CS478: Introduction to Deep Learning
Fire and Smoke Detection

By: Justinas Janovskis, Matthew Andreas
Instructor: Sreedevi Gutta

Problem Statement

Traditional fire-detecting methods are becoming outdated with today's current technologies, such as computer vision. Since the old way would require multiple people to sit in fire towers across the forest, this would cost a lot of money and require the workers to be in the towers for months at a time. Also, since humans aren't perfect, if a fire started and they weren't paying attention at that time, a fire could spread massively before they saw it. Plus, humans have to sleep, eat, and go to the bathroom, so you don't get 24/7 surveillance. However, if you let computers do the job, they could patrol the entire area 24 hours a day for 365 days a year. This would allow for timely fire reports that would allow for quick response times to put out fires. This also would not require a huge tower to be placed in the middle of a forest, but instead a small camera placed high up to survey the forest. This could also be employed in any areas with fire risk, not just the forest. Allowing computer vision to help solve our problem of fire detection could result in a big decrease in damage and loss of life from the fires we face each year, especially here in California.

Dataset and Preparation Process

The dataset was not originally sourced from the paper that we focused on in this project, but instead was pulled from another paper where the authors were also trying to use various YOLOv5 models to detect smoke and fire. The dataset is located on GitHub at https://github.com/vinchole/zzzccc.

The dataset had a total of 11,667, which the paper split into 73% training, 18% validation, and 9% testing. This gave us a split of 8494 training images, 2,114 validation images, and 1,059 testing images. These images were 640 x 640 and had labels with bounding boxes that went along with each image. Our dataset was also unbalanced as we had twice as many fire instances over smoke instances, which may have caused us some more inaccuracy in our results.

The following table includes of the models that we will be primarily focused on for our implementations, as well as the model name, the # of layers in the model, the # of parameters, the GLOPS, and also a brief description for what the model can be used for. This table will help for future reference when looking at the difference in performance of the models and also when comparing accuracy to efficiency.

Model	# of layers	# of parameters	GFLOPS	Best Suited For
yolov8n	129	3M	8.2	Low-power computations
yolov8s	129	11.1M	28.6	Mobile Devices
yolov8m	169	25.8M	79.1	General Use
yolov8l	209	43.6M	165.4	Larger Datasets/Powerful Hardware
yolov8x	209	68.1M	258.1	Huge amounts of data/ High Accuracy

Implementation

[Original Paper's Method]

- Initially when we were trying to get the paper's model to run, we tried to run it in google colab, but due to the number of epochs along with the size of our dataset, colab did not have enough resources for us, therefore we used the HPC in order to get the results from this method. We tried to follow the paper's implementation 1:1, and we did for most of the project, however in the paper, they had a random selection of the dataset as well which was unmentioned in the paper. For the paper's original method, they used the dataset that was already split into training, validation, and testing sections, which contained the 73%, 18% and 9% split as mentioned above. The parameters that were used were, image size was 640x640 pixels, 128 batch size, 300 epochs of training, SGD optimizer, Initial learning rate 0.01, Momentum 0.937, Weight Decay 0.0005, Learning Rate Warm-up for the first 3 epochs was 0-0.01, and finally there was no data augmentation. So we followed these exact steps to try and recreate their results, however our recreations fell a bit short as mentioned in the results section.

[Justin's Alternative Method]

- In a previous AI course, I worked with Yyolov8 models and wanted to build on that experience by applying transfer learning to improve both the speed and accuracy of yolov8 for our specific use case. Transfer learning is beneficial because it allows the use of pre-trained weights from yolov8 which was trained on the COCO dataset. These weights can be fine-tuned for detecting fire and smoke, enabling the model to converge faster and potentially achieve higher performance. Therefore I would be replacing the head of the model with only 2 categories of smoke and fire, in which I would freeze the model for 10 epochs to work on these new values and then unfreeze the model to start updating its weights based on the new categories.

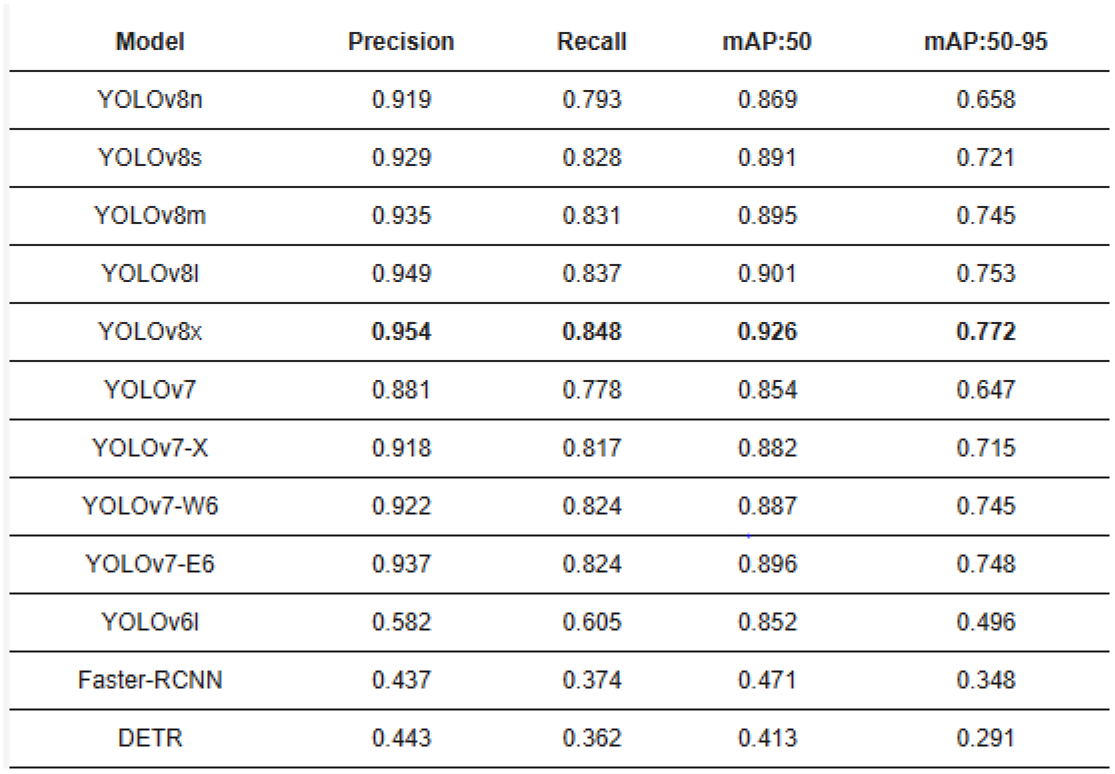
[Matthew's Alternative Method]

- For my method, I wanted to follow in the footsteps of the paper and use the next best yolo model to see if there was any improvement as the model improved. However, I saw that there had actually been four newer versions of YOLO since the paper was published, so I decided to use the newest, which was YOLO12. This model had the 5 versions just like the YOLOv8, which were the n, s, m, l, x models. Another reason for using this model was when I was researching how the model worked, I found articles explaining the different parts that set YOLO12 apart from the previous versions, and one was an attention module. We learned how powerful attention modules could be in different problems in class, so I wanted to see how that could change the scores each model received.
- For YOLO12, I used the exact same parameters that were in the paper's implementation for the nano, small, and medium models, but had to change the batch size for the large and extra-large models. This was because these models were so big that the HPC would run out of memory so the large model was run at a batch size of 64 and the extra-large model was run at a batch size of 32.

