



TECNOLÓGICO NACIONAL DE MÉXICO



Tecnológico Nacional De México
Instituto Tecnológico De Ciudad
Madero.

Ingeniería en Sistemas
Computacionales

Materia: Inteligencia Artificial.

Dr. Juan Frausto Solís.

EQUIPO #3

Integrantes:

Aguilera Flores Alexander Jair
20070570

Arteaga Morales Johan Sebastián.
20070498

Medellín Trejo José Luis
20070519

Obando Valladares Rubén Eduardo.
20070563

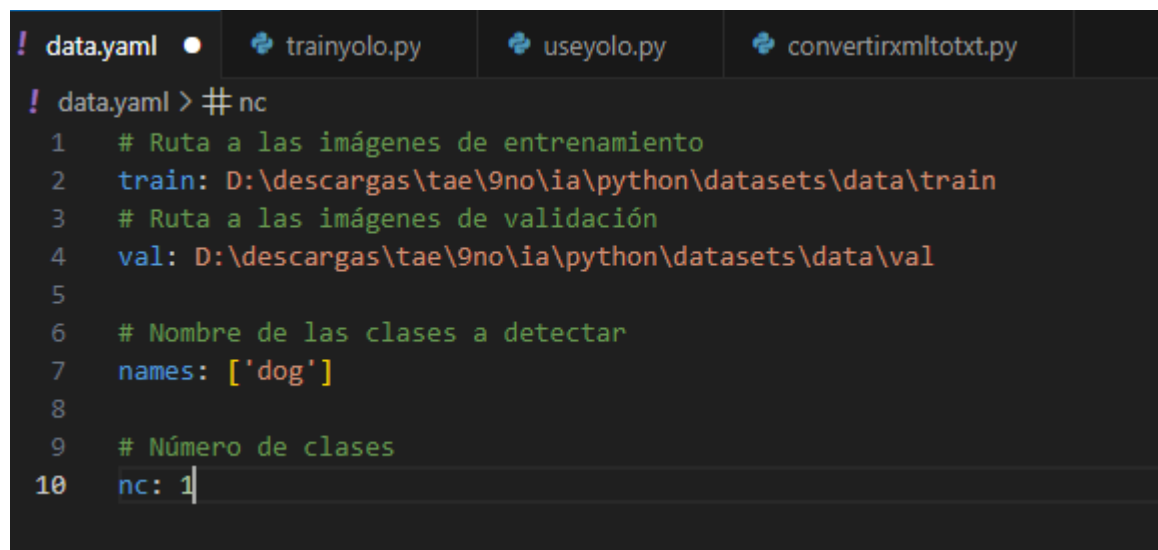
Este proyecto utiliza un modelo de inteligencia artificial para la detección de objetos mediante el entrenamiento de YOLO. Su finalidad es mejorar la precisión en tareas de identificación visual automatizada en aplicaciones industriales.

Base de datos utilizada: En nuestro caso el dataset lo obtuvimos en el siguiente enlace: [Stanford Dogs dataset for Fine-Grained Visual Categorization](#) el cual nos sirvió para el proceso de etiquetado.

Parámetros de configuración del modelo y entrenamiento:

- En este punto se especifican parámetros específicos utilizados en el entrenamiento del modelo. Esto incluye:

Creación del archivo .yaml

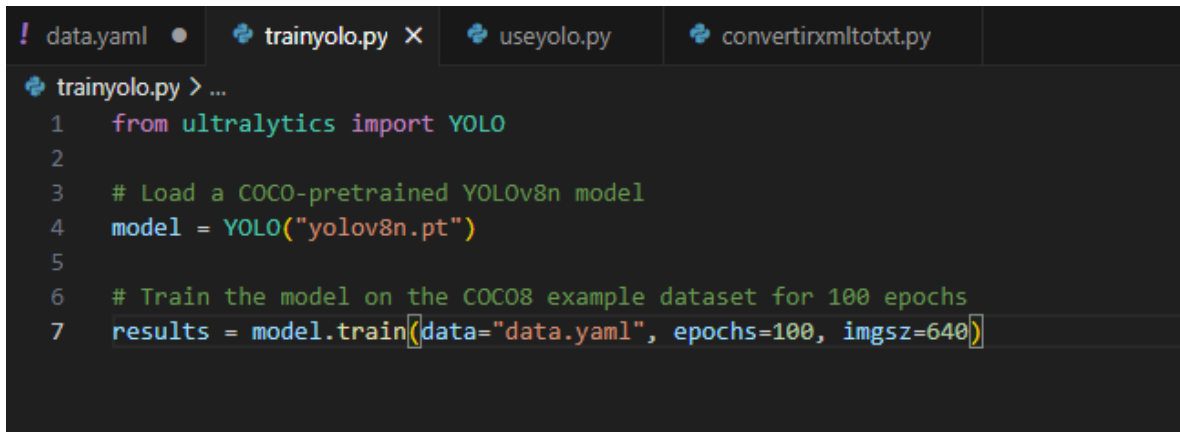


```
! data.yaml ● trainyolo.py useyolo.py convertirxmltotxt.py
! data.yaml > ## nc
1  # Ruta a las imágenes de entrenamiento
2  train: D:\descargas\tae\9no\ia\python\datasets\data\train
3  # Ruta a las imágenes de validación
4  val: D:\descargas\tae\9no\ia\python\datasets\data\val
5
6  # Nombre de las clases a detectar
7  names: ['dog']
8
9  # Número de clases
10 nc: 1
```

Se especifico la ruta de la carpeta train como de la carpeta val, el nombre de la clase que se va detectar y el número de clases.

