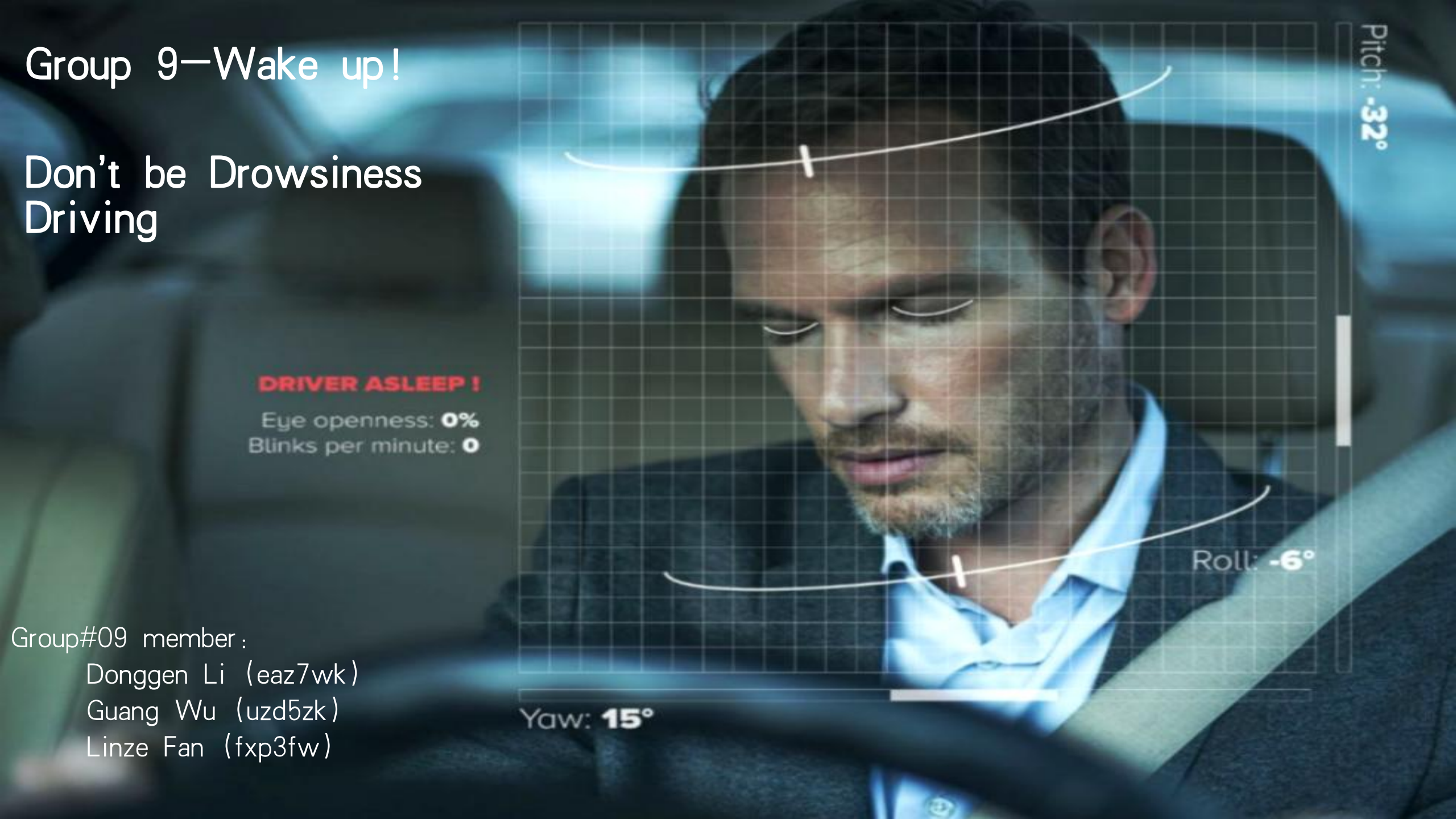
Group 9—Wake up!