Results

[Paper Method]

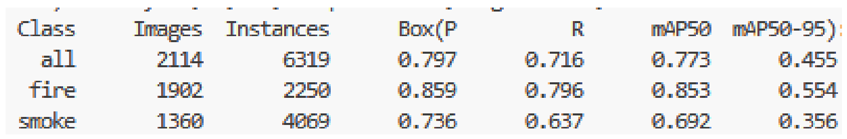
- These are the results that the author's of the paper received after running all of their models using their method.



Model	Precision	Recall	mAP:50	mAP:50-95
YOLOv8n	0.919	0.793	0.869	0.658
YOLOv8s	0.929	0.828	0.891	0.721
YOLOv8m	0.935	0.831	0.895	0.745
YOLOv8l	0.949	0.837	0.901	0.753
YOLOv8x	**0.954**	**0.848**	**0.926**	**0.772**
YOLOv7	0.881	0.778	0.854	0.647
YOLOv7-X	0.918	0.817	0.882	0.715
YOLOv7-W6	0.922	0.824	0.887	0.745
YOLOv7-E6	0.937	0.824	0.896	0.748
YOLOv6l	0.582	0.605	0.852	0.496
Faster-RCNN	0.437	0.374	0.471	0.348
DETR	0.443	0.362	0.413	0.291

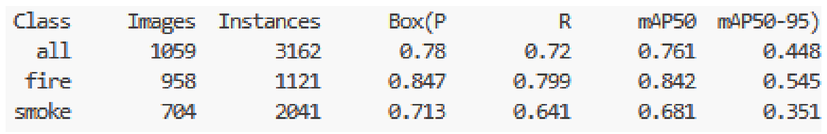
- In the table below, you will see all of the results that we received after we implemented our version of the paper's code and methodology. Below are all 5 of the yolov8 models used including yolov8n, yolov8s, yolov8m, yolov8l, yolov8x, as and both their training and testing results.

yolov8n - Training



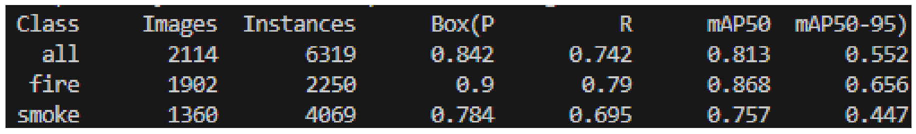
```
   Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
     all       2114       6319      0.797      0.716      0.773      0.455
    fire       1902       2250      0.859      0.796      0.853      0.554
   smoke       1360       4069      0.736      0.637      0.692      0.356
```

yolov8n - Testing



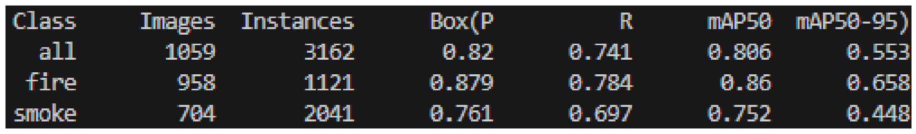
```
   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
     all       1059       3162       0.78       0.72      0.761      0.448
    fire        958       1121      0.847      0.799      0.842      0.545
   smoke        704       2041      0.713      0.641      0.681      0.351
```

yolov8s - Training



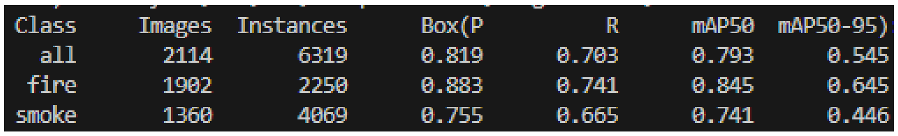
```
   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
     all       2114       6319      0.842      0.742      0.813      0.552
    fire       1902       2250        0.9       0.79      0.868      0.656
   smoke       1360       4069      0.784      0.695      0.757      0.447
```

yolov8s - Testing



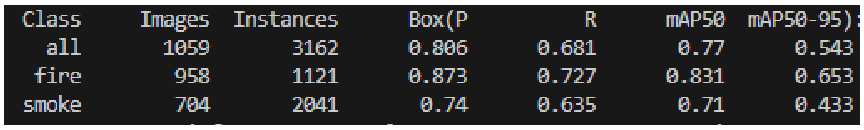
```
   Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
     all       1059       3162       0.82      0.741      0.806      0.553
    fire        958       1121      0.879      0.784       0.86      0.658
   smoke        704       2041      0.761      0.697      0.752      0.448
```

yolov8m - Training



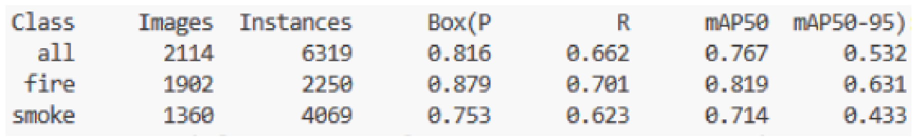
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
  all       2114       6319      0.819      0.703      0.793      0.545
 fire       1902       2250      0.883      0.741      0.845      0.645
smoke       1360       4069      0.755      0.665      0.741      0.446
```

yolov8m - Testing



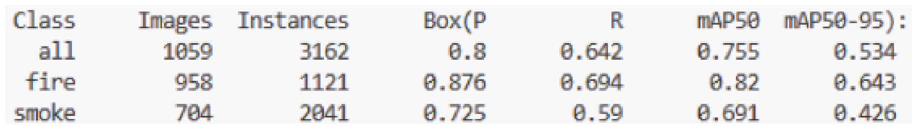
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       1059       3162      0.806      0.681       0.77      0.543
 fire        958       1121      0.873      0.727      0.831      0.653
smoke        704       2041       0.74      0.635       0.71      0.433
```

yolov8l - Training



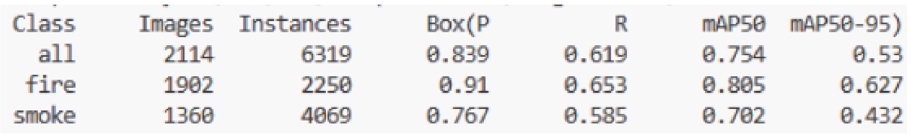
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
  all       2114       6319      0.816      0.662      0.767      0.532
 fire       1902       2250      0.879      0.701      0.819      0.631
smoke       1360       4069      0.753      0.623      0.714      0.433
```

yolov8l - Testing



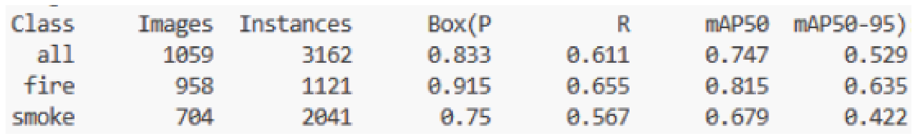
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       1059       3162        0.8      0.642      0.755      0.534
 fire        958       1121      0.876      0.694       0.82      0.643
smoke        704       2041      0.725       0.59      0.691      0.426
```

yolov8x - Training



```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
  all       2114       6319      0.839      0.619      0.754       0.53
 fire       1902       2250       0.91      0.653      0.805      0.627
smoke       1360       4069      0.767      0.585      0.702      0.432
```