Creación del entrenamiento

Trainyolo.py

A screenshot of a code editor with a dark theme. At the top, there are four tabs: 'data.yaml' with an error icon, 'trainyolo.py' with a close icon, 'useyolo.py', and 'convertirxmltotxt.py'. The 'trainyolo.py' tab is active, showing a Python script. The script starts with a comment 'trainyolo.py > ...' followed by seven lines of code. Line 1 imports YOLO from ultralytics. Line 2 is empty. Line 3 is a comment about loading a COCO-pretrained YOLOv8n model. Line 4 creates a model object. Line 5 is empty. Line 6 is a comment about training on the COCO8 dataset for 100 epochs. Line 7 calls the train method on the model with parameters data='data.yaml', epochs=100, and imgsz=640.

```
! data.yaml ● trainyolo.py × useyolo.py convertirxmltotxt.py
trainyolo.py > ...
1  from ultralytics import YOLO
2
3  # Load a COCO-pretrained YOLOv8n model
4  model = YOLO("yolov8n.pt")
5
6  # Train the model on the COCO8 example dataset for 100 epochs
7  results = model.train(data="data.yaml", epochs=100, imgsz=640)
```

Se uso la misma instrucción data en el github con un solo cambio que fue el nombre del archivo .yaml.

Comienzo del entrenamiento

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

22 [15, 18, 21] 1 751507 ultralytics.nn.modules.head.Detect [1, [64, 128, 256]]
Model summary: 225 layers, 3,011,043 parameters, 3,011,027 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs\detect\train12', view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
train: Scanning D:\descargas\tae\9no\ia\python\datasets\data\train\labels.cache... 146 images, 0 backgrounds, 0 corrupt: 100% [██████████] 146/146 [00:00<?, ?it/s]
val: Scanning D:\descargas\tae\9no\ia\python\datasets\data\val\labels.cache... 26 images, 0 backgrounds, 0 corrupt: 100% [██████████] 26/26 [00:00<?, ?it/s]
Plotting labels to runs\detect\train12\labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: Adam(lr=0.002, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
TensorBoard: model graph visualization added ✓
Image sizes 640 train, 640 val
Using 0 dataloader workers
Logging results to runs\detect\train12
Starting training for 100 epochs...

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
1/100 0G 0.9213 2.3 1.405 6 640: 100% [██████████] 10/10 [00:41<00:00, 4.12s/it]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 1/1 [00:03<00:00, 3.19s/it]
all 26 29 0.00372 1 0.936 0.715

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
2/100 0G 0.8909 1.43 1.401 4 640: 100% [██████████] 10/10 [00:44<00:00, 4.41s/it]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 1/1 [00:03<00:00, 3.03s/it]
all 26 29 0.84 0.182 0.841 0.545

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
3/100 0G 0.9011 1.295 1.362 6 640: 100% [██████████] 10/10 [00:44<00:00, 4.47s/it]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 1/1 [00:03<00:00, 3.09s/it]
all 26 29 0.746 0.708 0.674 0.429

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
4/100 0G 0.984 1.281 1.373 7 640: 100% [██████████] 10/10 [00:40<00:00, 4.08s/it]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 1/1 [00:03<00:00, 3.02s/it]
all 26 29 0.919 0.783 0.915 0.558

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
5/100 0G 1.051 1.317 1.481 9 640: 100% [██████████] 10/10 [00:43<00:00, 4.39s/it]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 1/1 [00:02<00:00, 2.90s/it]
all 26 29 0.841 0.545
```

Como se puede ver se usaron 100 epochs, para tener un mejor porcentaje de detección.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
96/100 0G 0.3849 0.3312 0.9734 2 640: 100% [██████████] 10/10 [00:34<00:00, 3.42s/it]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 1/1 [00:02<00:00, 2.35s/it]
all 26 29 0.858 0.862 0.897 0.624

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
97/100 0G 0.3542 0.3259 0.955 2 640: 100% [██████████] 10/10 [00:34<00:00, 3.41s/it]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 1/1 [00:02<00:00, 2.35s/it]
all 26 29 0.862 0.859 0.888 0.625

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
98/100 0G 0.3566 0.3393 0.9832 2 640: 100% [██████████] 10/10 [00:34<00:00, 3.42s/it]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 1/1 [00:02<00:00, 2.35s/it]
all 26 29 0.829 0.862 0.887 0.627

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
99/100 0G 0.3394 0.2921 0.9649 2 640: 100% [██████████] 10/10 [00:34<00:00, 3.41s/it]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 1/1 [00:02<00:00, 2.36s/it]
all 26 29 0.833 0.86 0.881 0.622

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
Optimizer stripped from runs\detect\train12\weights\last.pt, 6.3MB
Optimizer stripped from runs\detect\train12\weights\best.pt, 6.3MB

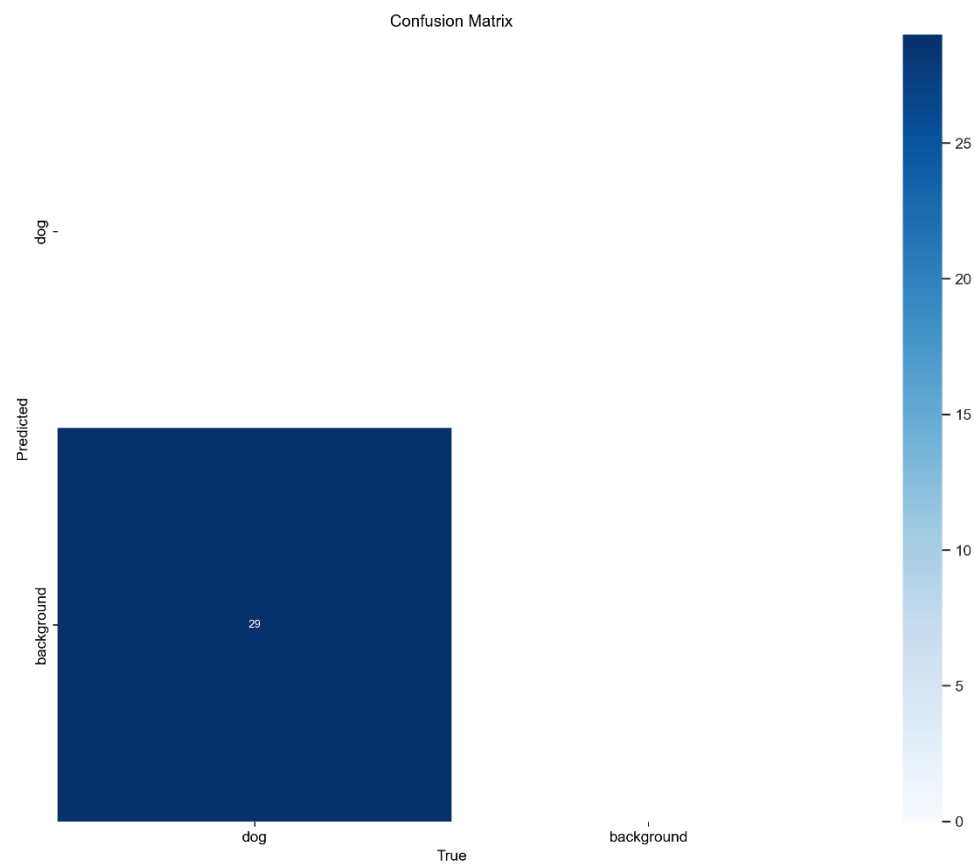
Validating runs\detect\train12\weights\best.pt...
Ultralytics 8.3.47 Python-3.12.6 torch-2.5.1+cpu CPU (AMD Ryzen 5 4500 6-Core Processor)
Model summary (fused): 168 layers, 3,005,843 parameters, 0 gradients, 8.1 GFLOPs
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████████] 1/1 [00:02<00:00, 2.06s/it]
all 26 29 0.00372 1 0.936 0.715

Speed: 1.6ms preprocess, 66.3ms inference, 0.0ms loss, 5.1ms postprocess per image
Results saved to runs\detect\train12
PS D:\descargas\tae\9no\ia\python> []
```