Don’t be Drowsiness Driving



Group#09 member：

Donggen Li （eaz7wk）

Guang Wu （uzd5zk）

Linze Fan （fxp3fw）

Project Objective

Use **Nvidia Jetson Nano** platform with provided camera to run inference on image classification tasks for deployed models on the platform locally.

This project aims to develop a real-time drowsiness detection system based on Jetson Nano. The system captures the driver's facial features such as eye closure and yawning in real-time and analyzes them using deep learning algorithms to identify the signs.

Project Advantages

Cost-Effective
Compared to traditional drowsiness detection systems, the Jetson Nano-based system is more affordable.

Scalability
The system can be expanded with additional features as needed.

Easy Integration
Compact hardware assize and flexible interface design facilitate integration with other in-vehicles systems.

Nvidia Jetson Nano

A small, powerful computer designed for edge AI applications.

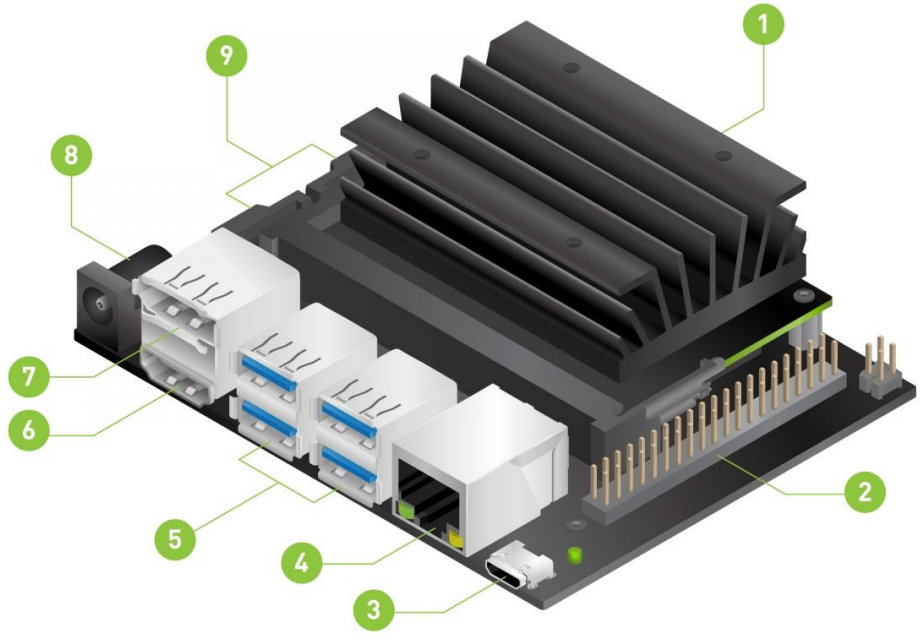
Use Cases:
Ideal for edge AI applications like computer vision, robotics and IoT, allowing real—time inference in devices with limited power and space

Power Efficiency:
Designed to operate with low power consumption (as little as 5 watts).