yolov8x - Testing



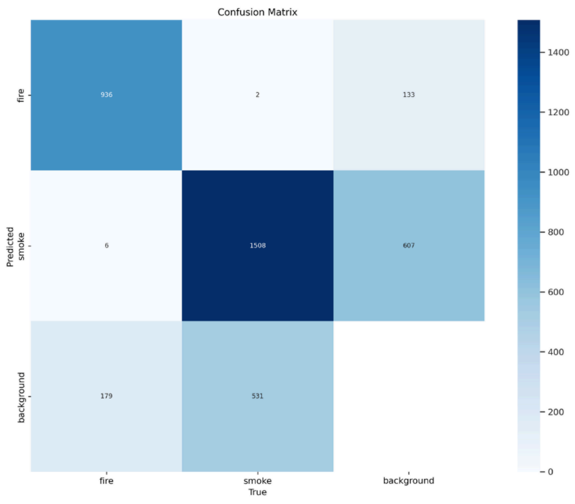
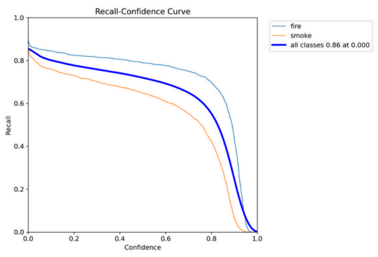
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
  all       1059       3162      0.833      0.611      0.747      0.529
 fire        958       1121      0.915      0.655      0.815      0.635
smoke        704       2041       0.75      0.567      0.679      0.422
```

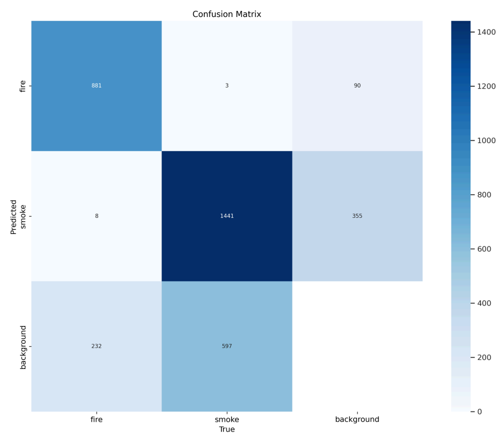
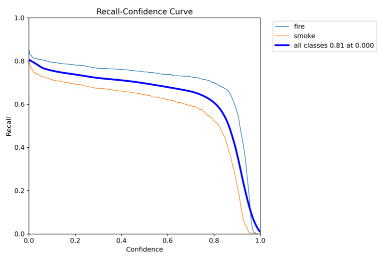
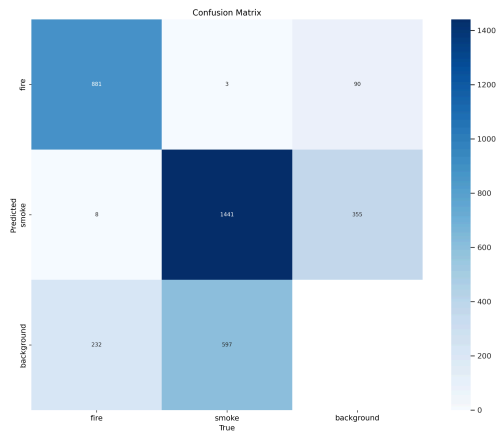
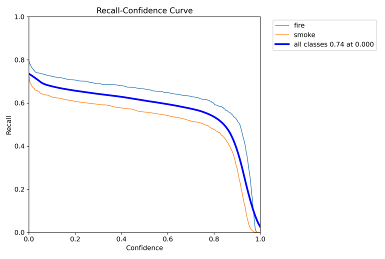
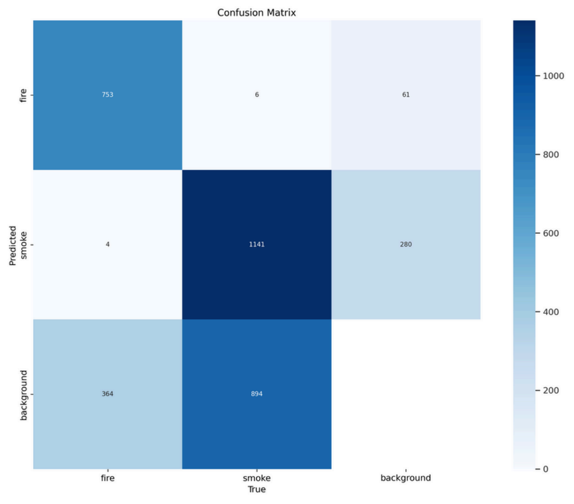
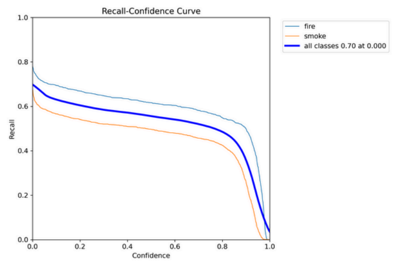
In the following table are the key metrics that we observed after running all of the paper's methods on our own, using the HPC. These are all of the results from the testing set and the key result we wanted to observe is the recall, due to false negatives having a major impact in fire detection. If a paper were to break and no report was made, then this could cause a massive wildfire which could harm and be fatal to so many people and animals, which is why we want to have the best recall possible. The precision is added to show how accurate our model was also in regards to false positives, just as a comparison for future reference.

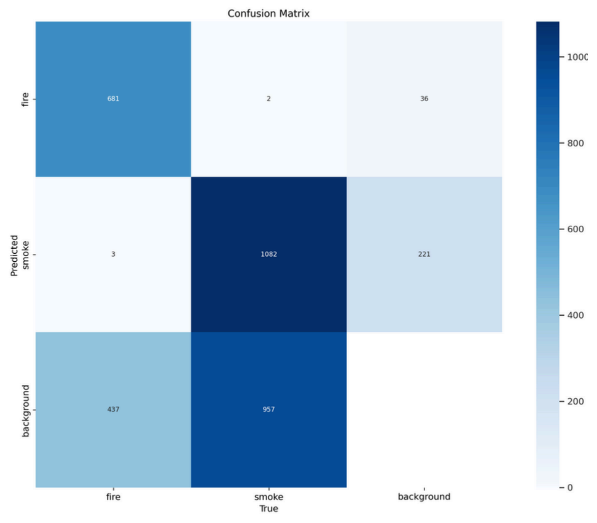
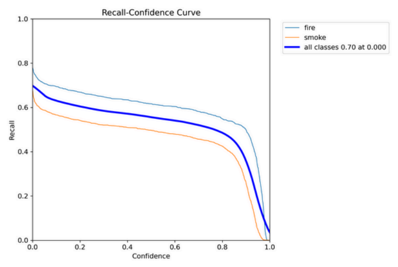
Model	P	R	Paper's P	Paper's R
Yolov8n	.78	.72	.919	.793
Yolov8s	.82	**.741**	.929	.828
Yolov8m	.806	.681	.935	.831

Yolov8l	.8	.642	.949	.837
Yolov8x	**.833**	.611	**.954**	**.848**

Comparing our results to the paper's original results, we can see that most of our results were behind in terms of precision and recall by a significant amount. This immense difference could be from the paper mentioning something about randomizing the training/testing sets, however they never provided the exact code or how the data was randomly split, therefore trying to recreate the exact results would be extremely difficult. From our results we can see that we actually had a difference from the paper's results, in the sense that our yolov8s had a higher recall than the rest of the models, where the paper had a better recall on the yolov8x model. However, for precision, we received the same result of the yolov8x model was the best for both, our implementation of the paper's methods, as well as the author's implementation of yolov8x. Therefore in our findings, we can see that if recall is an important factor, than yolov8s could produce results that are more balanced especially since it is a much smaller model than the yolov8x or yolov8l which have many more parameters.

Model	Confusion Matrix	Recall - Confidence Curve
yolov8n		

yolov8s		
yolov8m		
yolov8l		

yolov8x		

In the table above, most of the testing cross matrices have the same pattern where most of the fires were predicted as fires, smoke was mostly predicted as smoke, however background was never predicted as background. This was very interesting and it most likely means that our model was having a very hard time differentiating between smoke and other weather environments such as pollution or fog, which caused the model to believe that smoke was actually occurring. Another key result to look at for our confusion matrices, is that a decent amount of fires were being predicted as background, which is very concerning especially since we want these fires to be detected, however maybe the models were underfitting to the data, and were not fully recognizing the patterns of fires in the images. Or another key factor that could lead to this discrepancy is the unbalanced dataset where fire had more instances and therefore could've had an impact on the models. For the recall-confidence curves, it was interesting to see that the smaller models had a higher recall at a lower confidence than the bigger models, however all of the curves converged to be almost exactly the same as other models.