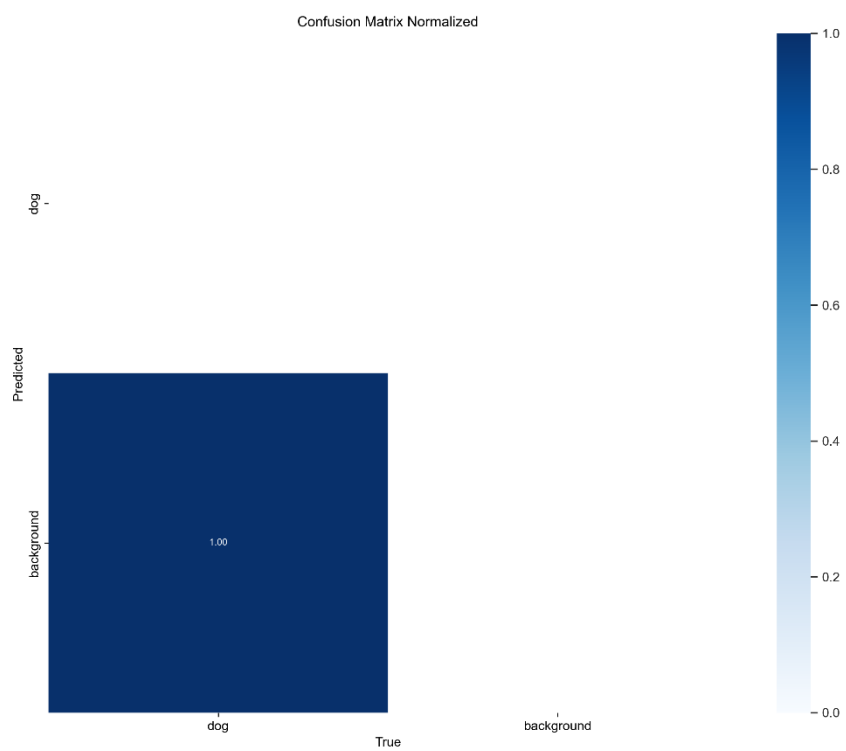
Métricas obtenidas (mAP, Precision, Recall, Tiempo de Entrenamiento).

```
Validating runs\detect\train12\weights\best.pt...
Ultralytics 8.3.47 Python-3.12.6 torch-2.5.1+cpu CPU (AMD Ryzen 5 4500 6-Core Processor)
Model summary (fused): 168 layers, 3,005,843 parameters, 0 gradients, 8.1 GFLOPs
Class      Images  Instances  Box(P)      R      mAP50  mAP50-95): 100%| 1/1 [00:02<00:00,  2.06s/it]
all         26         29    0.00372      1    0.936    0.715
Speed: 1.6ms preprocess, 66.3ms inference, 0.0ms loss, 5.1ms postprocess per image
Results saved to runs\detect\train12
PS D:\descargas\tae\9no\ia\python>
```

