**BDA602 - Machine Learning Assignment 2**

**Compare software/hardware performance of object detection performance using the Yolo model based on the Kneron Laboratory Book**

**Introduction:**

**Object detection** is a computer vision task that involves identifying and locating objects within images or videos. It plays a pivotal role in various applications, including surveillance, autonomous vehicles, and industrial automation. Object detection utilizes neural network models like YOLO (You Only Look Once) to accurately recognize and delineate objects in real-time. There are also USB hardwares like the Kneron USB Dongle (KL520 and KL720). This essay provides a detailed comparison between the object detection capabilities of YOLOv3 software and Kneron's Edge AI hardware KL520, examining aspects such as accuracy, speed, and the ability to overcome challenges.

**YOLOv3 Software for Object Detection in Various Applications:**

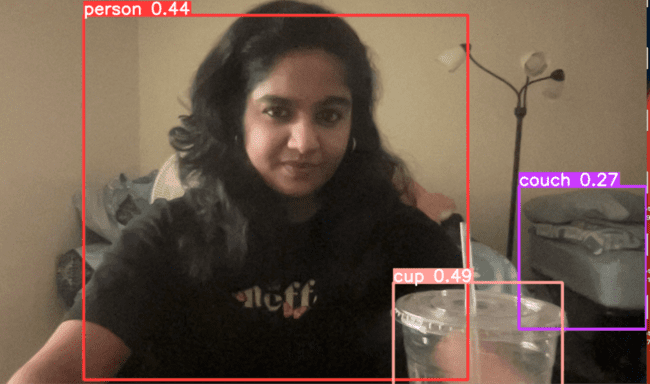
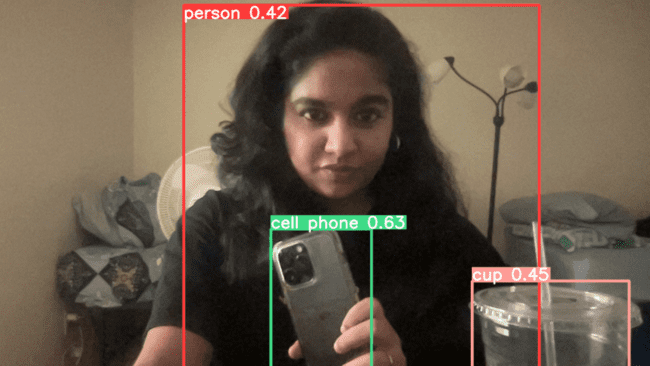
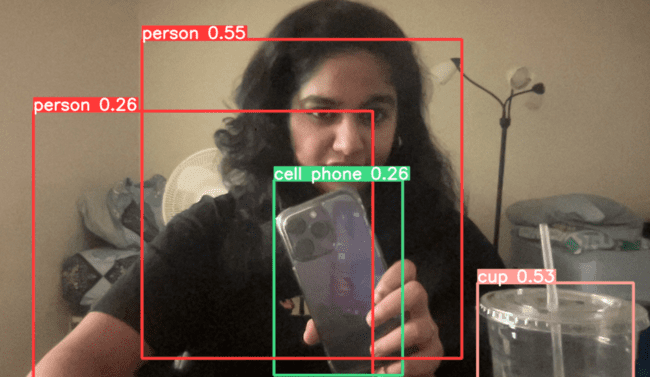
**You Only Look Once**, or YOLO, is a widely popular **object detection neural network model** used in the field of machine vision. YOLO's ability to accurately detect and locate objects within images or videos has made it a valuable tool in applications ranging from surveillance and autonomous vehicles to industrial automation and more. YOLOv3, particularly when implemented with the **Ultralytics framework**, offers a versatile platform for object detection with support for **different input sources** such as **webcams, images, videos**, and even online content like **YouTube**. In this essay, we will explore the utilization of YOLOv3 for object detection in various scenarios, including webcam, image, and image datasets, while discussing its capabilities and limitations.

