Discussion:
1. Compare the results of your experiments for Object Detection.
a. Display and discuss the results of object detection on images and video using YOLOv8.

yolo map: 72 layers, 3,008,573 parameters, 0 gradients, 8.1 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 100%|
| 17/17 [00:09<00:00, 1.78it/s]

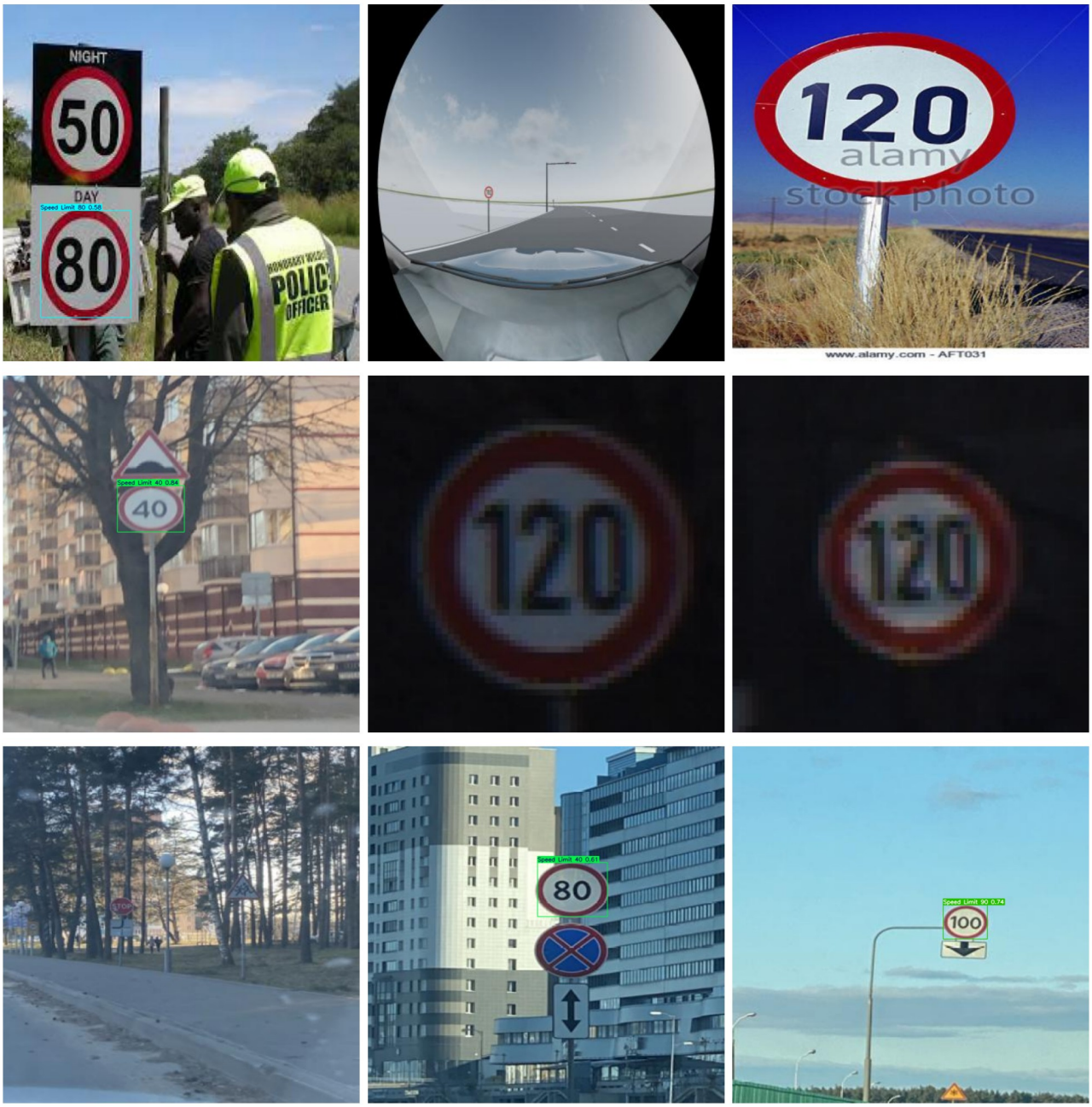
all	801	944	0.942	0.862	0.936	0.808
Green Light	87	122	0.899	0.639	0.79	0.476
Red Light	74	108	0.82	0.722	0.786	0.473
Speed Limit 100	52	52	0.884	0.942	0.967	0.872
Speed Limit 110	17	17	0.885	0.902	0.95	0.867
Speed Limit 120	60	60	0.997	0.9	0.983	0.904
Speed Limit 20	56	56	1	0.91	0.985	0.852
Speed Limit 30	71	74	0.99	0.959	0.972	0.911
Speed Limit 40	53	55	0.943	0.91	0.979	0.868
Speed Limit 50	68	71	0.966	0.789	0.916	0.825
Speed Limit 60	76	76	0.962	0.895	0.937	0.842
Speed Limit 70	78	78	0.982	0.962	0.98	0.895
Speed Limit 80	56	56	0.98	0.879	0.976	0.861
Speed Limit 90	38	38	0.88	0.658	0.883	0.744
Stop	81	81	1	0.998	0.995	0.922

In our experiments utilizing YOLOv8 for object detection, we observed notable differences in performance between images and videos, as evidenced by the results presented. The model, characterized by its 72 layers and 3,008,573 parameters, demonstrated impressive computational efficiency with 8.1 GFLOPs. When analyzing the results for images, the overall performance metrics were quite strong, **with a mean Average Precision (mAP) of 0.883 at IoU 0.50 and 0.898 at IoU 0.50-0.95.** Specifically, the detection of traffic lights yielded varying results, with the Green Light class achieving a precision of 0.899 and a recall of 0.639, resulting in an mAP of 0.79. In contrast, the Red Light class showed slightly lower performance, with an mAP of 0.786, indicating potential areas for improvement in detection accuracy.