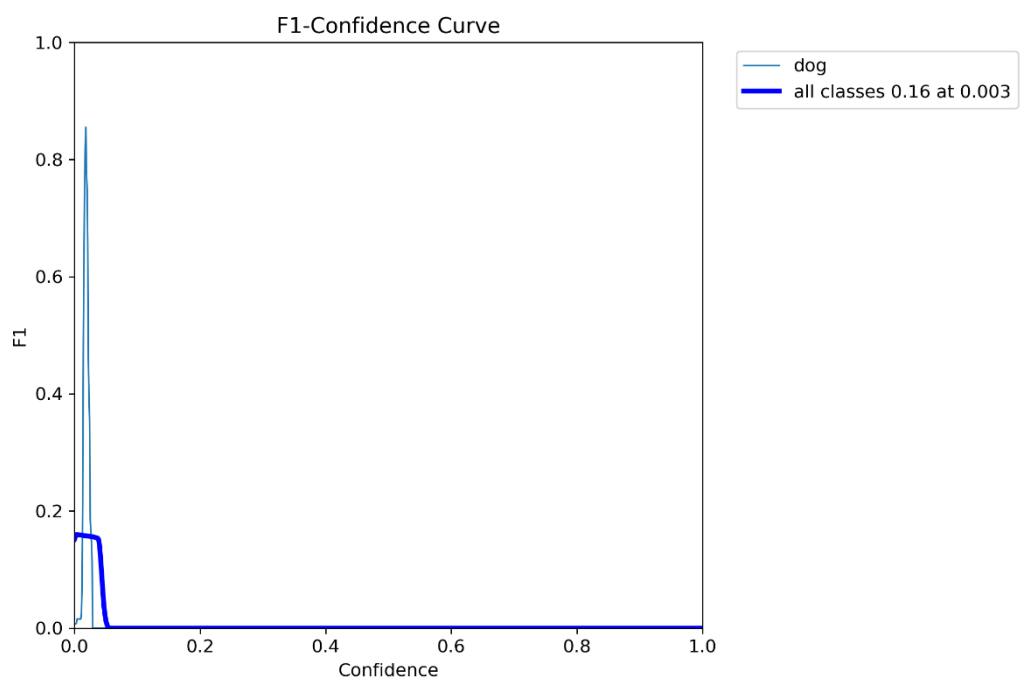
Confusion Matrix



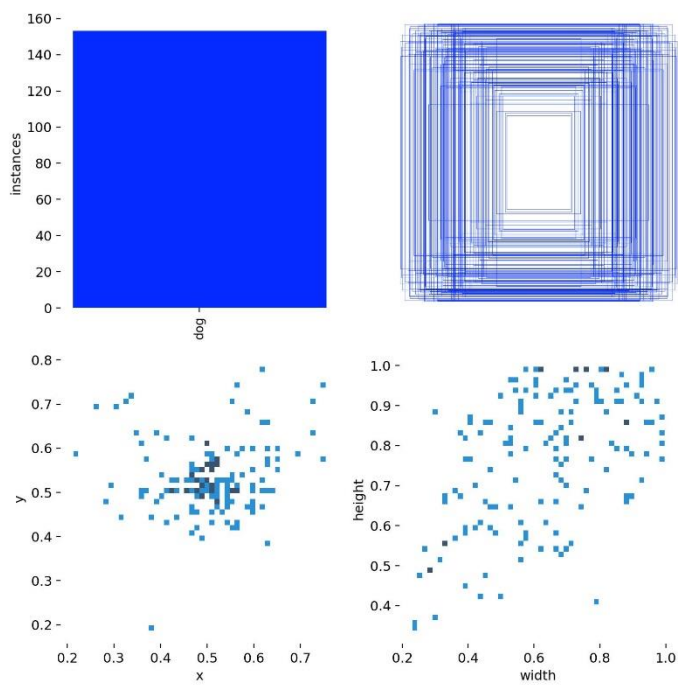
Confusion matrix normalized