**Webcam Object Detection:**

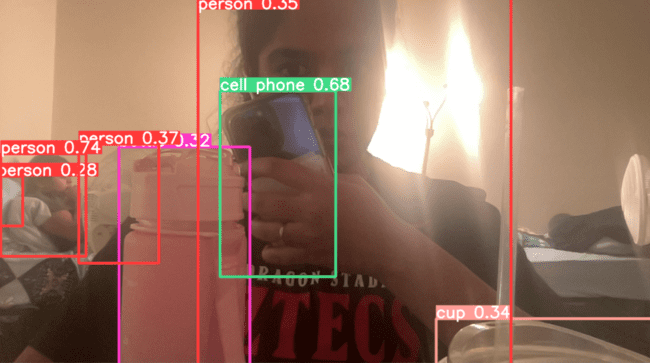
One of the most impressive aspects of **YOLOv3** is its real-time object detection capabilities. Using a webcam as the input source, YOLOv3 can instantly analyze the live feed from the camera, making it suitable for applications like surveillance systems, where real-time monitoring is crucial. To perform object detection using a webcam, one can use the following command:

**python detect.py --source 0 --device cpu**

This is particularly useful when GPU resources are limited or when a more power-efficient approach is desired.



Similarly, the below images are the results of **YOLOv5** using the input source 0 (**webcam**). We can see that it tries to detect most of the objects in the frame when compared to YOLOv3.



**Image Object Detection:**

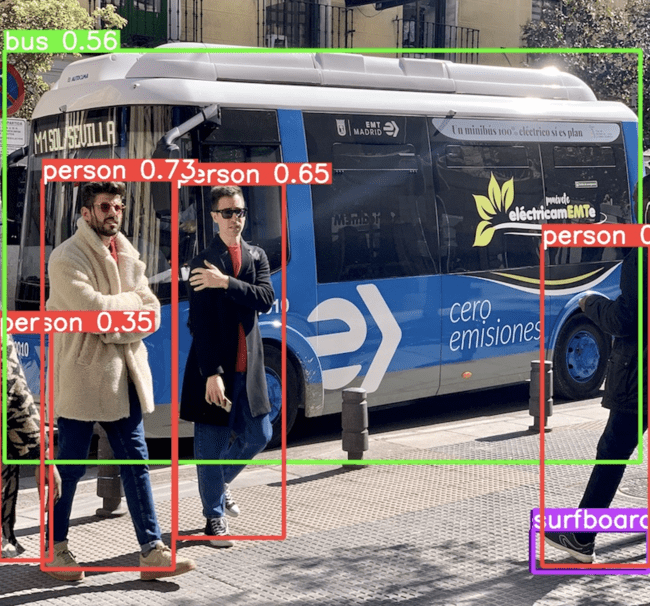
YOLOv3 also excels in detecting objects in **static images**. By simply providing an image file as the source, YOLOv3 can accurately identify and locate objects within the image. For example, to detect objects in an image named "bus.jpg," one can use the following command:

**python detect.py --source data/images/bus.jpg**

The result of this operation will be stored in a directory within the YOLOv3 installation path, where detailed information about the detected objects, including their labels and bounding boxes, can be found.

Object detection of the images found in the data/images/bus.png folder in the yolov3 directory:

Example 1:





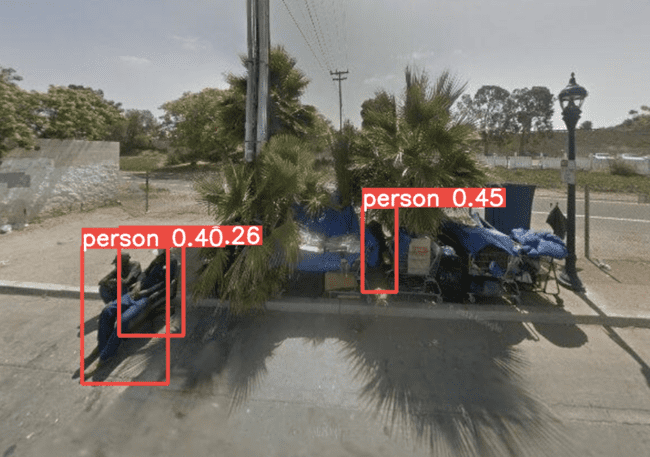
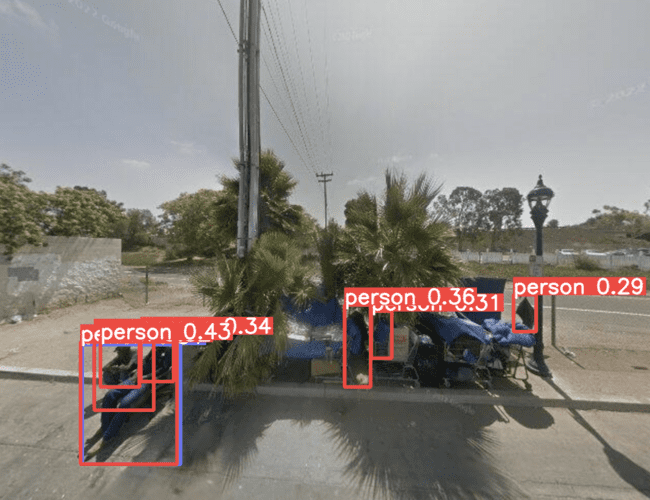
**Object Detection of images (person, tent.png) collected for my research work:**

