Slide 1 (Thao):

Hi everyone,

I'm excited to present the work of **Group 7**'s project on **'Traffic Vehicle Detection**' This project demonstrates how we applied deep learning techniques to solve real-world problems in traffic monitoring and object detection systems."

Our team consists of: Harry, Nevin, and myself, Thao

Slide 2 (Thao): Introduction

Our project focuses on designing a system capable of detecting transport vehicles and number plates in traffic scenes. To achieve this, we utilize CNNs for classification and YOLOv5 for object detection, we aim to achieve accurate results in identifying various objects.

Our motivation comes from challenges like improving traffic management, assisting law enforcement, and advancing autonomous driving technologies.

Potential applications of this system include:

- monitoring traffic patterns to optimize city planning and congestion management
- detecting law violations through number plate recognition,
- and ensuring safe navigation for self-driving cars.

Slide 3 (Thao): About Object Detection

When it comes to object detection, there are two main approaches:

a) In a **single-stage object detector**, we go directly from the image to classification and bounding box coordinates in a single step. The images are passed through a CNN and then the extracted features are directly used for classification and bounding box regression.
   Examples are **YOLO** family, **SSD**, **RetinaNet**

b) In the other hand, the **two-stage object detector** divides into 2 steps:
   - First, they identify regions of interest called **object proposals** and

- then the classification and localization happen only on the object proposals. Examples are the **R-CNN** family.

We chose **YOLO** because it's faster and easier to use. Unlike two-stage detectors, YOLO processes the entire image in one go, which is essential for real-time applications like traffic monitoring.

How YOLO Works

YOLO uses a grid-based system to analyze images:

1. **Grid Division**: The image is divided into a grid, with each cell responsible for predicting whether an object is present (probability close to 1) or not (probability close to 0).
2. **Class Probability Map**: For cells containing objects, YOLO predicts the class probabilities, helping narrow down the object's exact location.
3. **Bounding Box Prediction**: YOLO calculates the precise bounding box for each detected object using the grid cell's coordinates.

Old script:
In a single-stage object detector, we go directly from the image to classification and bounding box coordinates. The images are fed into a feature extractor using a CNN and then the extracted features are directly used for classification and regression for bounding box coordinates. Examples are YOLO family, SSD, RetinaNet
In the other hand, the two-stage object detector divides into 2 steps: extracts a series of regions of interest called object proposals and then the classification and localization happen only on the object proposals. Examples are R-CNN family.
One-stage brings us the faster speed and ease of use and that's why we go with YOLO model.
YOLO family uses a grid system to divide images into each cell with its own probability of 0 and 1 that the object will appear on the cell or not. Then it will make the class probability map to even narrow the correct area of the object.

Slide 4 (Thao): Dataset

Our dataset is sourced from Kaggle and includes over 1000 labeled images of traffic, vehicles and number plates, and CCTV footage videos across seven vehicle classes.

Images were labeled using LabelImg, a graphical tool for drawing bounding boxes around objects in the image and saving them in YOLOv5 format

Labels are stored in .txt files, with one file per image

For example, take a look at the label file corresponding to the image contains 2 trucks and a plate:

- The file provides the information required to draw bounding boxes on the image.
- Each file includes a line for every detected object, following the format: <class_id> <x_center> <y_center> <width> <height>
- Here we have 3 lines for 3 detected objects
    - The first line represents a number plate (Class ID = 1), the bounding box is highlighted in green
    - The second and third lines represent trucks (Class ID = 6) with their respective coordinates and bounding box dimensions, highlighted in orange

The bounding boxes / Box coordinates must be in normalized / scaled between 0 and 1.

- Format: <class_id> <x_center> <y_center> <width> <height>
    - The class column specifies the object type, while the remaining four values are normalized coordinates for the bounding box:
    - x_center and y_center represent the center of the box as a fraction of the image width and height.
    - width and height define the size of the box as a fraction of the image dimensions.

Note: Box coordinates must be in normalized xywh format (from 0 to 1).

Slide 5 (Harry): Data Preparation

The first thing we need to do is prepare our dataset. By using the online computer vision tools such as ~~Roboflow~~ **LabelImg** to create bounding boxes and labelling for each of your image. For example, after taking 1k images of the 401 traffic and upload it to the online tools, it will generate for us the label.txt file of each image and the .yaml for the model training. We can also do this step manually and we then have to organize the structure of the directories in the predefined format. The label file and the image must have the same name and separated by Label/Image folder name.

Slide 6 (Harry): YOLO Architecture

Next, we will go over the architecture of the YOLOv5 model. The model's network architecture consists of three parts: Backbone, Neck, and Head. Starts with the input layer, it will take the image and rescale it to 640x640 but not changing the ratio because it will take the longest edge and scale of that. The **backbone** extracts features from the input image and the main structure is multiple **CBS** (Conv + BatchNorm + SiLU - Sigmoid Linear Unit) modules and C3 modules. **C3** module is the most important here and the idea comes from Cross-Stage Partial Network. **CSPNet** is a CNN architecture that improves the learning ability and efficiency by reducing computation while increasing gradient combinations.

The **Neck** is like connections between backbone and the head, focusing on fusing features from different scales to give better detection. Yolov5 uses the method of FPN - Feature Pyramid Network and PAN - Path Aggregation Network. In FPN, information flows mainly from top-down. PANet adds an additional bottom-up path to shorten the information path. Last is the **Head** that outputs predictions for bounding boxes, object scores, and class probabilities.

(No read) Note:

Conv2D: Standard convolutional layers. **BatchNorm2D**: Normalization for faster convergence. **Leaky ReLU/SiLU**: Activation functions.

Slide 7 (Harry): Traffic Prediction

After we detect multiple vehicles on the input, we use the area comparison algorithm to predict the traffic. First, we define the **Region of Interest** specifically the street, the highway, the main intersection and calculate the total area by pixels. Then for each of the vehicle, we will calculate its area based on the x, y center and the width, heigh. Finally, we will combine to get the total area and compare it to the region of interest area.

Slide 8 (): Result

Slide 9 (): Challenges and Limitations

Code Script:

#download the data

We will import the dataset directly from Kagglehub. This dataset already has the directory organized.

#yolov5 clone

We then clone the Yolov5 model to our Google Collab and install its dependencies.

#create the dataset with train and validate data

We also need to create a .yaml file for declaring the name class, the train and val directory.

#train the model

Then we will train the model using our custom data but not the trained one. The epochs will be 20, batch is 16 and we use caching of ram and disk for faster speed. It will take 20 mins if use GPU and 1 hour for CPU

(Skip the #save the trained weights – Just to zip the model and download for future use)

#model result visualization

Now we will read our result after the model train. As you can see the loss of train and val is significant reduced after 20 epochs.

#val_result

This is the confusion matrix that will illustrate the actual value and the predicted value confidence.

#load the trained model

Now we have the best weight after the training, we will use that to detect some of the examples

#video testing

The model can also detect traffic in video or real-time. Right now we will test on one video and it will rejoin all of the frames back to the one whole video

#traffic

This is where our algorithm to detect the traffic happen. We will use a for loop to every frame of the video and send each frame to the model then get the results of coordinator to calculate total vehicle area and compare to the region of interest.