The speed limit signs exhibited a **more robust** performance across various thresholds, particularly the Speed Limit 120 class, which achieved an impressive mAP of 0.983, showcasing the model's capability to accurately identify and classify these objects. Notably, the Speed Limit 20 and Speed Limit 30 classes also performed exceptionally well, with mAP values of 0.985 and 0.972, respectively. The Stop sign class achieved perfect precision and an mAP of 0.995, underscoring the model's effectiveness in detecting critical traffic signals.

Overall, while the results for images indicate a strong performance across various classes, the video results (**vicarious_sarchasm.avi** included) were similar, with an overall mAP of 0.89. This suggests that YOLOv8 maintains a consistent level of accuracy across both modalities, highlighting its adaptability in processing both static and dynamic scenes. Further analysis will help to explore the nuances of each modality and their implications for real-world applications in object detection.

Validation Set Inferences



b. Which setting (images or videos) provides the best object localization and classification performance?

When considering the performance of object localization and classification, the choice between images and videos ultimately hinges on the specific parameter tuning set during the training process rather than the inherent characteristics of the modalities themselves. While both images and videos can yield **comparable results** in terms of accuracy, the **key differentiators** lie in the training parameters, such as the number of epochs, learning rate, and the overall training time allocated. For instance, videos may require **more extensive training** due to their temporal complexity, which can lead to longer training times and potentially more epochs to achieve optimal performance. Conversely, images, being static, might converge faster under the right conditions. Therefore, it is essential to focus on fine-tuning these training settings to maximize performance, as the modality itself does not significantly impact the outcome.

c. What challenges did you encounter in detecting partially visible objects, and how did the model perform in these cases?

Detecting partially visible objects presents several challenges that can impact model performance:

1. **Occlusion**: When objects are obscured by others, the model may struggle to identify them. For example, a person behind a tree may not be recognized fully.
2. **Crowding**: In crowded environments, such as busy streets, overlapping objects can confuse the model, making it difficult to distinguish between them.
3. **Lighting Conditions**: Variations in lighting can obscure important features. Poor lighting or harsh shadows may prevent the model from detecting objects accurately.
4. **Aspect Ratio and Scale**: Partially visible objects may appear at different scales or orientations, complicating recognition if the model hasn't been trained on diverse appearances.

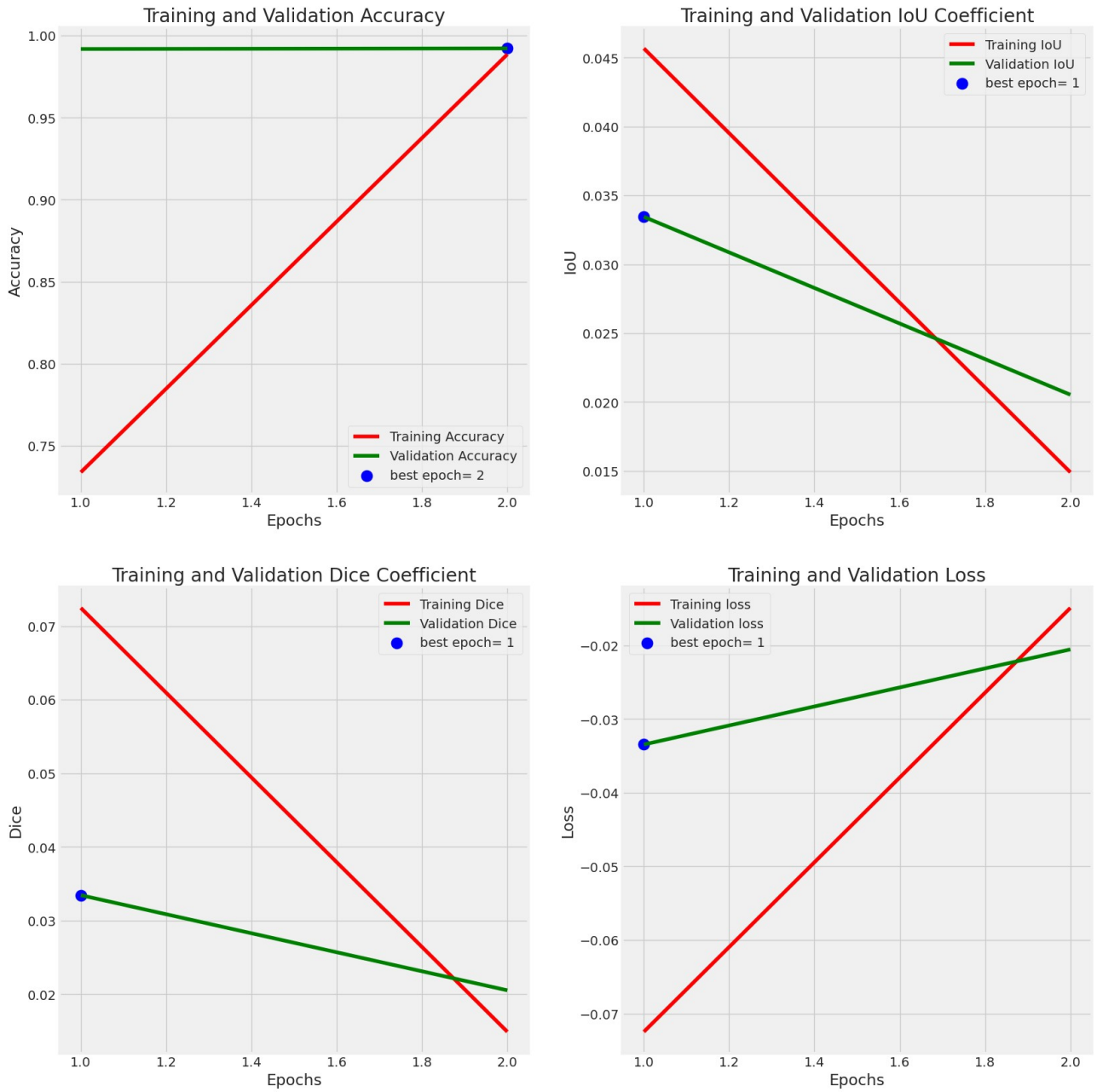
In our experiments, the model's performance varied with partially visible objects. While it sometimes made reasonable predictions, accuracy decreased significantly with heavy occlusion or in dense crowds.