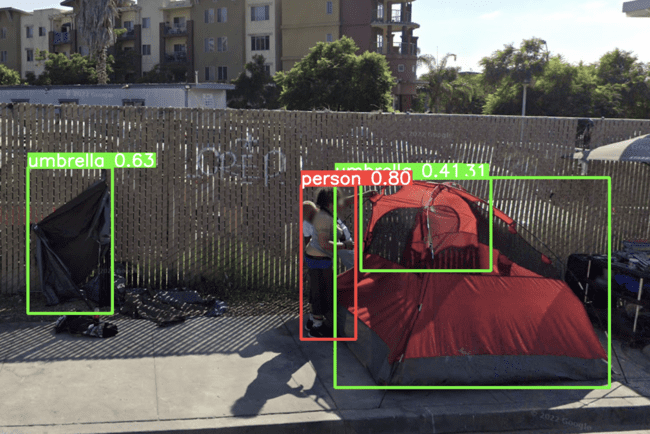
While YOLOv3 is known for its impressive accuracy in detecting common objects such as cars and people, its performance might vary when it comes to less common or custom object categories. To enhance its ability to detect objects not present in its pre-trained models, YOLOv3 can be fine-tuned with custom datasets.

In my case, if YOLOv3 struggled to detect objects like tents or buildings accurately, it may have been due to the model's lack of exposure to such objects during its training phase. The solution to this is to gather a dataset of images containing the objects I want to detect and train the YOLOv3 model with this custom dataset. Training a custom model can significantly improve detection accuracy for specific object categories.

Example 2:

**YOLOv3 Result YOLOv5 Result**

**Image of hair dryer from google (example 3):**

However, when I used a hair dryer and scissor image from google, it was not able to detect those objects. It showed ‘no detection’ when I ran the below code. This shows that it requires training with more images (custom dataset) in addition to the COCO dataset for it to recognize and detect. But, YOLOv5 was able to recognize and detect scissors accurately with a score of 0.85 but could not detect the hairdryer.

**YOLOv3 Result YOLOv5 Result**

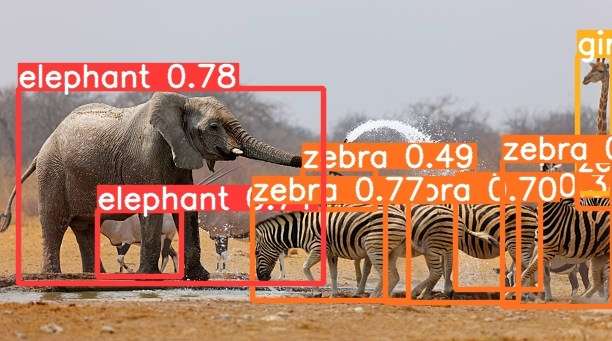
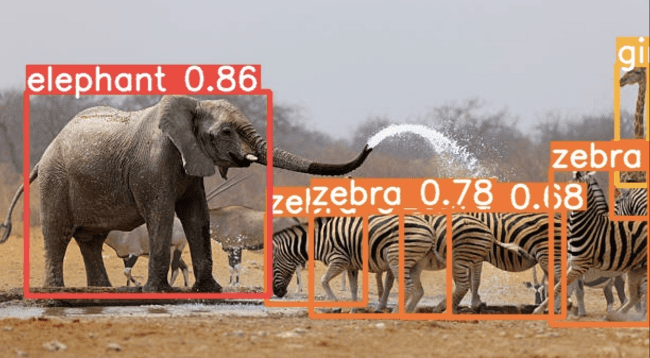
 

Example 6:







Here, it is able to detect almost all the animals in the frame. It was an accurate detection with a pre-processing speed of 1.6ms as it was able to recognize the giraffe even though it was quite hidden in the above frame. However, YOLOv5 detects almost everything with a speed of 1.4ms and from the terminal output we can see the number of animals detected as well. I also feel that there are false positives where in the above instance it detected some other animal in the background as an elephant.