TECHNICAL SPECIFICATIONS

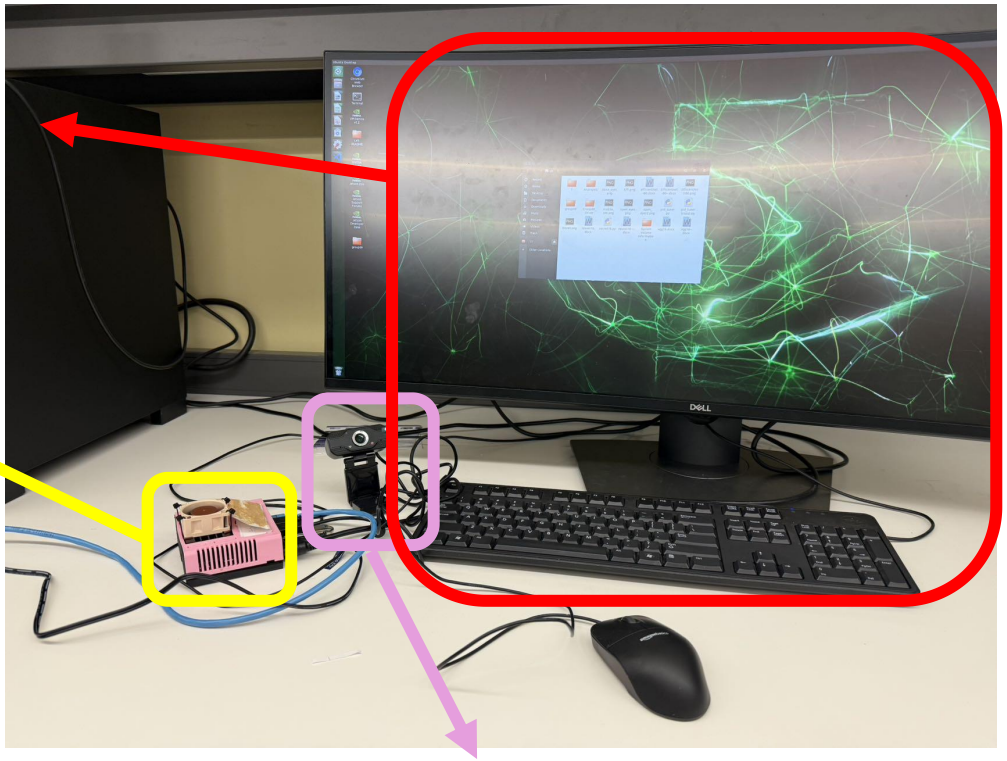
GPU	NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores
CPU	Quad-core ARM Cortex-A57 MPCore processor
Memory	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
Storage	16 GB eMMC 5.1

Initial Deployment Setup



Monitor
&
Keyboard

Jetson
Nano



Camera

1. microSD card slot for main storage
2. 40-pin expansion header
3. Micro-USB port for 5V power input, or for Device Mode
4. Gigabit Ethernet port
5. USB 3.0 ports (x4)
6. HDMI output port
7. DisplayPort connector
8. DC Barrel jack for 5V power input
9. MIPI CSI-2 camera connectors

```python
class DrowsinessDataset(Dataset):
    def __init__(self, image_dir, transform=None, validation_ratio=0.2, is_validation=False):
        files = list(
            map(lambda x: {'file': x, 'label': 1}, glob.glob('dataset/dataset_B_Eye_Images/openRightEyes/*.jpg')))
        files.extend(list(
            map(lambda x: {'file': x, 'label': 0}, glob.glob('dataset/dataset_B_Eye_Images/closedRightEyes/*.jpg'))))

        random.shuffle(files)
        validation_length = int(len(files) * validation_ratio)
        files = files[:validation_length] if is_validation else files[validation_length:]

        for file in files:
            img = cv2.imread(file['file'])
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            self.images.append(img)
            self.labels.append(file['label'])
```

Creates a dataset class to load images, assign labels (1 for open, 0 for closed), and split data into training and validation sets.

```python
# Load the trained model
model = models.mobilenet_v2()
model.features[0][0] = torch.nn.Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))  # For grayscale input
model.classifier[1] = torch.nn.Linear(model.last_channel, 2)  # Binary classification
model.load_state_dict(torch.load("mobilenet_model.pth"))  # Load the trained weights
model.eval()
```

Loads the pre-trained MobileNetV2 model and customizes its input layer for grayscale images and output layer for binary classification.

```python
# Helper function to preprocess an image and predict
def predict_drowsiness(image):
    # Convert image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    pil_image = Image.fromarray(gray)

    # Transform and prepare input
    transformed_image = transform(pil_image).unsqueeze(0).to(device)

    # Predict using the model
    with torch.no_grad():
        output = model(transformed_image)
        _, prediction = torch.max(output, 1)
    return prediction.item()  # 1 = Open eyes, 0 = Closed eyes
```

Converts a single eye image into a model-compatible format and classifies it as "open" or "closed."