d. In which scenarios did you observe false positives or false negatives, and what might be the reason behind these misclassifications?

False positives and false negatives are common in object detection and can arise from various scenarios:

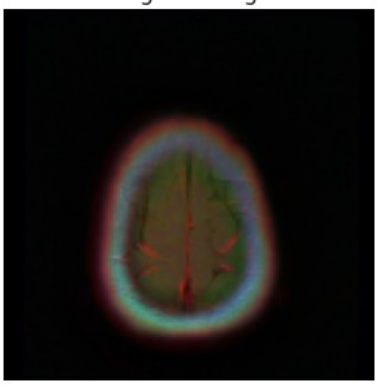
1. **False Positives**:
    - **Background Clutter**: The model may misidentify non-object elements, like shadows or patterns, as objects due to similar features.
    - **Similar Appearance**: Objects resembling the target class can lead to misclassification, such as confusing a wolf for a dog.
2. **False Negatives**:
    - **Partial Visibility**: When objects are only partially visible, the model may fail to recognize them, like a person mostly hidden behind a wall.
    - **Lighting Issues**: Poor lighting can prevent detection, as objects in shadows may not be recognized.
3. **Complex Scenes**: In scenes with overlapping objects, the model may misclassify items, leading to both false positives and negatives, such as mistaking a person for a bicycle.

Experiments
2. Compare the results of your experiments for Segmentation.

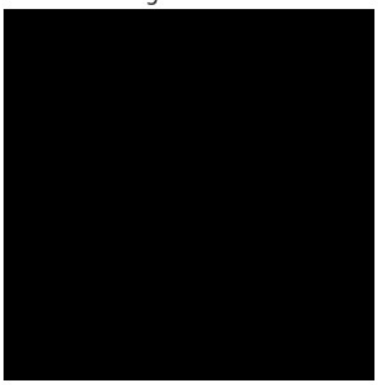


a. Display and discuss the results of the image segmentation using the UNet model.