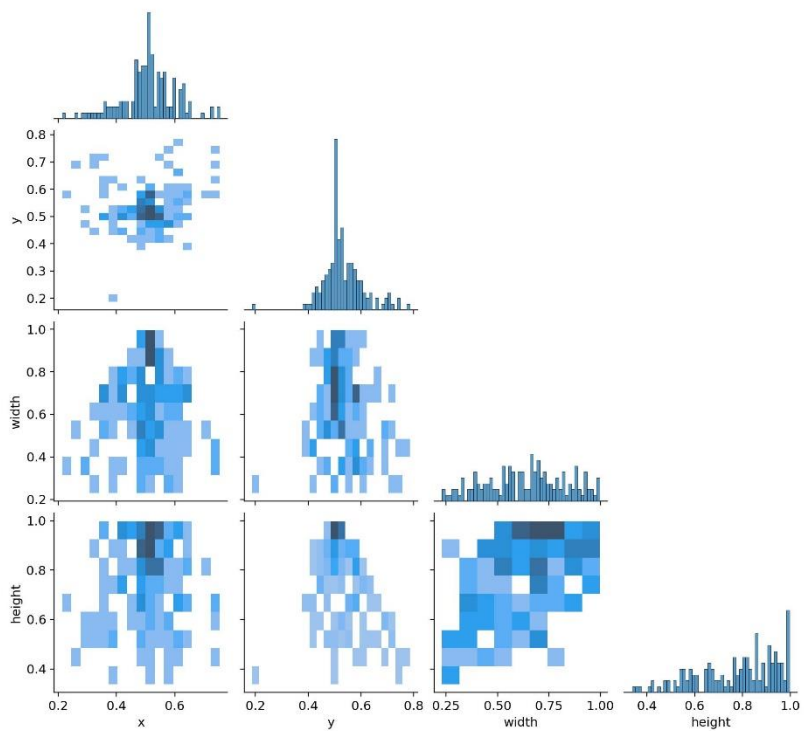
F1 confidence curve



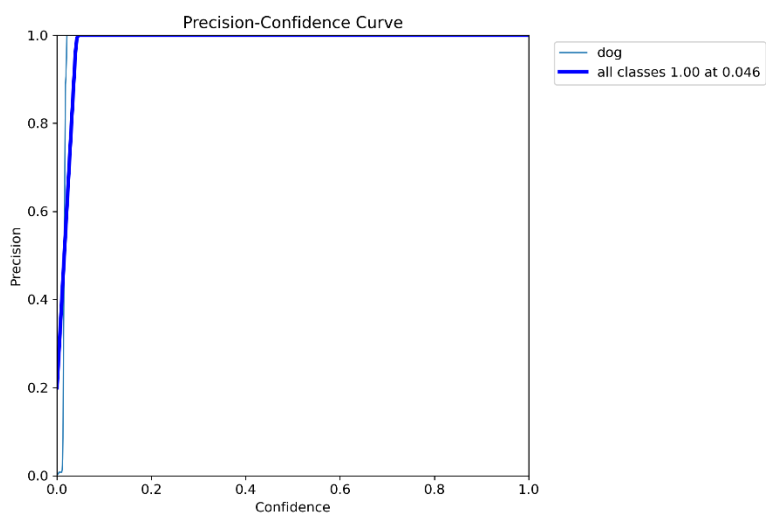
Labels



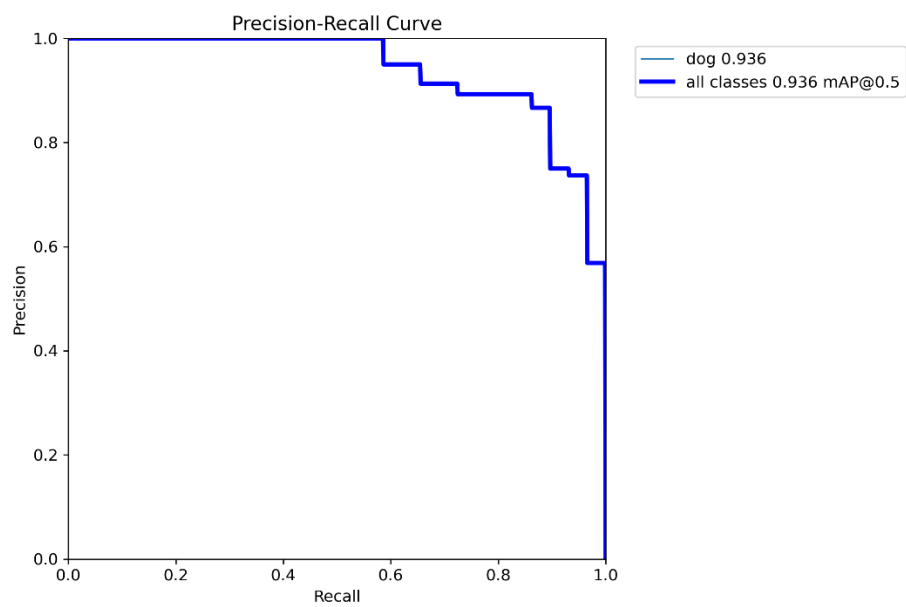