**Video Object Detection:**

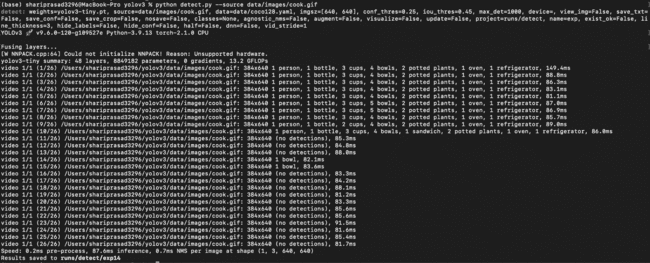
Beyond images, YOLOv3 can also handle video data effectively. By providing a video file as the input source, YOLOv3 can process the frames sequentially and generate a video output with detected objects highlighted. This is particularly useful for applications like video surveillance and autonomous vehicles, where the continuous monitoring and detection of objects are crucial.

In my case, I mentioned using a GIF as input, and both v3 and v5 provided an .mp4 video file as output, where objects were detected throughout the video. This functionality demonstrates the flexibility of these models in handling different input sources and generating meaningful outputs.

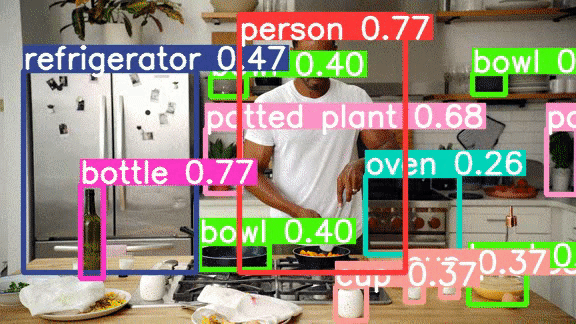
**Object Detection of a gif using Yolov3:**

Video-source:

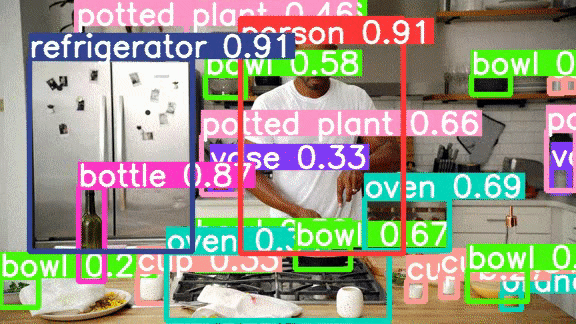
[](https://www.menshealth.com/sex-women/a28987323/the-hottest-thing-you-can-do-the-morning-after/)



**YOLO V3 Result:**



**YOLO V5 result:**



From the above videos, we can clearly say that YOLO V5 can detect more images in each frame than the YOLO V3.

However, it's important to note that the performance of YOLOv3 and v5 can vary depending on the specific object categories and the quality of the training data. Custom training can significantly enhance detection accuracy for specialized use cases. As technology continues to advance, YOLOv3, v5 and similar models will likely become even more effective in their object detection capabilities, making them indispensable tools in the field of computer vision.

In conclusion, YOLOv3 has demonstrated its utility and adaptability in object detection, and its potential for real-world applications is substantial. However, YOLOv5 has the ability to detect more objects with a longer run time. As the field of machine vision continues to evolve, YOLOv3, YOLOv5 will undoubtedly remain at the forefront of innovative solutions for object detection in a wide range of domains.

**Kneron's Edge AI Hardware and Its Impact on Object Detection**

In the realm of artificial intelligence and computer vision, the intersection of hardware and software is vital for achieving efficient and accurate object detection. With the massive growth of the Internet of Things (IoT) generating an unprecedented volume of data daily, traditional cloud-based processing faces significant challenges, especially when it comes to real-time, low-latency requirements. To address these challenges, Kneron has developed cutting-edge Edge AI hardware that complements advanced object detection models like YOLOv3 and YOLOv5. This essay will delve into Kneron's hardware solutions and then compare the software discussed in the previous prompt with this hardware.