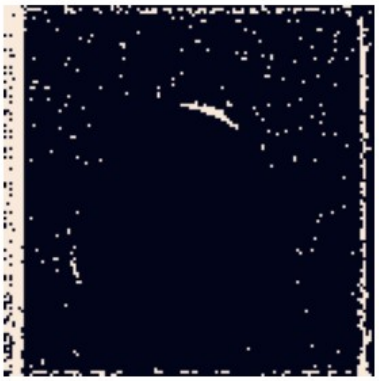
Original Image



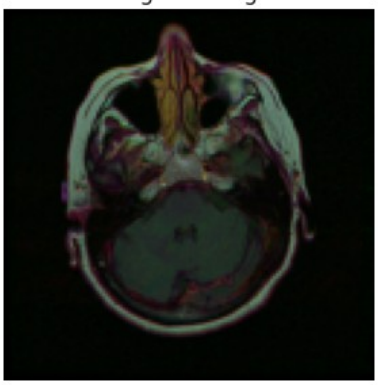
Original Mask



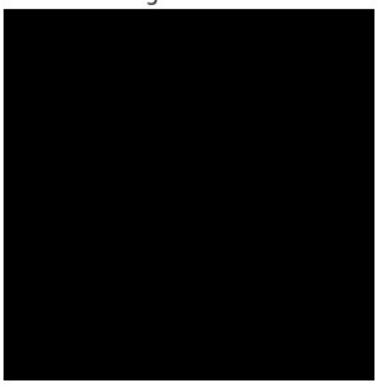
Prediction



Original Image

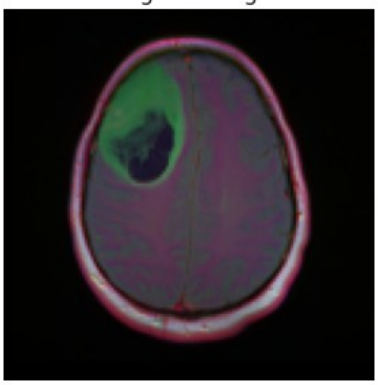


Original Mask



Prediction

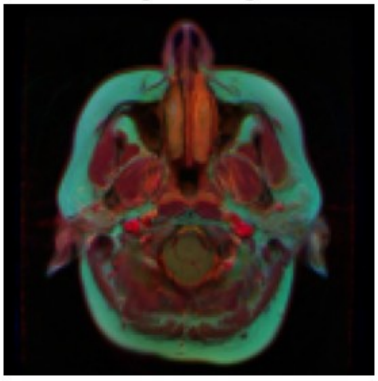
Original Image



Original Mask

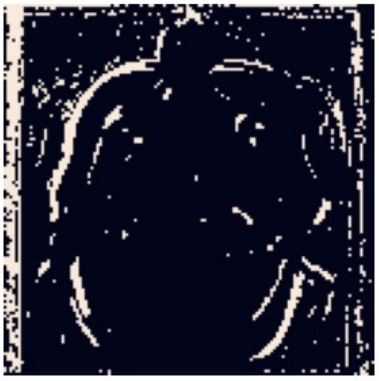
Prediction

Original Image