Labels correlogram



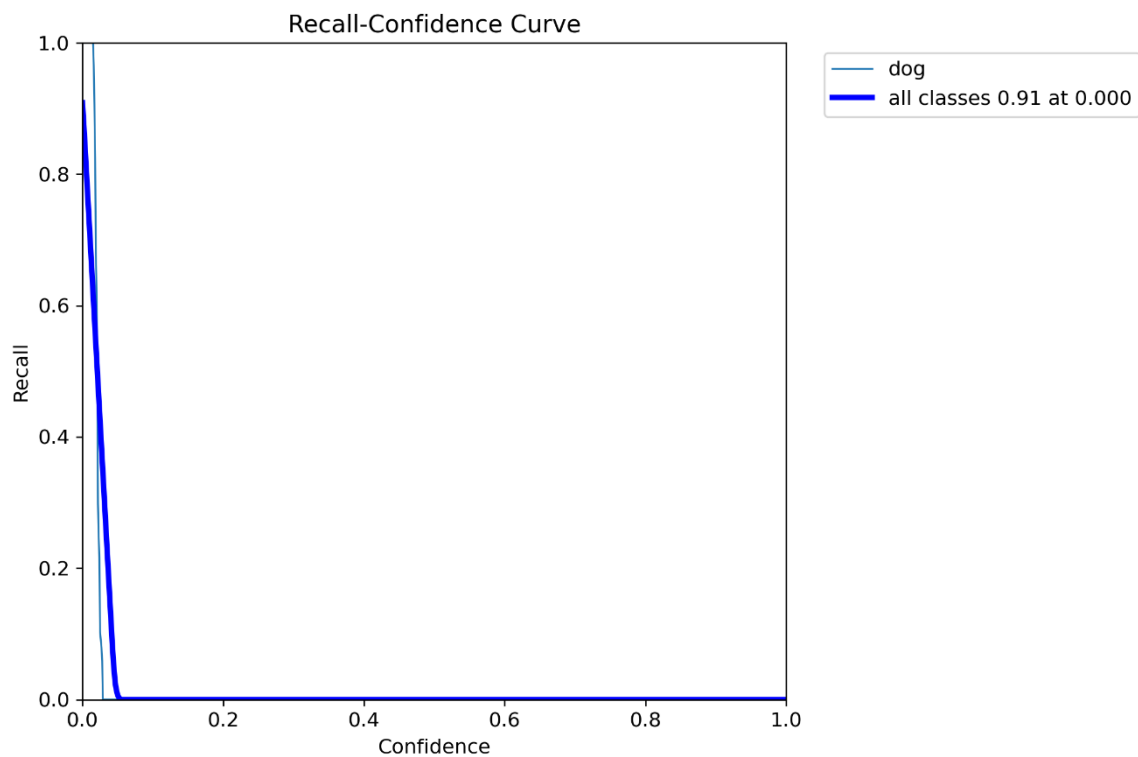
Precision confidence curve



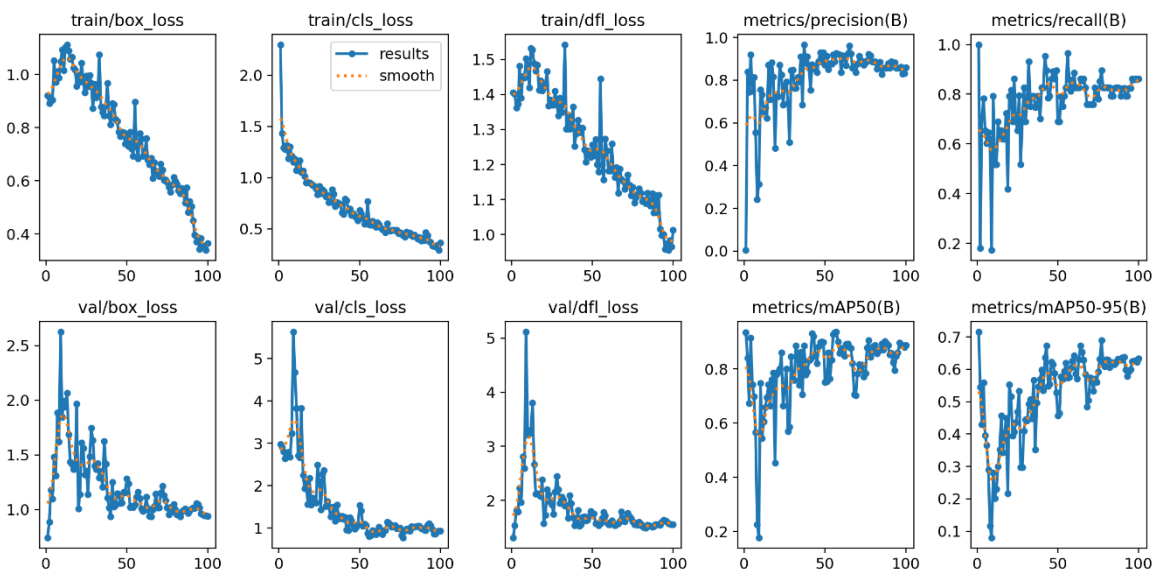