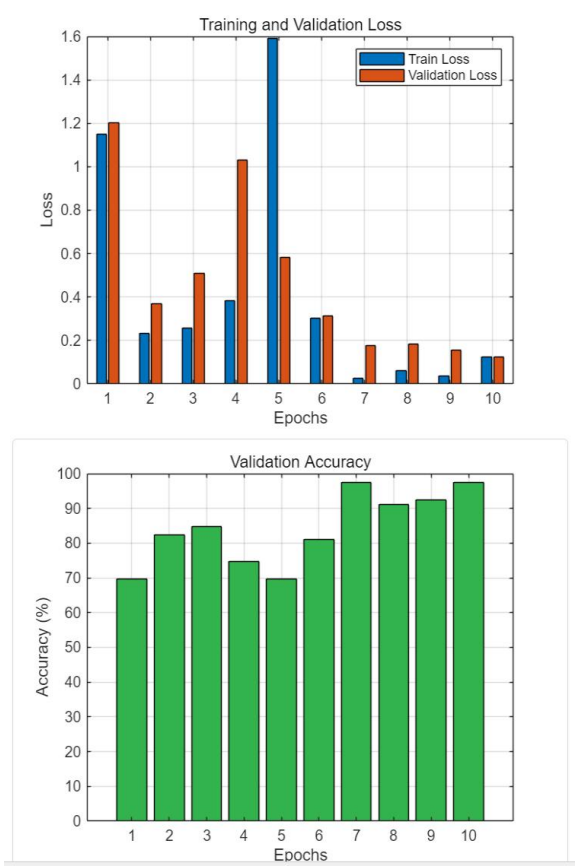
Captures real-time video, detects faces and eyes, classifies each eye, and displays the results.

```
cap = cv2.VideoCapture(0)  # Open webcam
while True:
    ret, frame = cap.read()  # Read video frames
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)  # Detect faces
    for (x, y, w, h) in faces:
        face = frame[y:y + h, x:x + w]
        eyes = eye_cascade.detectMultiScale(face)  # Detect eyes
        for (ex, ey, ew, eh) in eyes:
            eye = face[ey:ey + eh, ex:ex + ew]
            label = predict_drowsiness(eye)  # Predict drowsiness
            text = "Open" if label == 1 else "Closed"
            color = (0, 255, 0) if label == 1 else (0, 0, 255)
            cv2.rectangle(face, (ex, ey), (ex + ew, ey + eh), color, 2)  # Draw results
```

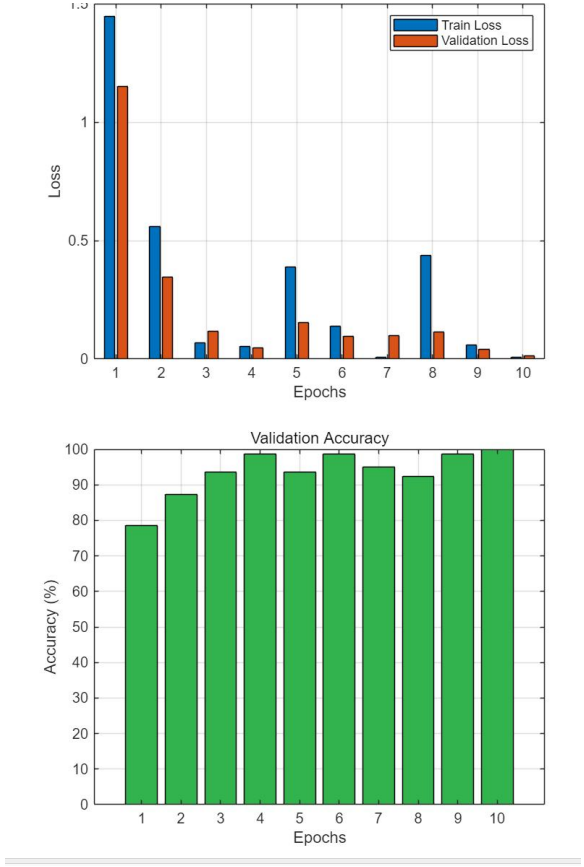
Deployment —— docker

Deployed, tested and compared three modules: ResNet 18, MobileNet and EfficientNet.