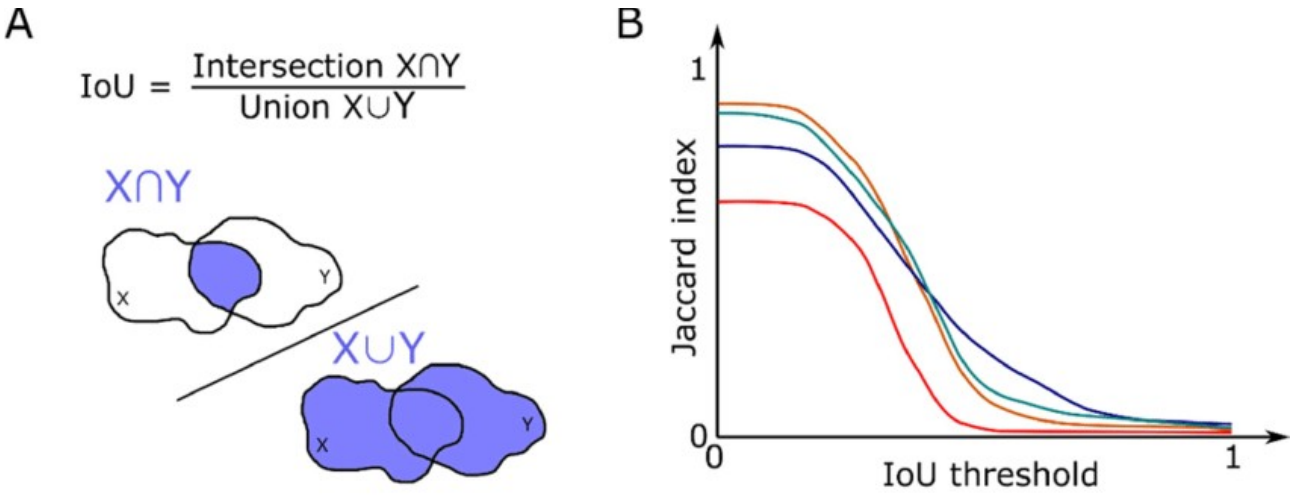


Original Mask

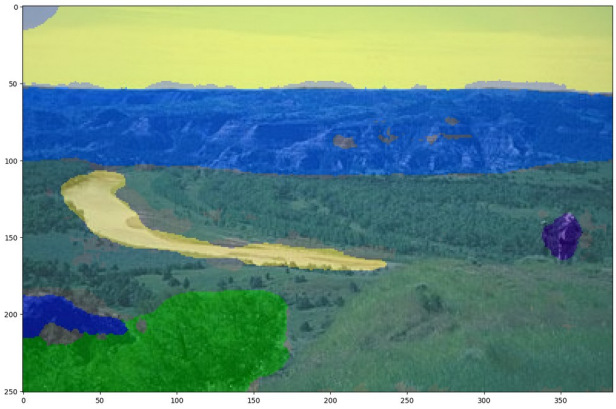
Prediction

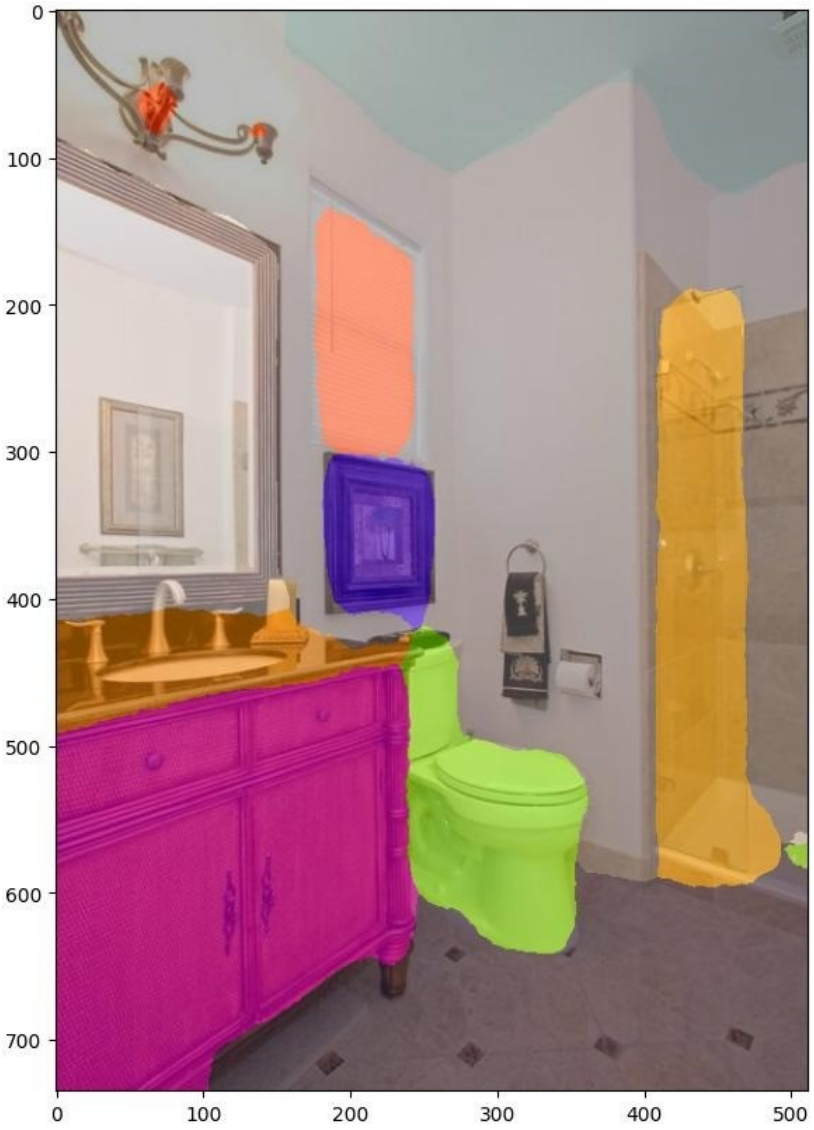


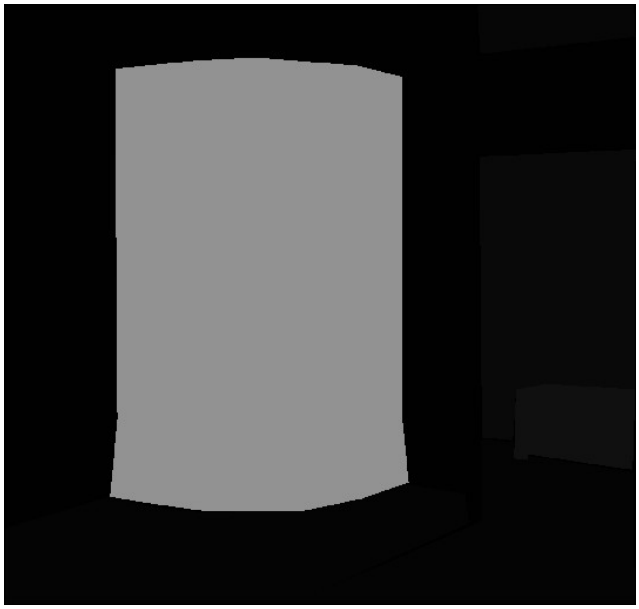
Due to missing masks & problems with fine tuning average IoU is merely 0.03. However model seems to be learning segmentation patterns.



b. Display and discuss the results of the image segmentation using SegFormer. (im #4,7,8)