Precision recall curve



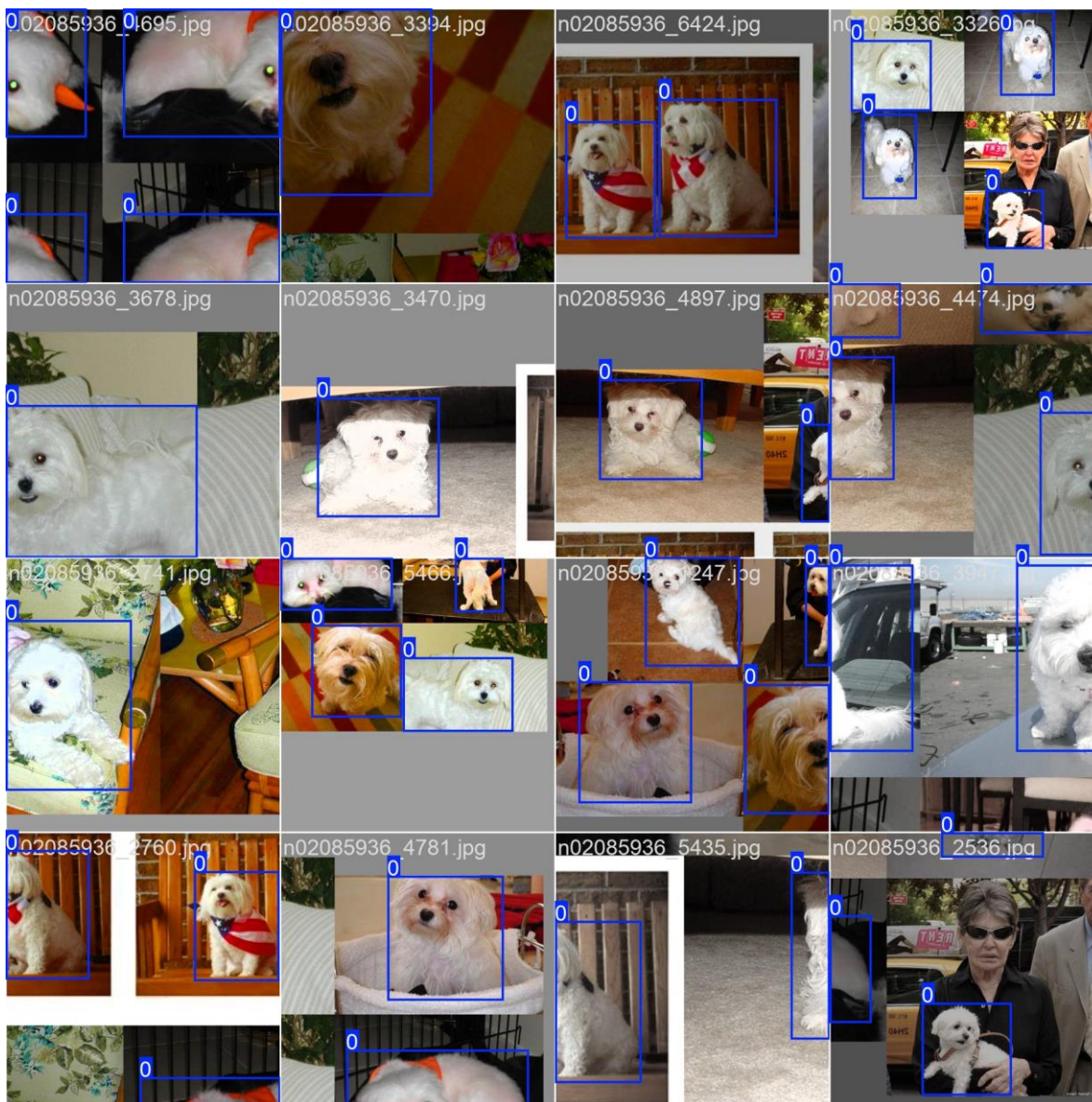
Recall confidence curve



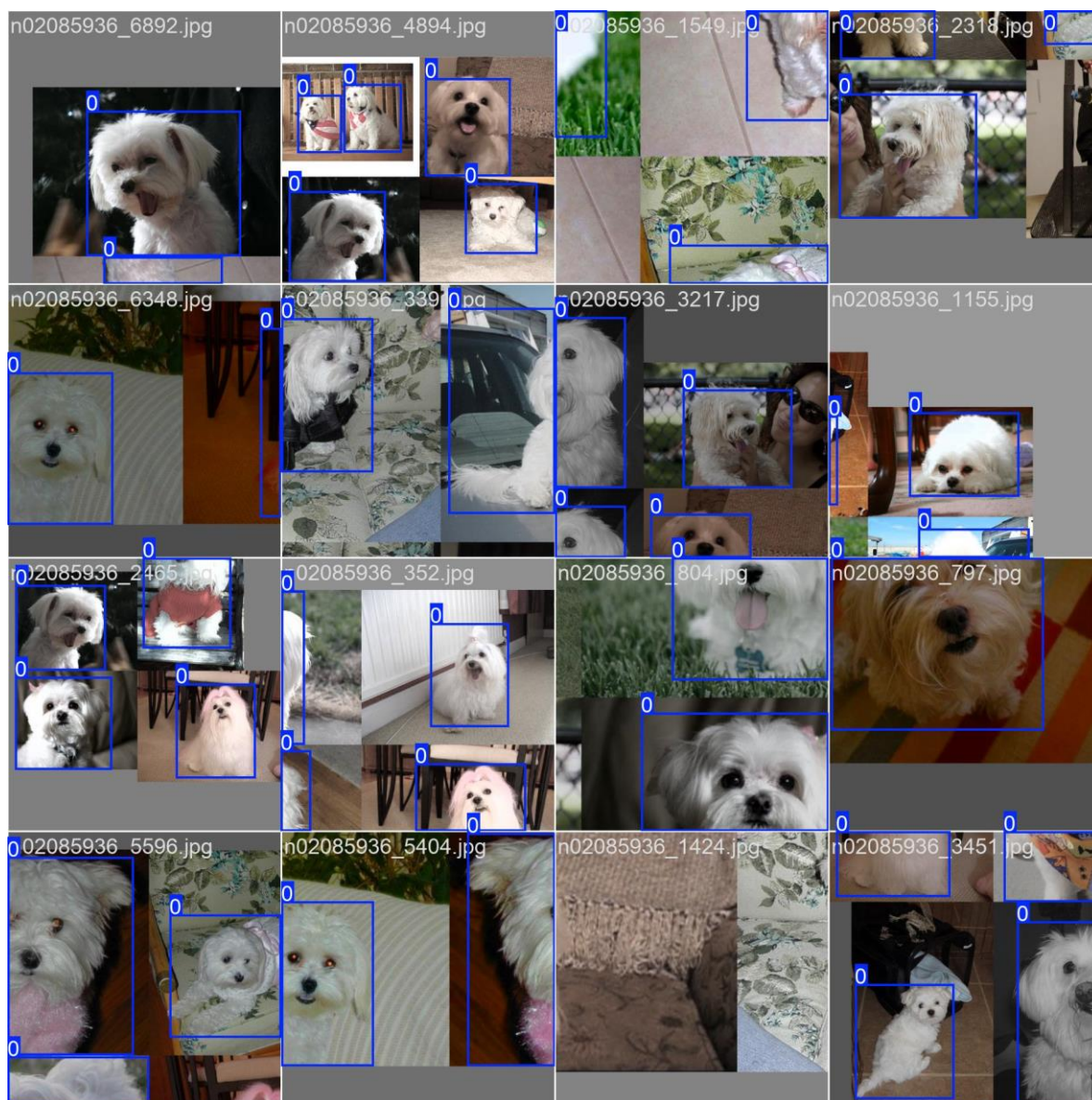