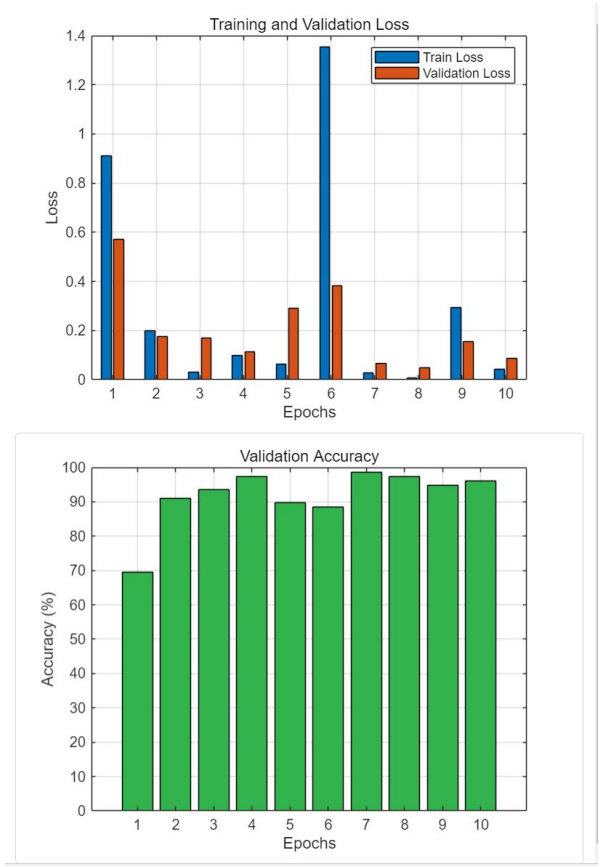
ResNet 18



MobileNet



EfficientNet



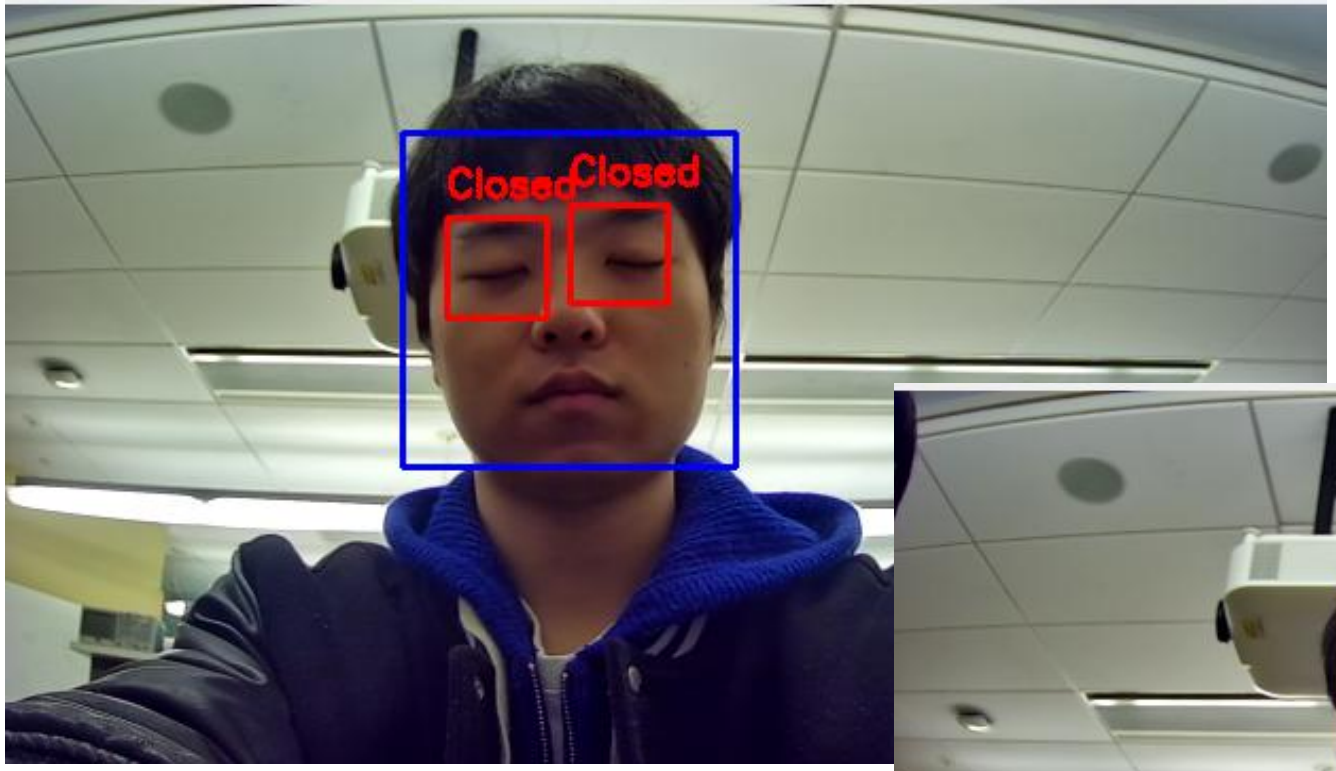