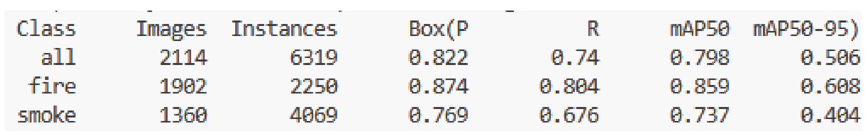
While attempting to replicate the paper's results, we found that yolov8s yielded the most balanced performance in our implementations of the original paper, especially in terms of recall. This model may be more suited to detecting fine-grained features in a real-world, noisy dataset like ours. The yolov8x model achieved the highest precision, but its relatively low recall undermines its suitability for our goal of minimizing undetected fires out in the real world surveillance.

[Justin's Transfer Learning Method]

- In the table below, you will see all of the results from implementing our version of the yolov8's models with transfer learning by replacing the head of the model.

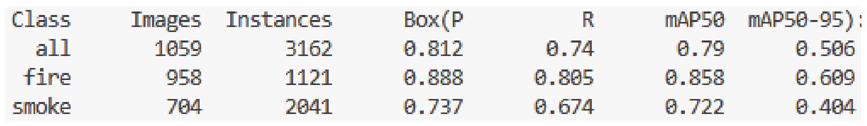
Below are all 5 of the yolov8 models used including yolov8n, yolov8s, yolov8m, yolov8l, yolov8x, as and both their training and testing results.

yolov8n - Training



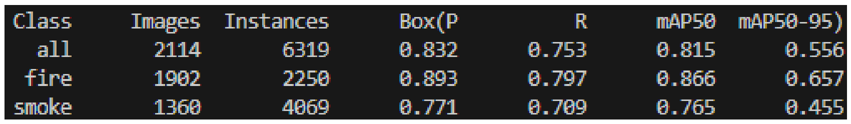
```
Class     Images  Instances      Box(P          R       mAP50  mAP50-95)
  all       2114       6319      0.822       0.74       0.798      0.506
 fire       1902       2250      0.874      0.804       0.859      0.608
smoke       1360       4069      0.769      0.676       0.737      0.404
```

yolov8n - Testing



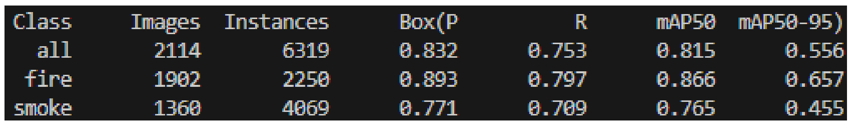
```
Class     Images  Instances      Box(P          R       mAP50  mAP50-95):
  all       1059       3162      0.812       0.74        0.79      0.506
 fire        958       1121      0.888      0.805       0.858      0.609
smoke        704       2041      0.737      0.674       0.722      0.404
```

yolov8s - Training



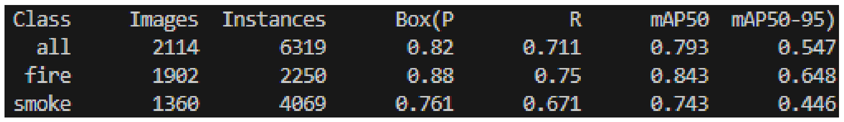
```
Class     Images  Instances      Box(P          R       mAP50  mAP50-95)
  all       2114       6319      0.832      0.753       0.815      0.556
 fire       1902       2250      0.893      0.797       0.866      0.657
smoke       1360       4069      0.771      0.709       0.765      0.455
```

yolov8s - Testing



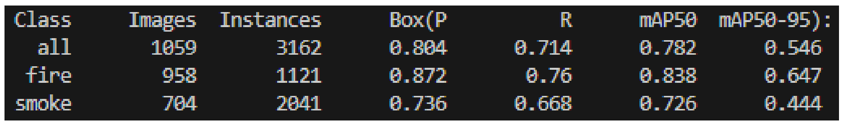
```
Class     Images  Instances      Box(P          R       mAP50  mAP50-95)
  all       2114       6319      0.832      0.753       0.815      0.556
 fire       1902       2250      0.893      0.797       0.866      0.657
smoke       1360       4069      0.771      0.709       0.765      0.455
```

yolov8m - Training



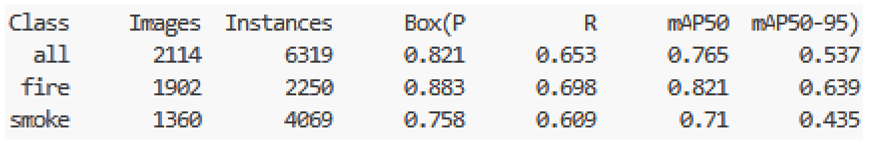
```
Class     Images  Instances      Box(P          R       mAP50  mAP50-95)
  all       2114       6319       0.82      0.711       0.793      0.547
 fire       1902       2250       0.88       0.75       0.843      0.648
smoke       1360       4069      0.761      0.671       0.743      0.446
```

yolov8m - Testing



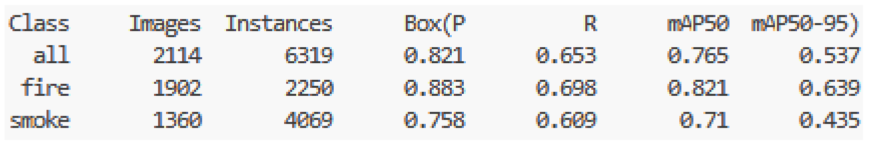
```
Class     Images  Instances      Box(P          R       mAP50  mAP50-95):
  all       1059       3162      0.804      0.714       0.782      0.546
 fire        958       1121      0.872       0.76       0.838      0.647
smoke        704       2041      0.736      0.668       0.726      0.444
```

yolov8l - Training



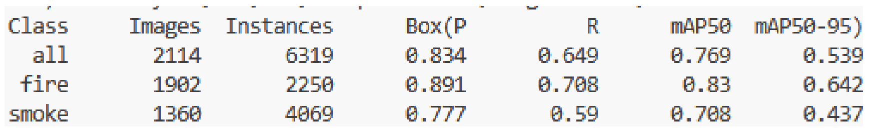
```
Class     Images  Instances      Box(P          R       mAP50  mAP50-95)
  all       2114       6319      0.821      0.653       0.765      0.537
 fire       1902       2250      0.883      0.698       0.821      0.639
smoke       1360       4069      0.758      0.609        0.71      0.435
```

yolov8l - Testing



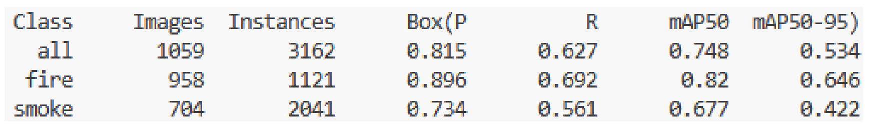
```
Class     Images  Instances      Box(P          R       mAP50  mAP50-95)
  all       2114       6319      0.821      0.653       0.765      0.537
 fire       1902       2250      0.883      0.698       0.821      0.639
smoke       1360       4069      0.758      0.609        0.71      0.435
```

yolov8x - Training