**Edge AI Computing: The Need for Kneron's Hardware**

As the IoT continues to expand, generating a staggering 330 million terabytes of data daily, processing this data efficiently and quickly has become a daunting task for traditional cloud servers. The bandwidth and latency issues associated with transferring such vast amounts of data to remote cloud servers have led to the emergence of Edge AI computing, a paradigm that processes data close to the source, i.e., the sensors themselves. This approach significantly alleviates the data transfer bottleneck, reduces server workloads, and facilitates real-time processing.

Kneron, a leading player in the Edge AI hardware space, has positioned itself as a pioneer in object detection and speed recognition applications. Kneron's hardware offerings provide high-speed, low-power real-time processing capabilities, making it a valuable asset in scenarios where timely object detection is critical.

**Kneron's Neural Processing Unit (NPU)**

Kneron's hardware foundation lies in its Neural Processing Unit (NPU), which is designed to execute inference operations efficiently. To harness the power of the NPU, the following steps are necessary:

**Driver Installation:** Since Kneron's NPU is a unique USB device, Microsoft does not currently support it. Consequently, a USB driver must be installed to initialize the hardware. The Zadig application is employed for this purpose, ensuring that the NPU is recognized and ready for use.

**Firmware Update:** **Kneron's NPU firmware** is updated using the Kneron PLUS (Platform Library Unified Software) and Kneron DFUT (Device Flash Upgrade Toolchain). This process ensures that the **hardware** is equipped with the **latest enhancements and bug fixes**.

**Library Installation:** To fully support object detection, several Python libraries are required. This includes PyTorch, TensorFlow, OpenCV, Pillow, Matplotlib, and NumPy, each of which contributes to the robustness of the object detection pipeline.

**Yolo v3 and Yolo v5 Inference**

Kneron's hardware is seamlessly integrated with object detection models such as Yolo v3 and Yolo v5, enhancing their capabilities for real-time inference. To perform object detection using these models, users can connect Kneron hardware to their computer and invoke the corresponding Python scripts. For Yolo v3 inference, Kneron's KL-520 is utilized, while Yolo v5 inference leverages the KL-720. The results are impressive, with high accuracy and real-time processing that is critical in applications like autonomous vehicles and surveillance systems.

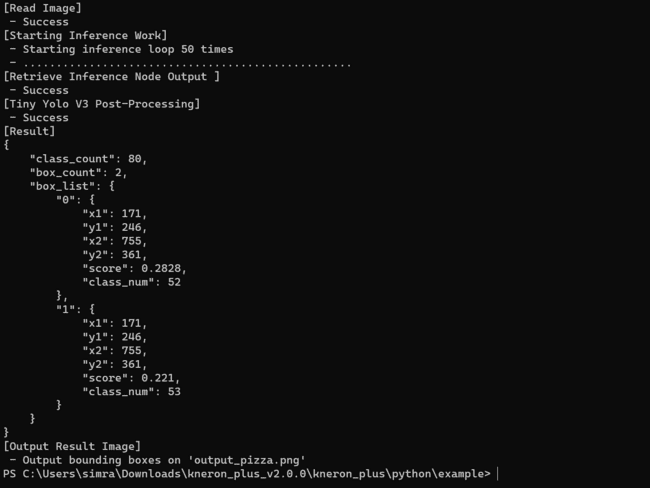
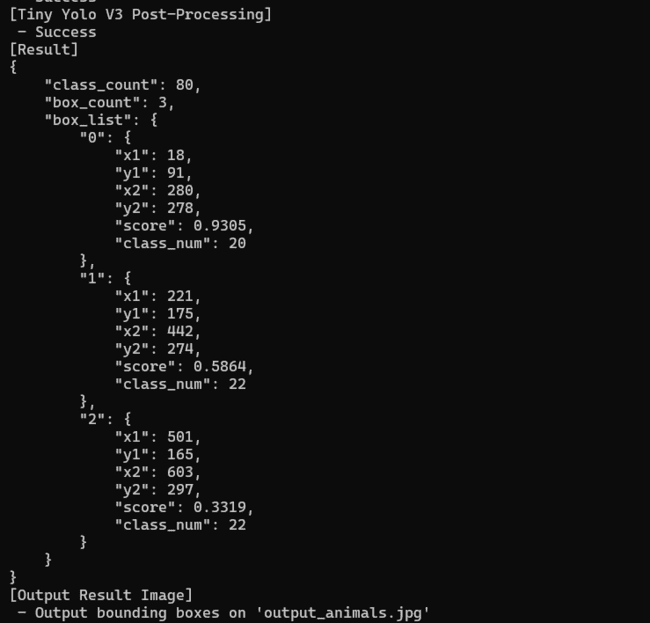
**Comparison of KL520 with Software-Only Approaches**