Why choose MobileNet

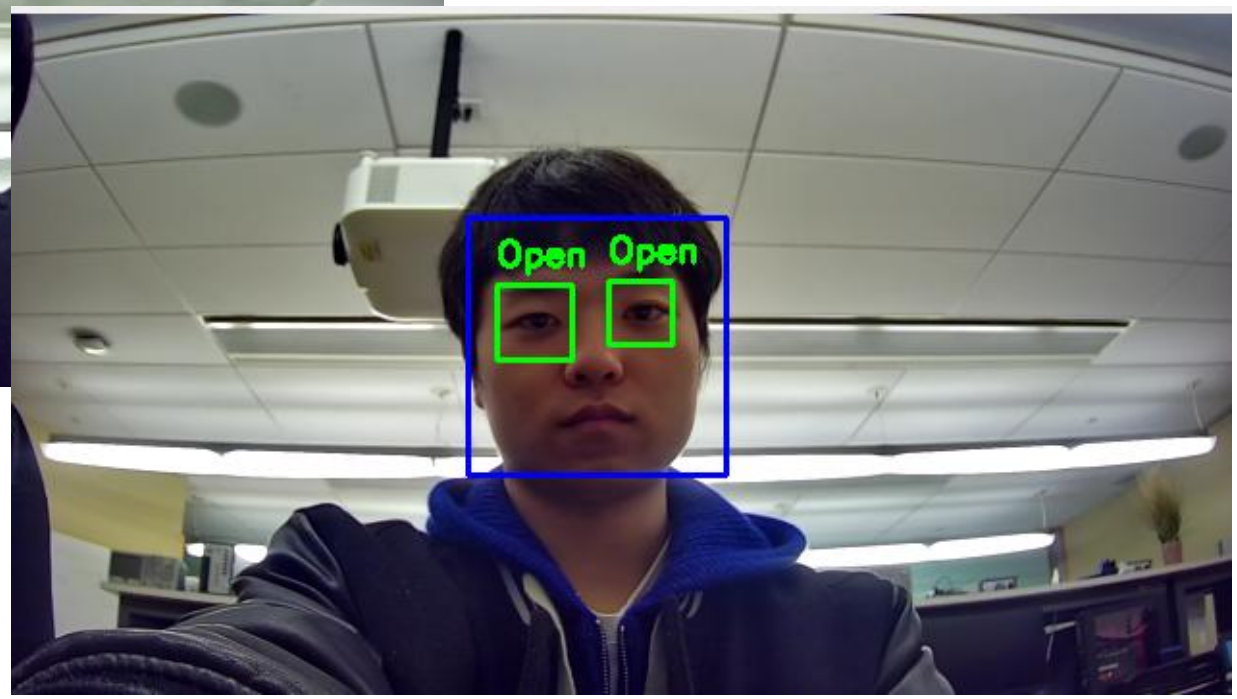
Jetson Nano —— A small, low-power device designed for edge AI tasks.

