Classical Computer Vision Still has its Place in Object Detection

Nick Cantalupa
Khoury College of Computer Science
Northeastern University
Boston, MA
cantalupa.n@northeastern.edu

Sean Duffy
Khoury College of Computer Science
Northeastern University
Boston, MA
duffy.se@northeastern.edu

Abstract — This project builds upon our established methods of real-time object detection. Our initial work establishes a baseline using a processing pipeline that includes binary thresholding, erosion, dilation, segmentation, and K-Nearest Neighbors matching to identify each object. To integrate deep neural networks into the program, we translated the original code from C++ to Python, improving both performance and the user interface. We then added functionality that allows users to switch between the classical object detection methods and a deep learning approach. Users can choose between a YOLOS-Tiny model and a larger DETR-ResNet-50 model. After developing the updated program, we conducted tests to compare performance between the classical method and the two neural network models. These tests evaluated object detection accuracy under varying lighting conditions and assessed each model's ability to handle object rotations. Finally, we introduced adversarial images into the system to test the robustness of each method.

INTRODUCTION

The task of object detection in the field of computer vision is one that is widely applicable to many different domains. Being able to identify and track objects in a frame is useful in a variety of situations, including robotic vision, video surveillance, autonomous vehicles, and many others. Each of these applications also has unique specifications for the task they are aiming to solve. For example, an autonomous vehicle requires a very high frame rate while traveling at high speeds and must perform effectively under various lighting conditions to allow operation at all hours. In contrast, a food defect detection system operates in a stationary environment under specific, controlled conditions and can be much more specialized in identifying subtle differences in the target product. Given such diverse applications and system requirements, it is important to understand the different methods of object detection and be familiar with their behaviors in order to choose the approach that best fits the task at hand.

This project extends upon our initial baseline program for object detection. The program follows a processing pipeline for each frame. The image is first processed using binary thresholding to separate the background from the foreground. The system is designed to work with a white background, allowing for easy identification of the object in the frame. After thresholding, the frame undergoes a cycle of erosion and dilation, which aims to close any small gaps in the object caused by noise and to smooth out the edges, resulting in a more consistent input for feature extraction. The frame is then processed using a segmentation algorithm to identify connected components and separate the different objects in the scene. Once the objects are segmented, they undergo feature extraction, which calculates the bounding box's percent fill, aspect ratio, and Hu moment values. These features are passed through a K-Nearest Neighbors algorithm to cross-reference a database and identify which object most closely resembles those in the frame. The system also allows users to add new items to the database by inputting the label of the identified items.

Building on this classical approach to object detection, we have added capabilities that allow the system to switch between the previous method and two different neural networks: YOLOS-Tiny and DETR-ResNet-50. The YOLOS model has 6.5M parameters, offering a smaller, faster alternative, while the ResNet-50 model contains 41.6M parameters. These options are intended to highlight the advantages and disadvantages of differing model complexities. Having three models with varying levels of complexity allowed us to run tests to evaluate performance in different scenarios. These tests examined how each model handles object rotations in the frame, as well as the effects of low to

high lighting conditions. We also analyzed the relative speeds of the models to assess how performance is influenced by processing time.

In addition to measuring and comparing performance in normal scenarios we were interested in evaluating how each of the detection methods performs against adversarial attacks. In order to do this, we calculated adversarial patches meant to cause misclassification of objects. The adversarial patches were calculated using ResNet50, so that in addition to seeing differences between the classical methods and deep learning methods, we may see differences between the two networks. Only DETR-ResNet-50 makes use of the ResNet50 CNN, so it should be more susceptible to the patches.

RELATED WORK

Object detection has evolved over the last decades featuring newer deep learning methods that offer superior accuracy and well generalizable systems. The transformer architectures have gained popularity due to their effectiveness at capturing the context of images. Of those transformers, the DETR (DEtection Transformer), introduced by Carion et al. from Meta marked a shift in how objects were being classified within the images. As opposed to using a region-proposal based method, Meta used the transformer as an end-to-end object detection method, eliminating the need for methods like non-maxima suppression. Using the transformer model on top of a CNN backbone, the model was successful at capturing the context of the objects and how different objects in an image can relate to each other. The time complexity of this model did present an issue for some applications, resulting in the development of some smaller models.