Object detection is a critical aspect of computer vision, and the choice of software or hardware can significantly impact its performance. Kneron's **hardware** solutions provide a significant advantage over traditional software-only approaches for object detection. While software can certainly achieve **remarkable results**, the hardware accelerators, such as Kneron's NPUs, offer **unparalleled processing speed** and **energy efficiency**. This leads to **real-time, low-latency object detection capabilities**, which are often vital in critical applications like **surveillance, autonomous vehicles**, and **industrial automation**.

Additionally, the **Edge AI computing** approach championed by **Kneron** directly addresses the challenges associated with the IoT's data deluge. By processing data at the edge, Kneron's hardware not only **relieves cloud server workloads** but also ensures that object detection can occur promptly and efficiently, even in remote or resource-constrained environments.

In conclusion, Kneron's Edge AI hardware represents a significant leap in object detection capabilities. When combined with advanced object detection models like Yolo v3 and Yolo v5, it offers high-speed, **low-power**, and **real-time processing** capabilities. This hardware approach, compared to software-only solutions, provides an edge in **accuracy, speed, and resource utilization.** As the world of IoT and AI continues to evolve, hardware solutions like Kneron's will play a pivotal role in pushing the boundaries of what's possible in object detection.

Powershell output:

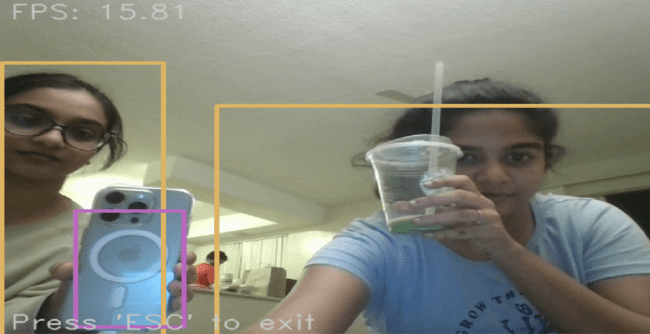


**Results generated:**





**Webcam output:**



**Notable Comparisons** between hardware and software:

**1.Webcam Detection:**

**Software (YOLOv3)**: The software-based webcam detection can **capture most objects** in a **few milliseconds** but occasionally misclassified objects. For instance, it may **mistake a table fan for a toilet or a pen for a brush**, which indicates a limitation in accuracy.

**Hardware (Kneron KL520)**: The hardware-based webcam detection **takes more time** to detect objects but excels in accurately classifying objects. While it may have some notable misclassifications, they are comparatively less frequent.

Inference: While the software offers real-time detection, the **hardware outperforms** in classification accuracy, making it a preferred choice when **precision is crucial**.

**2.Image Detection:**

The **software-based** image detection showed proficiency in **identifying common objects** like people, buses, cars, and even scissors (YOLOv5). However, it faced **challenges** in detecting **less common** items such as hairdryers and tents. On the other hand, the **hardware-based** detection demonstrated a significant ability to **identify** a **broader range of objects**, including hairdryers and tents. It's important to note that the **hardware** **occasionally misclassified** objects, like **labeling a tent as a suitcase** or a **hairdryer as a fire hydrant** but the processing speed is faster than the software.

One potential **solution** to improve accuracy is to gather diverse images of these less common objects and **train** the hardware model with a **custom dataset**. This custom training approach could help the hardware recognize and classify such objects more accurately.

In terms of **confidence scores**, the **hardware** generally **performed well** for objects it was familiar with. For instance, it achieved a high confidence score of **0.94** for detecting an **elephant**, surpassing the **software's score** of **0.84**. However, there were cases where the hardware misclassified objects. For example, it detected a pizza as both a hot dog and a pizza, each with relatively low scores (**0.2828 and 0.221**). In contrast, the **software correctly** identified the **pizza** with a score of **0.54**.