MobileNet

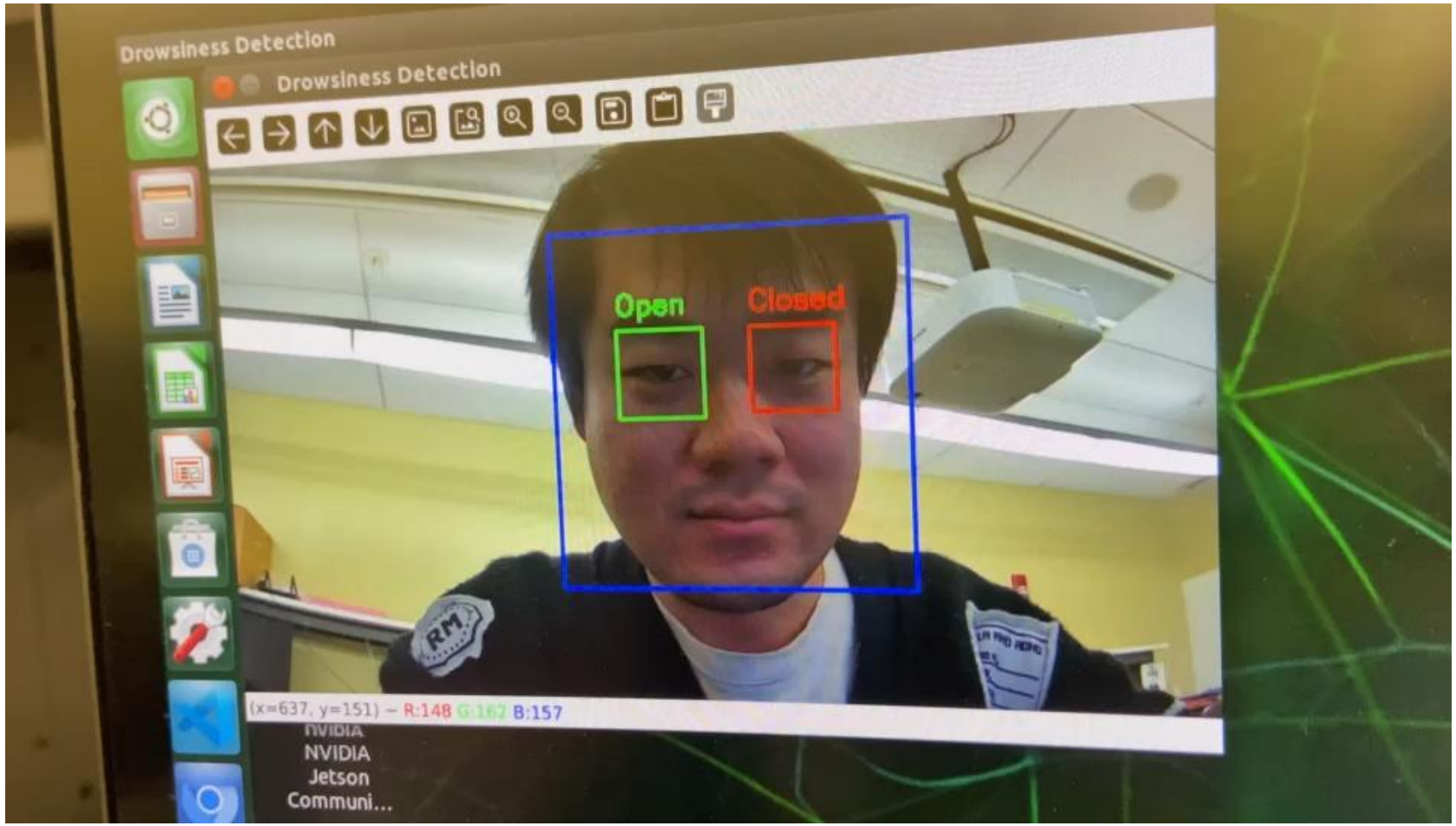
1. Uses fewer computations and memory compared to ResNet or EfficientNet.
2. Built for mobile and embedded devices, while ResNet and EfficientNet require more resources.
3. MobileNet processes images quickly, which is perfect for tasks like real-time object detection.
4. MobileNet models are smaller and can be adjusted to balance accuracy and speed.