The YOLOS model presented by Fang et al. explores the idea of applying a standard vision transformer to the object detection world without making major changes to the architecture. The YOLOS-tiny model is a smaller version of the original designed to balance accuracy with speed and efficiency. Despite the simplicity of the model, the YOLOS models were able to achieve successful scores on the COCO image datasets. Our work aims to highlight the tradeoffs between larger and more lightweight models in terms of speed, efficiency, and robustness to adapt to the task required.

Adversarial attacks are well studied, with multiple techniques for altering inputs to computer vision systems such that they are not able to perform as well as intended. These techniques often exploit the high dimensional nature of the latent space to “move” inputs across decision boundaries. In many cases this results in changes to the input that are imperceptible to a human but can effectively cause classification errors. Brown et al. took an alternative approach, focusing on adding an image that was so visually salient a neural network would classify a proximal object as the patch’s class. Instead of learning an alteration to a specific input such that it is misclassified, a patch is learned that can cause misclassification when placed over multiple different inputs.

METHODS

Classical Object Detection

In the original C++ implementation, all functions in the object detection pipeline were written from scratch (with the exception of connected component identification). In order to improve the efficiency, when translating the program to Python, OpenCV functions were used where possible. The final processing steps for object detection were: bilateral filtering to denoise while preserving edges, otsu thresholding, morphological closing, connected component identification, feature extraction, and matching. Filtering is applied to the connected components such that only components within a certain range of areas are considered. The features extracted from the filtered components are aspect ratio, percent fill, and all 7 Hu moments (rotation, scale, and translation) invariant moments. As objects are labelled in training mode, their label and features are stored in a database. The normalized database is stored in memory and objects are matched via KNN with K=3. In detection mode (not training), only objects in the database are detected. This is achieved by limiting matching to objects with a NN distance below a threshold.

Deep Learning Approaches

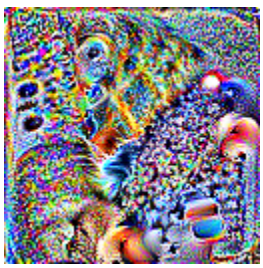
Our deep learning method for real time object recognition was implemented in Python using Hugging Face models. The two models that we used from Hugging Face are the facebook/detr-resnet-50 model and the hustvl/yolos-tiny model. These models have 41.6M and 6.5M parameters respectively so they give a good comparison of how a smaller and a more substantial model could perform when put into a live processing application. The ResNet50 model was trained on the COCO 2017 dataset which has 80 different classes for object recognition. The YOLOS model was trained on ImageNet1k and then fine-tuned on COCO 2017 so it too is able to detect the 80 categories from the dataset.

Our application first preprocesses the frame by converting it from a BGR image format to an RGB format. It then feeds that image into the respective image processor associated with the model. The model outputs the classes of the objects detected along with the screen coordinates of the bounding box. We then use the Pillow ImageDraw Python package to draw the bounding box and add the label. This image gets converted back to BGR and displayed.

Adversarial Patches

The adversarial patch was calculated as in Brown et al. The patch was initialized as 128x128 random noise and was applied on 60 different example images during training. The 60 images contain 5 different orientations of 12 different objects against a white background as they may be seen in the evaluation of the classical object detection. It's important that the training images resemble the expected evaluation environment so the calculated patch will be as effective as possible. For each patch a selected misclassification target is identified, we sought classes that were present in both ImageNet and COCO such that it could both be trained on ResNet50 and identified on the COCO-trained object detection networks. The targets we used were "passenger car" and "remote".

During the training process the patch is applied at a random location on the 224x224 input image, with random rotation between -30 and 30 degrees, and with random scale between 80% and 120%. These alterations make the patch more robust in real world application where it may not be captured by the camera at the ideal scale or orientation. After the forward pass through the network, cross entropy loss is calculated against the target class (misclassification), and the patch is updated with respect to the gradient of the loss. The car patch was trained for 100 epochs and the remote patch for 250 epochs, both with a batch size of 10.