```
Class     Images  Instances      Box(P          R       mAP50  mAP50-95)
  all       2114       6319      0.834      0.649       0.769      0.539
 fire       1902       2250      0.891      0.708        0.83      0.642
smoke       1360       4069      0.777       0.59       0.708      0.437
```

yolov8x - Testing

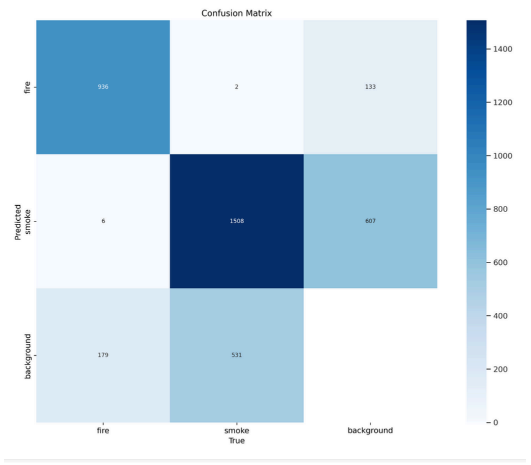
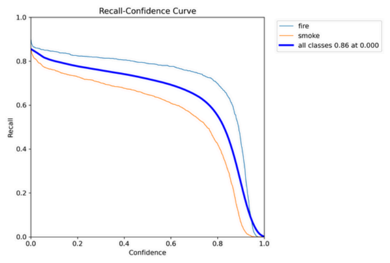


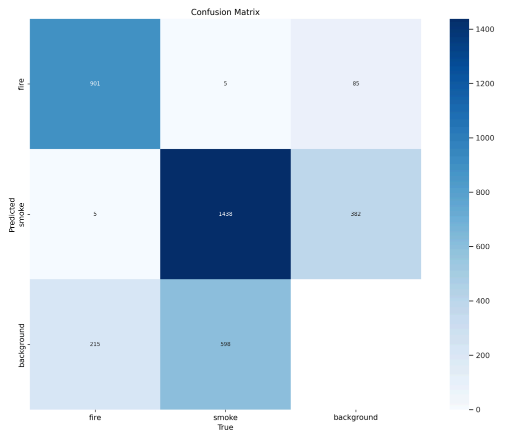
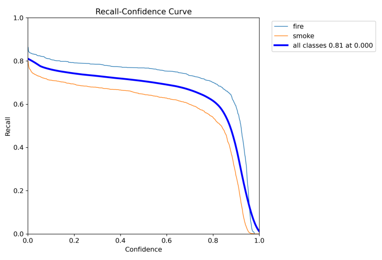
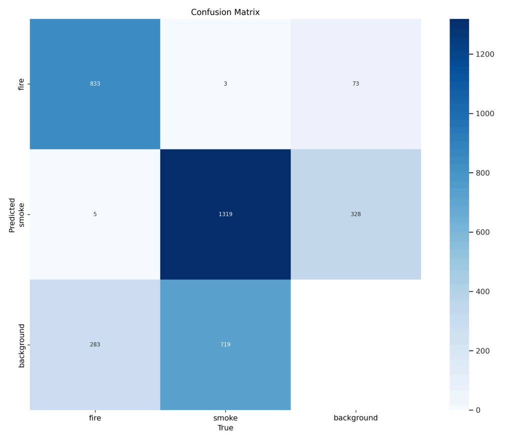
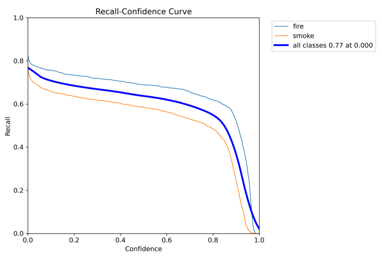
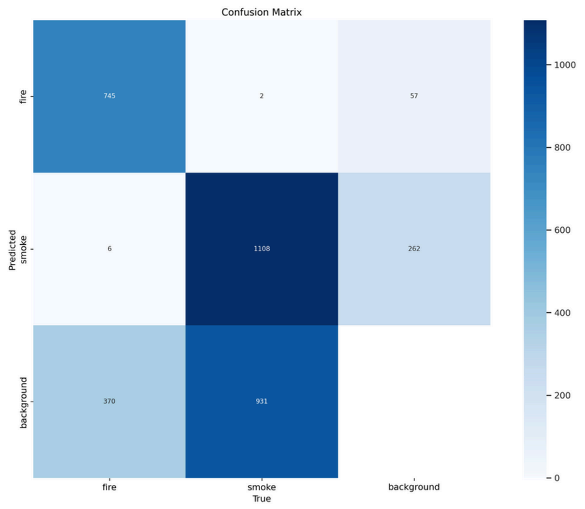
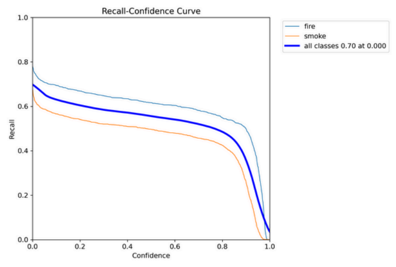
```
Class     Images  Instances      Box(P          R       mAP50  mAP50-95)
  all       1059       3162      0.815      0.627       0.748      0.534
 fire        958       1121      0.896      0.692        0.82      0.646
smoke        704       2041      0.734      0.561       0.677      0.422
```

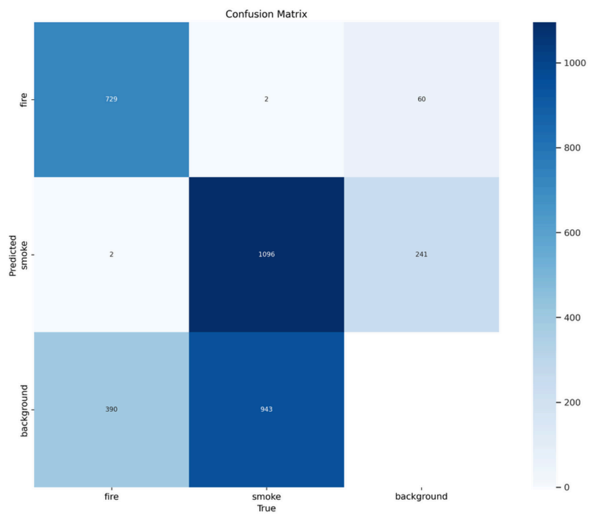
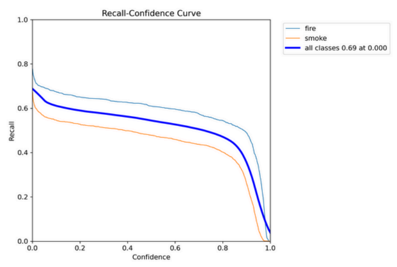
Following all of that data, below is a summarized table of all of the testing set values that we received for the 5 models, specifically the Precision and Recall values which were of our primary concern when working on this project.

Model	P (transfer)	R (transfer)	P (No transfer)	R (No transfer)
Yolov8n	.812	.74	.78	.72
Yolov8s	**.832**	**.753**	.82	**.741**
Yolov8m	.804	.714	.806	.681
Yolov8l	.821	.653	.8	.642
Yolov8x	.815	.627	**.833**	.611

From the table above, we can see that most of the metrics that we received from using transfer learning were actually higher than without transfer learning. The biggest upgrade was our recall for the yolov8m model which ended up increasing by over 3.3%. Therefore from this data comparison we are able to see that there is indeed a beneficial factor to using transfer learning as the model was able to reduce the amount of false negatives and false positives simultaneously and in return we received better results. It is a bit surprising however that our yolov8s model was able to outperform the rest of the models, but this is probably due to the yolov8s recognizing patterns in these small scale fire detections and having better predictions in regards to those patterns.

Model	Confusion Matrix	Recall - Confidence Curve
yolov8n		

yolov8s		
yolov8m		
yolov8l		

yolov8x		

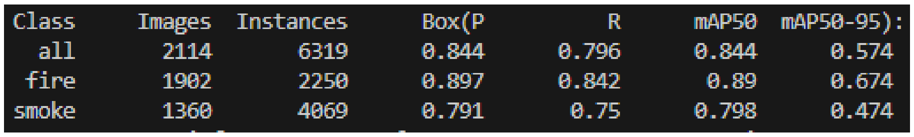
In the table above, our data for transfer learning actually looks very similar to the original paper's method that we implemented, and the main key change is that more of the smoke images were actually being detected as smoke. But interestingly enough, it was surprising to see that background was never predicted by the model and again this is most likely to be fog/background weather or rain which could have made the images seem like they were smoke but in reality they had nothing in them, so our model had a hard time differentiating these. And similarly our recall-confidence curves also converged at similar spots, which indicates that these models are strong at detecting fires even when the confidence is not high.

[Matthew's Alternative Method's results]

For my results, I found them very interesting since many might think that the larger models would perform better in everything. However, the models that seemed to do the best on all of the metrics were the medium, small, and nano models. These models have a fraction of the parameters but seemed to do better on every metric. Nano took first in its high precision, with a score of 0.85, and mAP50 with 0.839. Small took first in its recall with 0.79. Medium took first in its mAP50-95 with 0.598.

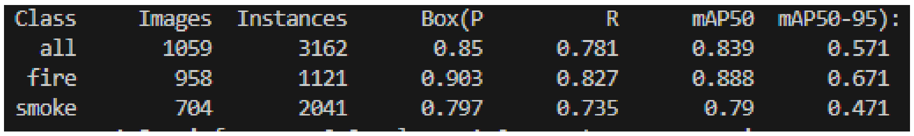
In the table below, you will see all of the results that we received after we implemented the YOLO12 models in the same way as the paper's methodology. Below are all 5 of the YOLO12 models used, including YOLO12n, YOLO12s, YOLO12m, YOLO12l, YOLO12x, and both their training and testing results.

yolo12n - Training



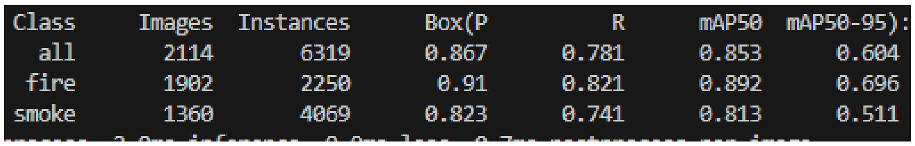
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       2114       6319      0.844      0.796      0.844      0.574
 fire       1902       2250      0.897      0.842       0.89      0.674
smoke       1360       4069      0.791       0.75      0.798      0.474
```