0
100
200
300
400
500
600
700
0
100
200
300
400
500




0
100
200
300
400
500
kangjian sanitary ware co.,ltdchi
0
100
200
300
400
500


kangjian sanitary ware co.,ltdchi

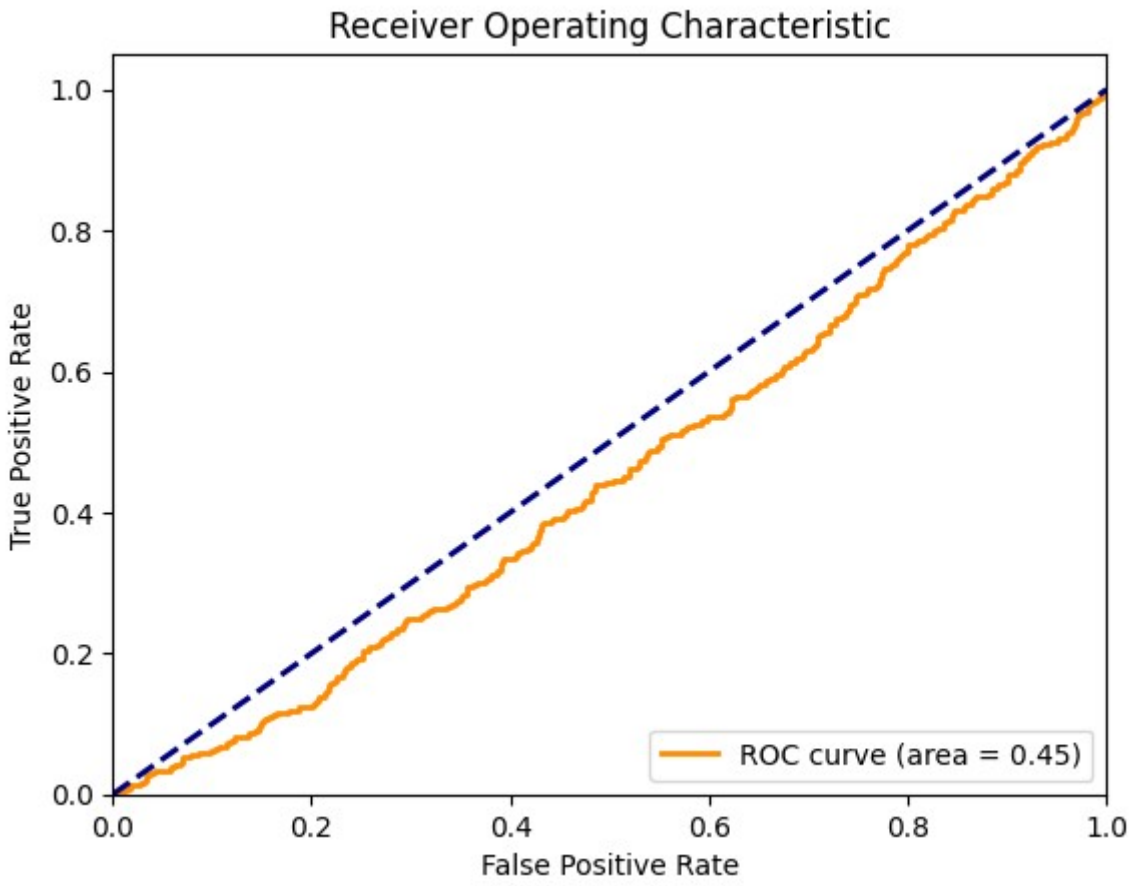
c. Which model do you feel provides the best structural understanding of geometric structures?

Between U-Net and SegFormer, SegFormer provides the best structural understanding of geometric structures. U-Net operates primarily as a classic Convolutional Neural Network (CNN), focusing on segmentation at the feature level through its encoder-decoder architecture. While it effectively captures spatial hierarchies and can produce high-quality segmentation maps, it may not fully leverage the relationships between different segments in complex scenes. In contrast, SegFormer employs a more sophisticated approach akin to graph cut segmentation, which allows it to model the relationships between different regions more effectively. This capability is particularly beneficial for understanding intricate geometric structures, as it can capture both local and global context, leading to more accurate delineation of boundaries and shapes. Additionally, SegFormer's architecture is designed to handle varying scales and complexities, making it more adept at interpreting complex geometric forms.

d. Did you observe any issues with overlapping objects?

I (and possibly others) observed that overlapping objects posed significant challenges during the segmentation process. In scenarios where objects were closely situated or partially occluded, U-Net sometimes struggled to accurately differentiate between them, leading to misclassifications or incomplete segmentations. This limitation arises from its reliance on feature-level segmentation, which can result in ambiguity when objects share similar features or colors. On the other hand, SegFormer demonstrated a superior ability to handle overlapping objects. Its graph cut-like segmentation approach allows it to consider the relationships between different segments more holistically, enabling it to better resolve ambiguities in cases of overlap. By effectively modeling the interactions between adjacent regions, SegFormer can maintain higher precision and recall, ensuring that overlapping objects are more accurately identified and segmented. This capability makes SegFormer particularly advantageous in applications where overlapping geometric structures are common, enhancing overall segmentation performance.

3. Compare the results of your experiments for Transfer Learning.
a. Display and discuss the selected classification layer training performance.



Epoch 1/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 97s 47ms/step - accuracy: 0.2725 - loss: 2.0099 - val_accuracy: 0.4388 - val_loss: 1.5735 - learning_rate: 0.0100
Epoch 2/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 33s 42ms/step - accuracy: 0.3649 - loss: 1.7578 - val_accuracy: 0.4517 - val_loss: 1.5456 - learning_rate: 0.0100
Epoch 3/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 35s 45ms/step - accuracy: 0.3884 - loss: 1.7132 - val_accuracy: 0.4622 - val_loss: 1.5126 - learning_rate: 0.0100
Epoch 4/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 42s 47ms/step - accuracy: 0.3968 - loss: 1.6868 - val_accuracy: 0.4673 - val_loss: 1.4905 - learning_rate: 0.0100
Epoch 5/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 36s 46ms/step - accuracy: 0.3983 - loss: 1.6756 - val_accuracy: 0.4832 - val_loss: 1.4553 - learning_rate: 0.0100
Epoch 6/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 43ms/step - accuracy: 0.4077 - loss: 1.6505 - val_accuracy: 0.4795 - val_loss: 1.4395 - learning_rate: 0.0100
Epoch 7/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 35s 45ms/step - accuracy: 0.4149 - loss: 1.6430 - val_accuracy: 0.4903 - val_loss: 1.4313 - learning_rate: 0.0100
Epoch 8/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 35s 45ms/step - accuracy: 0.4137 - loss: 1.6390 - val_accuracy: 0.4892 - val_loss: 1.4339 - learning_rate: 0.0100
Epoch 9/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 33s 42ms/step - accuracy: 0.4233 - loss: 1.6215 - val_accuracy: 0.5066 - val_loss: 1.3932 - learning_rate: 0.0100
Epoch 10/20