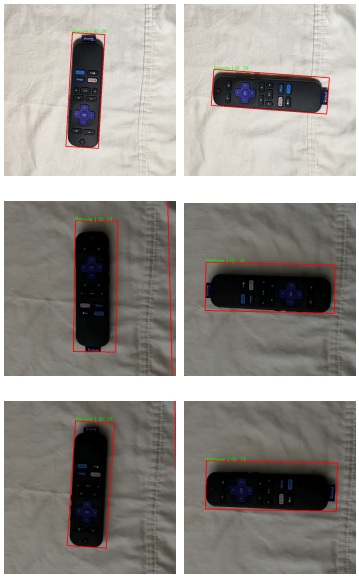


Adversarial Car Patch Adversarial Remote Patch

The patches were evaluated both physically and digitally. In addition to printing the patches, functionality was added to the object detection system to overlay them on the video frames. By clicking on the video feed window, you can show the patch and cycle through them. The patch can be moved within the frame and scaled using the "-" and "+" keys.

EXPERIMENTS & RESULTS

Our focus for testing the performance of the different models focus on three areas of measurement; performance in high versus low light scenes, tracking 360 degree rotations, and speed of the model. All of the tests are performed using a mounted iPhone camera aimed down upon a scene with a white background. This scenario is used to allow for the classical object detection model to threshold correctly. The 9 different objects were tested through each of the scenarios. The 9 objects tested include a book, a cell phone, a fork, a knife, a computer mouse, a TV remote, a pair of scissors, a spoon, and a toothbrush. The objects were selected from the list of categories in the COCO 2017 dataset that the neural network models were fine-tuned on. The performance was determined on a binary scale, 1 if the model could successfully identify the correct object class, and 0 if it gave the incorrect object class or failed to identify a bounding box. To test the model performance in different lighting, the objects were placed in the frame with a single dim lamp, aimed toward the ceiling to only provide minimal ambient lighting. They were then tested again using a direct white light source aimed at the object. To test the rotational accuracy of the model, each item was observed 4 times, at 0 degree rotation, 90 degree rotation, 180 degree rotation, and 270 degree rotation. This was done in both the light and dark conditions. The images below show samples of the images being tested using a TV remote in the 8 different conditions.



Eight different situations that all objects were tested under, first four in well lit environment and last four in dark environment

The speed of the models is reported in frames per second. The speed test was run using a 2019 Macbook Pro. Below are the results for both tests.

TABLE I

Model	*Total Accuracy*	*Dark Accuracy*	*Light Accuracy*	*Percent Difference*
Classical	58.33	41.67	75.00	+33.33
ResNet50	**79.17**	**80.56**	**77.78**	-2.78
YOLOS	45.83	44.44	47.22	+2.78

The object detection accuracy measurements showed that the ResNet50 model was the most accurate model across both light and dark situations and all rotations. The classical method performed the second best overall and had a very comparable score to the ResNet50 model in the light environments. The YOLOS model performed the worst overall, only having a slight advantage in the dark situations over the classical method. A big point that should be noticed here is that the neural network model both had negligible differences between the dark and light performances, while the classical method had a stark 33 point jump from dark to light. This highlights how the classical model is designed for high contrast inputs that will perform well when put through the thresholding processes. Items that are reflective such as the fork and knife blade have a difficult time getting through the thresholding and getting identified in the classical system.

TABLE II

Model	*Frames Per Second*
Classical	3.88
ResNet50	0.56
YOLOS	2.03

The complexity of the model was observed to play a big role in how fast the program was able to run and how many frames per second it was able to output. The classical model easily performed the fastest of those tested, with speeds 6.9x that of the ResNet50 model and 1.9x the YOLOS model. The classical method here still shows that it can be a very viable method for certain tasks when higher speeds and more frames are required. These results also show how much the size of the neural network matters to the speed and performance. The YOLOS model was 3.6x faster than the ResNet50 model, however the large dropoff in accuracy may not be worth the speedup. These models were chosen as extremes in terms of model size for local real time object detection so a middle ground model between 6.5M and 41.6M parameters could be the optimal choice based on the required task.

Adversarial Attack

The adversarial patches were not an effective attack against our deep learning system. The deep learning methods were unaffected by both the physical and digital attacks. Notably, the patches did not seem to be very salient. In the absence of another object, it would be expected that the patch itself would be identified as its target class. This did not occur.

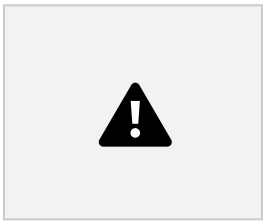
Scissors and patch Patch attack on scissors

The adversarial attacks were similarly ineffective against classical object detection. In cases where the application of the patch did not affect the outline of the object, the detected classes remained unchanged. Placing the patch at the edge of the object could successfully cause misclassification or lack of identification, but this was more an effect of changing the shape of the connected component than

the actual content of the patch itself. This is shown with a hockey puck, which is as effective as the patch in causing detection errors of a dumbbell. This shows that the weakness to adversarial patches is only a result of the known limitations.