yolo12n - Testing



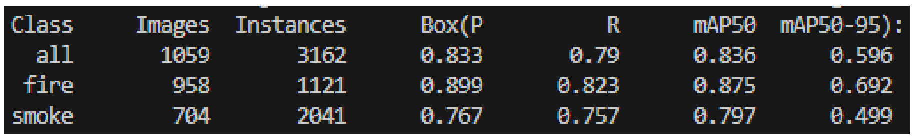
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       1059       3162       0.85      0.781      0.839      0.571
 fire        958       1121      0.903      0.827      0.888      0.671
smoke        704       2041      0.797      0.735       0.79      0.471
```

yolo12s - Training



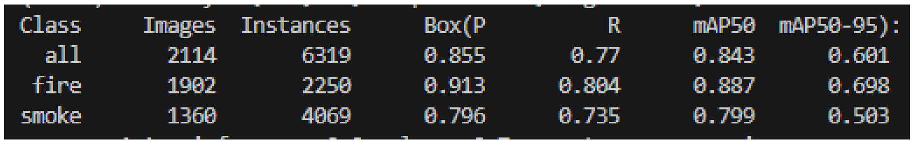
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       2114       6319      0.867      0.781      0.853      0.604
 fire       1902       2250       0.91      0.821      0.892      0.696
smoke       1360       4069      0.823      0.741      0.813      0.511
```

yolo12s - Testing



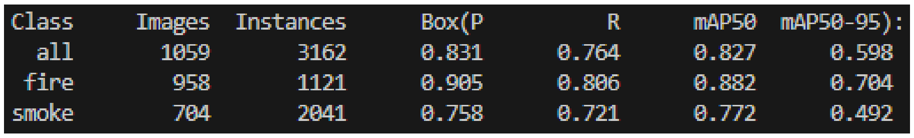
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       1059       3162      0.833       0.79      0.836      0.596
 fire        958       1121      0.899      0.823      0.875      0.692
smoke        704       2041      0.767      0.757      0.797      0.499
```

yolo12m - Training



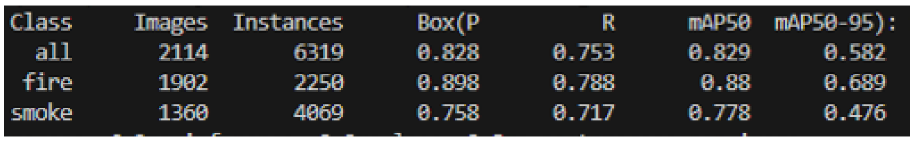
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       2114       6319      0.855       0.77      0.843      0.601
 fire       1902       2250      0.913      0.804      0.887      0.698
smoke       1360       4069      0.796      0.735      0.799      0.503
```

yolo12m - Testing



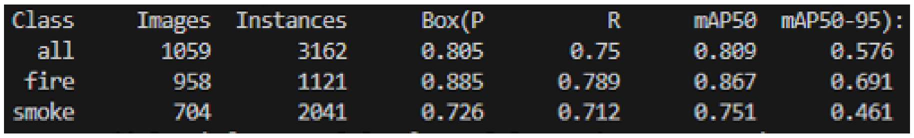
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       1059       3162      0.831      0.764      0.827      0.598
 fire        958       1121      0.905      0.806      0.882      0.704
smoke        704       2041      0.758      0.721      0.772      0.492
```

yolo12l - Training



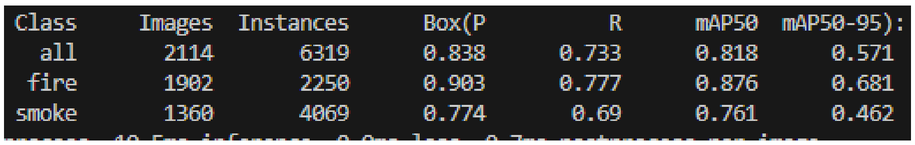
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       2114       6319      0.828      0.753      0.829      0.582
 fire       1902       2250      0.898      0.788       0.88      0.689
smoke       1360       4069      0.758      0.717      0.778      0.476
```