Additionally, a comparison between the hardware and software's ability to detect a car revealed interesting insights. The hardware achieved a high score of 0.92 when the entire car was visible in the frame, while the software achieved a slightly lower score of 0.89. However, in cases where only the rear end of the car was visible, the software outperformed the hardware. This suggests that **both** the hardware and software have **limitations** that can be mitigated through training with a custom dataset tailored to specific object classes of interest.

**Software:** The YOLOv3 software performed well in detecting common objects but struggled with less common ones, like tents and hair dryers, showcasing limitations in its pre-trained models.

**Hardware:** Kneron's hardware exhibited its prowess by successfully detecting a broader range of objects, including tents and hair dryers. However, it occasionally misclassified objects.

**Inference:** Kneron's hardware offers a more extensive range of object detection, especially in less common categories, though some misclassifications occur.

**Memory Utilization:**

* **Software**: **Consumes more memory** due to the processing and memory requirements of real-time object detection. For example, the image generated after detection may be larger (e.g., 92KB for animals.png).
* **Hardware**: Hardware exhibited **efficient resource utilization** after thorough comparison from all the examples that I had used, **consuming lesser memory** (e.g., 82KB for animals.png) compared to software. This attribute is crucial in resource-constrained environments, as it allows for efficient and optimal performance without excessive memory overhead.

**Adaptability:**

* **Software**: Adaptable to different environments and scenarios with software updates and model changes. Can be updated and fine-tuned as needed.
* **Hardware**: Typically less adaptable to changes and updates, as hardware is fixed and may require physical modifications for improvements.

**Speed:**

* **Software**: While offering real-time detection, exhibited a slightly **slower processing pace**, highlighting a speed advantage for the hardware.
* **Hardware**: In the comparison, it was found that the **hardware excelled in terms of speed**, particularly when the **loop time** was set to **5 or 10**. Even with the increased processing load and a loop time as short as 5 to 10, the hardware consistently detected objects faster than the software, providing results in just a few milliseconds. Running the inference process multiple times is generally done to ensure the consistency of the results. However, it also gave the same results as when the loop time was set to 50. This underlines the exceptional speed and efficiency of the hardware in real-time object detection scenarios.

**Versatility:**

* **Software**: Versatile and widely applicable, capable of handling a range of objects, but may face limitations with less common objects.
* **Hardware**: Demonstrates versatility by successfully detecting a broader range of objects, including less common items, but may have occasional misclassifications.

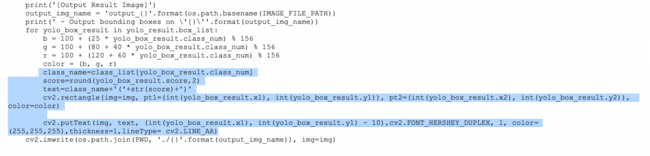
**Difficulties I faced and steps I took to overcome:**

The software part was very straight forward whereas the hardware part took some time to understand the installation of drivers, applications like zadig, and the installation of KneronDFUT and Kneron Plus and to load the correct binary files for the ncpu and scpu. The software part gave the objects classification above the bounding box with the confidence score whereas the hardware part was coded to only detect and create bounding boxes but could not display the class of the object and the confidence score. However, it displayed these contents in the command prompt where it printed the coordinates of the boxes, score and the class number.

Since a list of different classes were given in the hardware instructions manual, I created a python list in the KL520DemoGenericImageInferencePostYolo.py file.



I named the list as class\_list and used the output printed in the console where it contained coordinates, score and class num. I used this class num as an index to the class\_list and tried to fetch the class name and score and displayed them above the bounding box for a better understanding. I have attached the output images in the hardware section containing bounding boxes and class name for reference.



**Conclusion:**

In summary, the comparison between software and hardware-based object detection, using YOLOv3 software and Kneron's KL520 hardware, reveals essential trade-offs. Hardware offers faster speeds, allowing for loop time modifications, while software is adaptable and consumes more memory but may have occasional misclassifications. Hardware's memory efficiency and precision, alongside its versatility, make it an excellent choice for specific applications. The decision should align with particular requirements, considering speed, memory, adaptability, and versatility, with potential room for improvements through custom training and dataset adjustments.