Patch causing identification error when placed too close

Successful identification when patch does not change the shape of the object



Puck and dumbbell successfully identified

Puck and dumbbell causing identification error when placed too close

DISCUSSION & CONCLUSION

Our comparison between the classical methods of real time object detection and the newer neural network based methods highlight some of the advantages and disadvantages that both methods bring with them in their implementations as one builds a live processing system. Our classical method being translated from C++ to Python allowed us to take advantage of some of the Python libraries that streamline the code and make the program a more usable system. Some aspects of the project that the classical method excels at are the speed, due to its simplicity, and its simple process for adding new items to the database. The program we implemented allows you to run the classical method in training mode where you can use a keystroke to freeze the frame and label an object detected in the frame. This will add that category as a new class in the system that can be identified by the system. This means that the user can add new training data and update the system with new classes in a matter of seconds. The training data in this situation will be specialized to specific scenes that the algorithm is being used for. This means that it can be highly specialized for the required task and easily adaptable as new classes and categories are needed. Contrast this with the neural networks that are fine-tuned to identify specific classes using, one would need to again fine-tune the model whenever a class is needed. That said, the classical system does come with some drawbacks. The classical system may excel in ideal situations, but it then is not generalizable and broadly adaptable. The system requires a consistent background color to contrast the item. The current system struggled to distinguish reflective surfaces such as the silverware or scissors. The classical method can perform well in more controlled settings and you can design the system to fit exactly the type of object recognition that is needed.

The neural network methods we examined in this project showed that these models can be very good at generalizing to a wide variety of tasks. Both of the models were able to be integrated into the existing program we had established and produce fair to great results. Since they were both trained on ImageNet and fine-tuned on COCO datasets, they are able to extract features from scenes and capture some of the semantics of the objects being detected instead of strictly making decisions based on shape and size. This also makes them strong in scenes that have differing light conditions. The results showed that there is negligible difference between the light and dark tests for the both of the neural network methods. As long as the camera is able to pick out most of the object colors then it is able to run effectively. The speed of these models are a major drawback. In order to get good results, the much larger model had to be used which reduced the framerate to an almost unusable speed. This speed of one frame every two seconds would only be effective for applications that can wait for the program to respond. Any motion in the scene will throw off the detection. The YOLOS model was able to run faster but at the cost of a huge performance drop off. Any application where real time object detection is being used, there are going to be many different decisions to be made that will tailor the program to the specific task. The results from this testing show how neural network applications are not always going to be the best solution. There are many scenarios where a simpler system with classical computer vision methods could be exactly the right solution to the problem.

Further experimentation would be needed in order to fully evaluate the robustness of our deep learning system to adversarial attacks. The lack of salience of the patches may indicate that they have not been adequately calculated for misclassification as the target class. This may be explored with application to

more rudimentary networks than the two that we evaluated, like Fast-RCNN. It is possible that the complex architecture of our networks makes them robust to attacks. If a network like Fast-RCNN was similarly impervious to the attacks, we may need to recalculate patches. They may benefit from more example images, more training iterations, and a loss function that measures 'printability' of the colors. The classical methods would likely not be affected by further changes in the patches, as the mechanisms of that system are not the target of this adversarial patch method.

REFERENCES

Brown, T. B., Mané, D., Roy, A., Abadi, M., & Gilmer, J. (2018). Adversarial Patch. *arXiv preprint arXiv:1712.09665*. https://arxiv.org/abs/1712.09665

Fang, Y., Liao, B., Wang, X., Fang, J., Qi, J., Wu, R., Niu, J., & Liu, W. (2021). You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection. *arXiv preprint arXiv:2106.00666*. https://arxiv.org/abs/2106.00666

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*. https://arxiv.org/abs/1512.03385

Meta AI Research. (n.d.). End-to-End Object Detection with Transformers. https://ai.meta.com/research/publications/end-to-end-object-detection-with-transformers/

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *arXiv preprint arXiv:1506.02640*. https://arxiv.org/abs/1506.02640