yolo12l - Testing



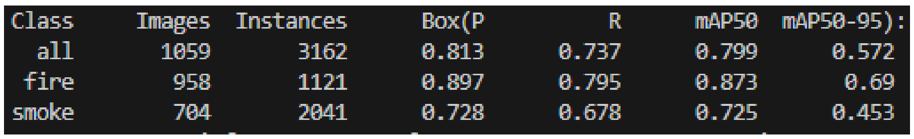
```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       1059       3162      0.805       0.75      0.809      0.576
 fire        958       1121      0.885      0.789      0.867      0.691
smoke        704       2041      0.726      0.712      0.751      0.461
```

yolo12x - Training



```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       2114       6319      0.838      0.733      0.818      0.571
 fire       1902       2250      0.903      0.777      0.876      0.681
smoke       1360       4069      0.774       0.69      0.761      0.462
```

yolo12x - Testing



```
Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
  all       1059       3162      0.813      0.737      0.799      0.572
 fire        958       1121      0.897      0.795      0.873       0.69
smoke        704       2041      0.728      0.678      0.725      0.453
```

In the following table are the key metrics that we observed after running all of the YOLO12 models, using the HPC. These are all of the results from the testing set, and the key result we wanted to observe is the recall, due to false negatives having a major impact in fire detection. If a model were to miss a fire and no report was made, then this could cause a massive wildfire which could harm and be fatal to so many people and animals, which is why we want to have the best recall possible. The precision is added to show how accurate our model was also in regards to false positives, just as a comparison for future reference.

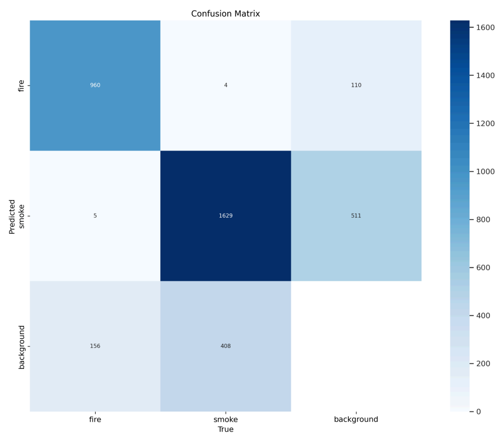
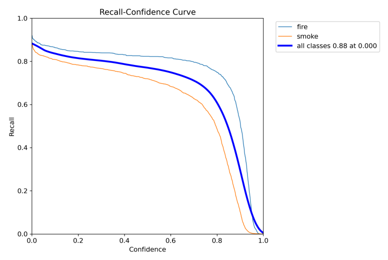
Model	Our original Implementation P	Our original Implementation R	Model	Yolo12's P	Yolo12's R
Yolov8n	.78	.72	Yolo12n	**.85**	.781
Yolov8s	.82	**.741**	Yolo12s	.833	**.79**
Yolov8m	.806	.681	Yolo12m	.831	.764
Yolov8l	.8	.642	Yolo12l	.805	.75
Yolov8x	**.833**	.611	Yolo12x	.813	.737

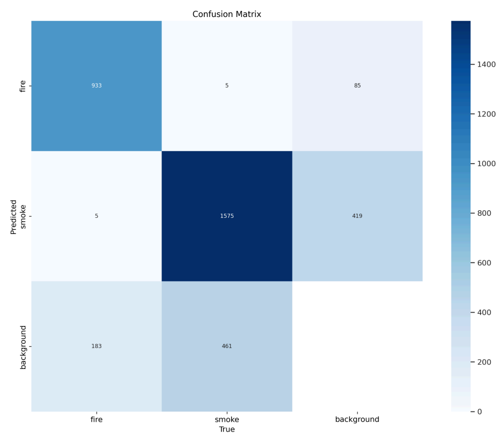
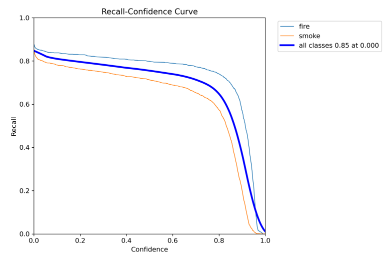
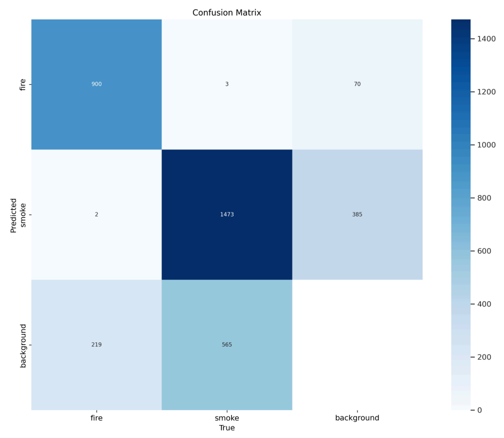
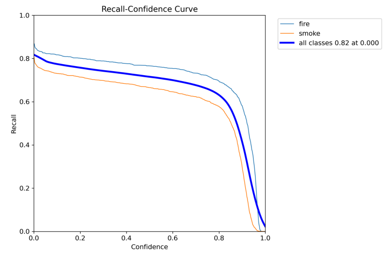
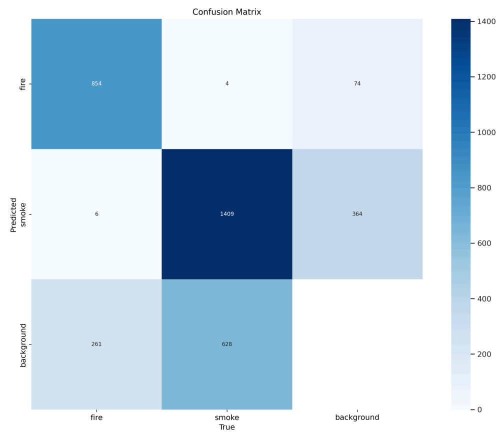
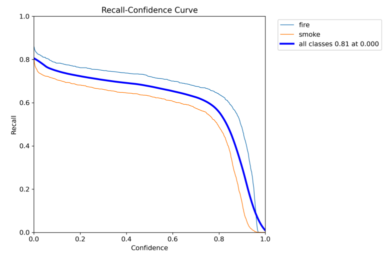
We decided to compare the results to our implementation of the paper's model since it would be a more accurate comparison, as we were not able to get as high of results as the paper. Comparing the results of the YOLO12 models to our implementation of the paper's model results, we can see that YOLO12 has made a big improvement in recall, with many of the scores increasing 0.05 or more. This was really cool to see, especially since we were working to increase the recall score to get fewer false negatives, because false negatives in a fire and smoke detection system could lead to unnoticed fires becoming massive without being noticed and causing a lot of damage and deaths. We can also see that almost all models from YOLO12 increased in it's precision except for YOLO12x, which actually dropped by .02, which I found extremely odd. I thought maybe it overfitted, but if you look at the training metrics, you can see that still the bigger models did not do too good compared to the small models.