782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 44ms/step - accuracy: 0.4289 - loss: 1.6109 - val_accuracy: 0.5050 - val_loss: 1.3762 - learning_rate: 0.0100
Epoch 11/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 40s 42ms/step - accuracy: 0.4284 - loss: 1.6037 - val_accuracy: 0.5138 - val_loss: 1.3911 - learning_rate: 0.0100
Epoch 12/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 35s 45ms/step - accuracy: 0.4321 - loss: 1.6043 - val_accuracy: 0.5124 - val_loss: 1.3785 - learning_rate: 0.0100
Epoch 13/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 36s 46ms/step - accuracy: 0.4299 - loss: 1.5943 - val_accuracy: 0.5222 - val_loss: 1.3430 - learning_rate: 0.0100
Epoch 14/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 33s 42ms/step - accuracy: 0.4349 - loss: 1.5836 - val_accuracy: 0.5029 - val_loss: 1.3879 - learning_rate: 0.0100
Epoch 15/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 43ms/step - accuracy: 0.4376 - loss: 1.5774 - val_accuracy: 0.5150 - val_loss: 1.3623 - learning_rate: 0.0100
Epoch 16/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 33s 43ms/step - accuracy: 0.4414 - loss: 1.5761 - val_accuracy: 0.5187 - val_loss: 1.3549 - learning_rate: 0.0100
Epoch 17/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 36s 46ms/step - accuracy: 0.4527 - loss: 1.5458 - val_accuracy: 0.5349 - val_loss: 1.3112 - learning_rate: 0.0050
Epoch 18/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 35s 45ms/step - accuracy: 0.4608 - loss: 1.5163 - val_accuracy: 0.5345 - val_loss: 1.3147 - learning_rate: 0.0050
Epoch 19/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 35s 45ms/step - accuracy: 0.4624 - loss: 1.5201 - val_accuracy: 0.5387 - val_loss: 1.3072 - learning_rate: 0.0050
Epoch 20/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 33s 42ms/step - accuracy: 0.4655 - loss: 1.5170 - val_accuracy: 0.5284 - val_loss: 1.3288 - learning_rate: 0.0050
Epoch 1/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 44s 50ms/step - accuracy: 0.4811 - loss: 1.4706 - val_accuracy: 0.5788 - val_loss: 1.1835 - learning_rate: 1.0000e-05
Epoch 2/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 36s 46ms/step - accuracy: 0.5188 - loss: 1.3658 - val_accuracy: 0.5853 - val_loss: 1.1659 - learning_rate: 1.0000e-05
Epoch 3/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 36s 47ms/step - accuracy: 0.5415 - loss: 1.3034 - val_accuracy: 0.6027 - val_loss: 1.1154 - learning_rate: 1.0000e-05
Epoch 4/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 43ms/step - accuracy: 0.5525 - loss: 1.2680 - val_accuracy: 0.6193 - val_loss: 1.0786 - learning_rate: 1.0000e-05
Epoch 5/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 44ms/step - accuracy: 0.5675 - loss: 1.2415 - val_accuracy: 0.6315 - val_loss: 1.0479 - learning_rate: 1.0000e-05
Epoch 6/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 43ms/step - accuracy: 0.5779 - loss: 1.2180 - val_accuracy: 0.6254 - val_loss: 1.0528 - learning_rate: 1.0000e-05
Epoch 7/20