Demo





Demo


Drowsiness Detection
Drowsiness Detection
Open
Closed
(x=637, y=151) ~ R:148 G:162 B:157
NVIDIA
NVIDIA
Jetson
Communi...

Conclusion

1. There are certain limitations in Jetson Nano performance such as its GPU computing power and memory

2. When real-time requirements are very high or resources are very limited, consider using MobileNet.

Lessons and Optimization

1.Insufficient data augmentation may cause the model to perform poorly for model-specific scenarios (such as light changes or rapid eye opening and closing).

——Increase the diversity of the training set:

(1). Including data for different lighting conditions (angles, blur, etc.).

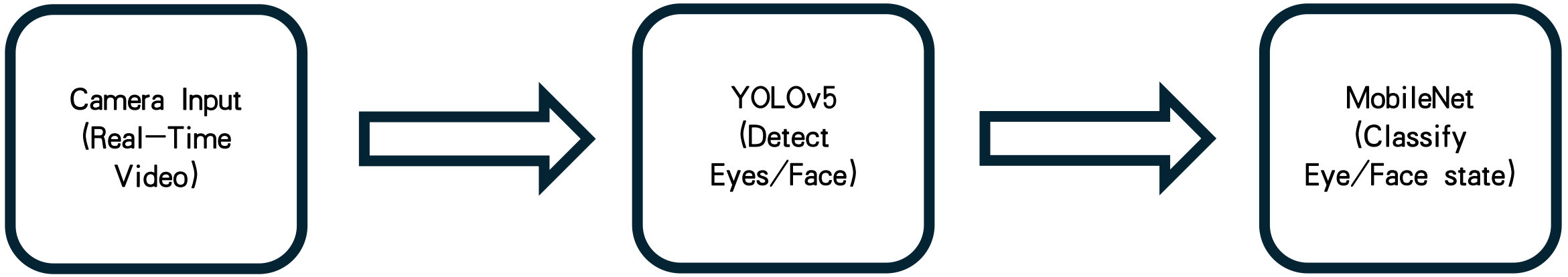
(2). Apply data augmentation techniques (rotation, brightness adjustment, blurring, etc.) to improve model robustness

2. Judging from the fluctuations in training and validation losses of ResNet18, the model may be overfitting, especially in epoch 5. Model overfitting may cause the training effect to be out of touch with the actual application

YOLOv5

YOLO (You Only Look Once) v5 is a lightweight object detection model created by Ultralytics

A target detection algorithm that focuses on real-time detection of the location and category of multiple objects in images and videos.
Unlike MobileNet, YOLOv5 focuses more on locating and labeling objects.



http://venividiwiki.ee.virginia.edu/mediawiki/index.php/JetsonNano

https://docs.edgeimpulse.com/experts/featured-machine-learning-projects/nvidia-omniverse-synthetic-data

Questions?