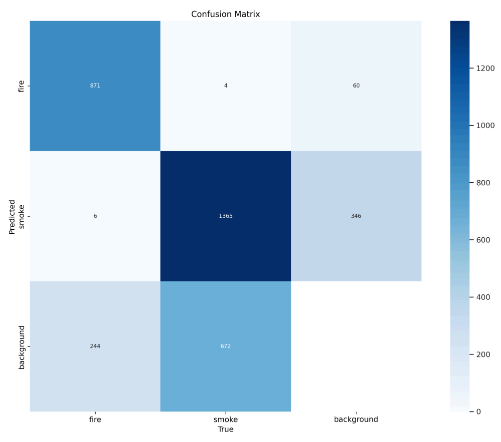
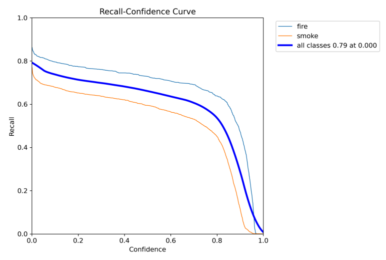
Model	Our original	Our original	Model	Yolo12's	Yolo12's

	Implementation mAP50	Implementation mAP50-95		mAP50	mAP50-95
Yolov8n	.761	.448	Yolo12n	**.839**	.571
Yolov8s	**.806**	**.553**	Yolo12s	.836	.596
Yolov8m	.77	.543	Yolo12m	.827	**.598**
Yolov8l	.755	.534	Yolo12l	.809	.576
Yolov8x	.747	.529	Yolo12x	.799	.572

I decided I also wanted to look at and compare the mAP50 and mAP50-95 scores to see if the larger models did better in this area. However, as we can see in both the original paper implementation and the YOLO12 implementation, the larger models seem to have some of the lower scores. The cool thing to see is that even though they are low, they improved which was really awesome to see. Compared to the real paper results, we got the opposite since they got that the yolov8x was best in all categories and we see that it has the worst score in the mAP50 and the second worst score in the mAP50-95. We were really puzzled about why we could not get the same results as in the paper, and I really wonder how high the scores would be with the YOLO12 models if we could figure out the right implementation.

Model	Confusion Matrix	Recall - Confidence Curve
YOLO12n		

YOLO12s		
YOLO12m		
YOLO12l		

YOLO12x		

In the table above, we can see all of the confusion matrices and recall confidence curves for all of the YOLO12 models. One of the trends that we saw in this and other methods was that there are no correctly classified backgrounds which we found very strange and could never figure out a way to fix. However, it still was predicting background, as we could see in the true fire and smoke columns. Another trend we saw on the YOLO12 models was that as the model became larger, it had a harder and harder time predicting smoke as smoke and not background. We theorized that the problem might have been that, since there were so many parameters in the bigger models, they had trouble distinguishing between weather conditions and the smoke. Looking at the recall confidence graphs, we can also see that as we get bigger and bigger models it again gets lower and lower values.

Model	P (transfer)	R (transfer)	Model	Yolo12's P	Yolo12's R
Yolov8n	.812	.74	Yolo12n	**.85**	.781
Yolov8s	**.832**	**.753**	Yolo12s	.833	**.79**
Yolov8m	.804	.714	Yolo12m	.831	.764
Yolov8l	.821	.653	Yolo12l	.805	.75
Yolov8x	.815	.627	Yolo12x	.813	.737

We also wanted to compare the differences in recall and precision with the YOLOv8 transfer learning method and the YOLO12 method, which we can see in the table above. For all of the values in the recall column, we can see that YOLO12 had better values across the board and on the larger models had a jump of .1 or more which was really interesting to see. However, in the precision column, we can see that for the top 3 models, the YOLO12 models beat the YOLOv8 models, but the larger models actually did better in the YOLOv8 version than the YOLO12. This was weird because you would think that if a model had a better recall than another, it would also have better precision, but in this case, that is not true. This kind of shows the beauty of choosing the right model for the task, since different models can be good at different things.

After looking at all the comparisons, we can see that the YOLO12 models have the highest recall and precision scores out of the 3 implementations. However, these scores are still a little low compared to the original paper's scores. We really wished we could have figured out the exact way the paper's scores were obtained so we could have a real comparison with the original paper. However, I do believe the results we obtained were still substantial at seeing that if YOLO12 was implemented like the method in the paper, it would have an even better precision and recall, possibly high enough to put into real-world use.

Challenges

- **Label Inconsistencies:** Some of the images were misclassified in the dataset. This impacted both CNN training and evaluation accuracy.
- **False positives/negatives:** Specifically for the CNN, the CNN occasionally predicted fire when there was no fire in the image and vice versa, especially when conditions were not ideal in the image.
- **YOLO's Misses:** YOLO sometimes failed to detect small fire or smoke regions. Which was the motive of adding the CNN fallback mechanism.
- **Colab**: Using google colab had its limitations such as GPU availability, session timeouts, and restricted storage. Which occasionally interrupted progress after spending hours doing something.
- **Time:** Even with the 2 HPC accounts we got, it was really difficult to run all the models up to the paper specifications. The HPC definitely made this project finishable, but still barely let us finish on time.
- **Exact Paper Implementation:** We were really close to getting these models to run exactly as they did in the paper, but as we saw by our results, we were missing something. This caused us not to have a really valuable comparison with the results in the paper, since our results were substantially lower than theirs, but we still compared our implementations' results with our new methods.

Future Work

[Justin]

- If this project were to be continued in the future, it would be worth it to try and randomize our data as a lot of the images came from videos and were just frames of each video, but it also seemed like the data wasn't properly randomized in the training/testing sets. Other future improvements would be to fine-tune our transfer method more, and possibly changing parameters to see if any further improvements could be made, but time constraints prevented us from trying out new strategies. It would also be interesting to try it in a real world surveillance camera to see if it can detect fires, because the images will be scanned pretty quickly, so as long as some of the frames can be captured, it can still be a useful application.

[Matthew]

- For future work, I would love to try to tune the hyperparameters for the YOLO12 models to try to make the recall for the models be even better and further reduce the number of false negatives. We could also try applying the new transfer learning we applied to the YOLOv8 and see if that gives us an even better recall. After all of that, it would be really cool to work with actual camera footage of the start of a forest fire and have the model run and see how accurate and how fast the model is at finding the fire.

Code

- Code for Original Paper's Implementation, Transfer Learning, and also yolov12 Models

Contributions

[Justin]

- In this project, I worked on finding the original paper as well as coming up with our original method which we scrapped for our newer methods. I worked on implementing and writing the code for both the original paper's methods as well as figuring out how to apply transfer learning to the yolov8 models. I also figured out how to get the HPC running our code and even running it on 4 GPU's so that our results would arrive much faster, and therefore allowing us to get results to present and report. Finally, I worked on the presentation, as well as the final report, primarily focused on all of the paper's original and my transfer learning methods and results.

[Matthew]

- On our project, I helped look for and find the paper that we decided to do our project on. I also helped Justin run a few of the models from the original paper implementation, as well as some of the new transfer learning models. I ran the small and medium models in both the original and transfer learning implementations. After completing this, I wanted to move to my own method, which was running the same data in the same way as the paper in a newer Yolo model. So I ran all 5 models for YOLO12 and compared all of the results to the original paper's results to see which was better. I also worked on the presentation and final report, focusing on my YOLO12 implementation

References

1. Fire and Smoke Detection Using Fine-Tuned YOLOv8 and YOLOv7 Deep Models
2. https://docs.ultralytics.com/models/yolo12/
3. https://arxiv.org/html/2407.12040v7