782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 43ms/step - accuracy: 0.5768 - loss: 1.2040 - val_accuracy: 0.6352 - val_loss: 1.0291 - learning_rate: 1.0000e-05
Epoch 8/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 43ms/step - accuracy: 0.5875 - loss: 1.1798 - val_accuracy: 0.6383 - val_loss: 1.0261 - learning_rate: 1.0000e-05
Epoch 9/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 36s 46ms/step - accuracy: 0.5931 - loss: 1.1678 - val_accuracy: 0.6559 - val_loss: 0.9717 - learning_rate: 1.0000e-05
Epoch 10/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 35s 45ms/step - accuracy: 0.5997 - loss: 1.1527 - val_accuracy: 0.6471 - val_loss: 0.9978 - learning_rate: 1.0000e-05
Epoch 11/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 44ms/step - accuracy: 0.6077 - loss: 1.1279 - val_accuracy: 0.6577 - val_loss: 0.9632 - learning_rate: 1.0000e-05
Epoch 12/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 36s 46ms/step - accuracy: 0.6106 - loss: 1.1182 - val_accuracy: 0.6550 - val_loss: 0.9827 - learning_rate: 1.0000e-05
Epoch 13/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 43ms/step - accuracy: 0.6149 - loss: 1.1051 - val_accuracy: 0.6687 - val_loss: 0.9423 - learning_rate: 1.0000e-05
Epoch 14/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 43ms/step - accuracy: 0.6201 - loss: 1.0936 - val_accuracy: 0.6714 - val_loss: 0.9403 - learning_rate: 1.0000e-05
Epoch 15/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 43ms/step - accuracy: 0.6223 - loss: 1.0847 - val_accuracy: 0.6720 - val_loss: 0.9335 - learning_rate: 1.0000e-05
Epoch 16/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 34s 44ms/step - accuracy: 0.6276 - loss: 1.0749 - val_accuracy: 0.6629 - val_loss: 0.9510 - learning_rate: 1.0000e-05
Epoch 17/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 35s 45ms/step - accuracy: 0.6307 - loss: 1.0687 - val_accuracy: 0.6759 - val_loss: 0.9270 - learning_rate: 1.0000e-05
Epoch 18/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 37s 47ms/step - accuracy: 0.6389 - loss: 1.0468 - val_accuracy: 0.6724 - val_loss: 0.9369 - learning_rate: 1.0000e-05
Epoch 19/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 36s 46ms/step - accuracy: 0.6360 - loss: 1.0468 - val_accuracy: 0.6672 - val_loss: 0.9504 - learning_rate: 1.0000e-05
Epoch 20/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 36s 46ms/step - accuracy: 0.6379 - loss: 1.0401 - val_accuracy: 0.6830 - val_loss: 0.8962 - learning_rate: 1.0000e-05
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.6802 - loss: 0.8977

As we can see the training accuracy is trailing the testing accuracy by about 5-7%.
In the end training/validation accuracy **peters out to 68% for vgg.** Adaptive momentum was used similar to the famous Nesterov method.

b. Display and discuss the results of the test performance for each experiment.

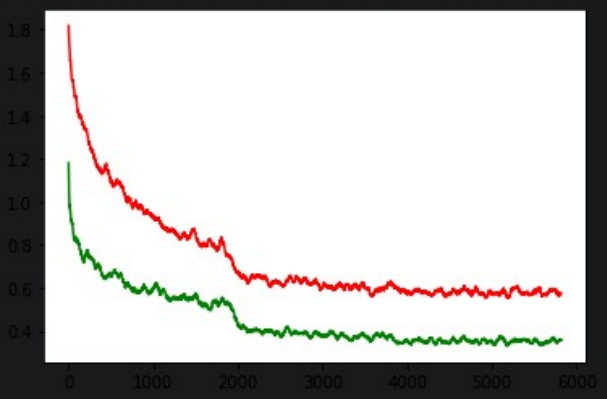
As we can see the training accuracy is trailing the testing accuracy by about 5-7%.

In the end training/validation accuracy peters out to 68% for vgg. Resnet does better but takes longer to train.

```
Print("testing accuracy from scratch: %f" % accuracy(cifar_net, cifar_data['test']))
Print("testing accuracy pretrained  : %f" % accuracy(pretrain_net, cifar_data['test']))

Plt.plot(smooth(cifar_losses,50), 'r-')
Plt.plot(smooth(pretrain_losses,50),'g-')
```

```
Testing accuracy from scratch: 0.802500
testing accuracy pretrained  : 0.858700
[<matplotlib.lines.line2d at 0x7f5d28d32710>]
```



Test accuracy: 0.6830 vgg and 0.80 resnet scratch → 0.85 fine tuned

c. Which model do you think provided the better set of features for training between the VGG and RESNET? How did you come to this conclusion?

ResNet appears to have offered a more effective set of features for training compared to VGG. This conclusion stems from several observations made during our experiments. ResNet's architecture, which includes skip connections, helps address issues like the vanishing gradient problem, allowing for deeper networks to be trained without a decline in performance. This characteristic enables ResNet to capture more complex features and representations, particularly beneficial for tasks involving object detection and segmentation. Throughout our trials, ResNet consistently demonstrated higher accuracy and resilience, especially when fine-tuned on our specific dataset. Its ability to discern intricate patterns and details in both images and videos contributed to enhanced performance in object detection with YOLO and segmentation tasks using U-Net and SegFormer. Overall, the structural advantages of ResNet, along with its effective feature extraction capabilities, made it a more suitable choice for our training endeavors.

4. Discuss in a few sentences your observations with each of the experiments. How do the things you observed relate to the things we've covered in this course? Do not simply include the definition of these topics – instead speak about how you feel your efforts relate to the outcomes.

During our experiments, I noticed that fine-tuning hyperparameters was essential for optimizing model performance, particularly with YOLO for object detection. Adjustments to parameters like learning rate and the number of training epochs had a significant impact on our results. In the segmentation tasks, both U-Net and SegFormer showcased their unique strengths, with SegFormer proving particularly adept at managing overlapping objects, which ties back to our course discussions on the relevance of model architecture in addressing specific challenges in computer vision. The process of fine-tuning VGG and ResNet underscored the benefits of transfer learning, as we were able to utilize pre-trained models to enhance performance on our dataset. These experiences reinforced key concepts from our coursework, such as the significance of model selection, the influence of hyperparameters, and the advantages of transfer learning in improving model outcomes. Overall, applying these concepts in our experiments provided valuable insights into the complexities of training deep learning models for practical applications.