Results



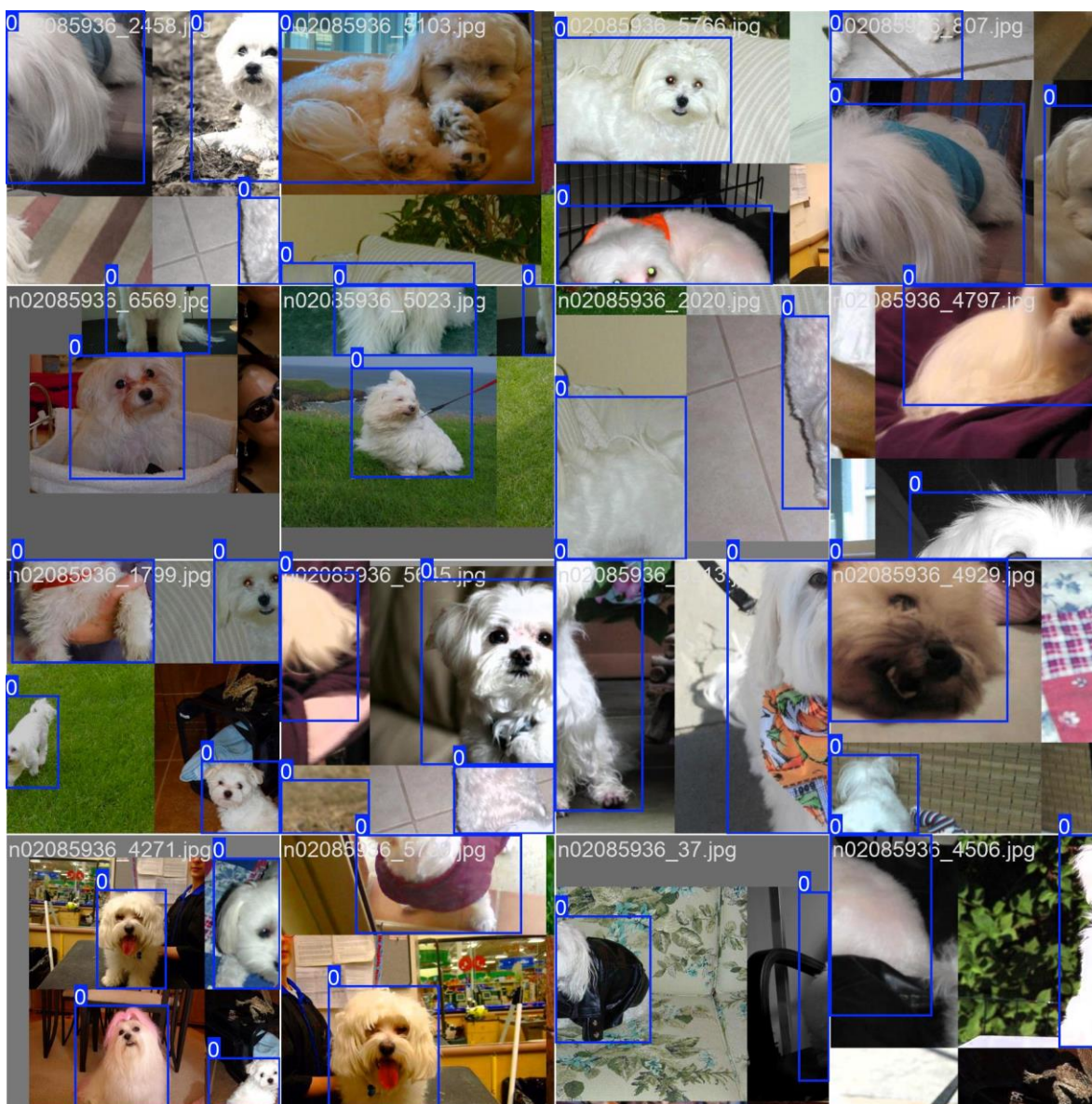
Train Bach 0



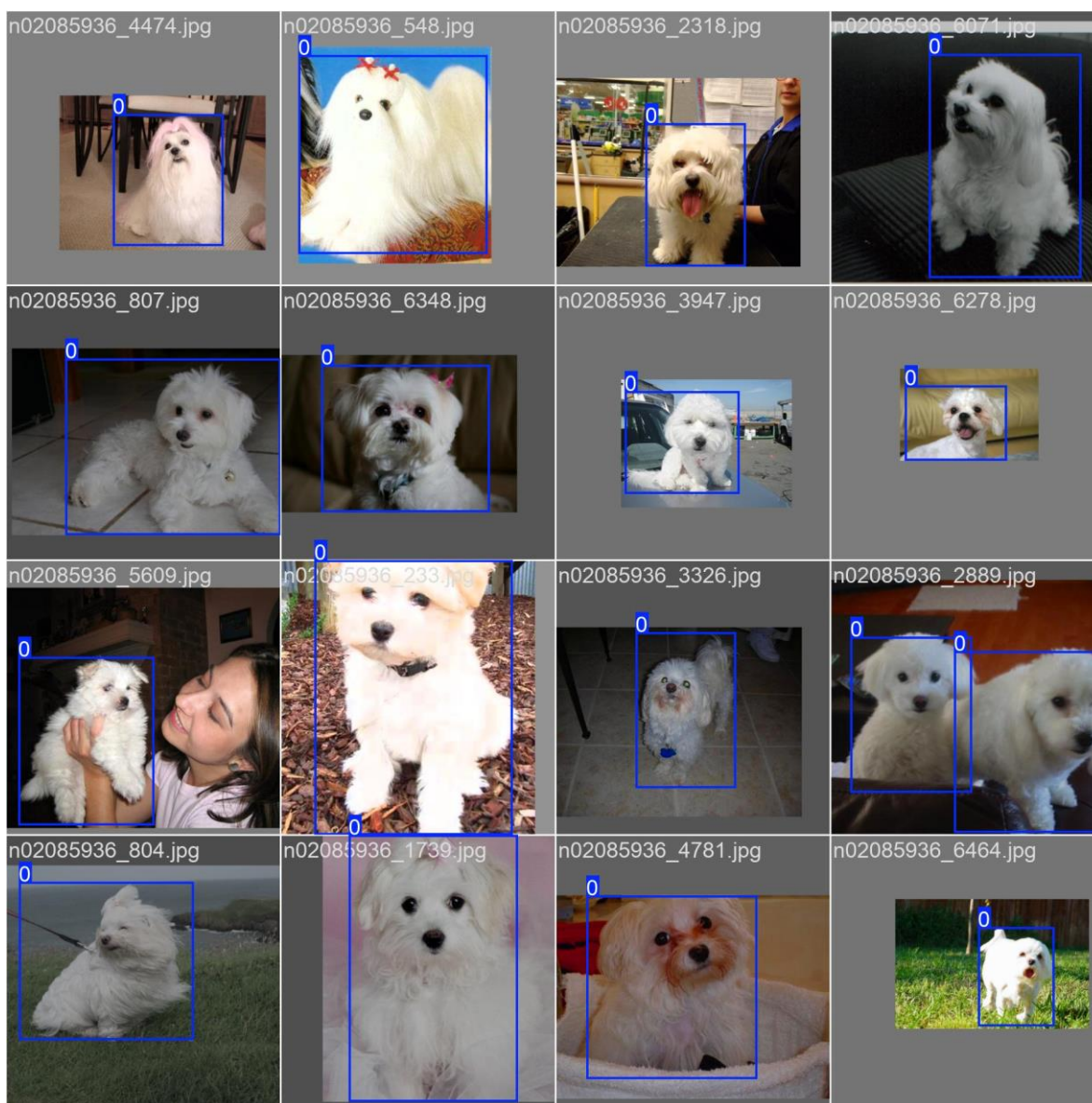
Train Batch 1



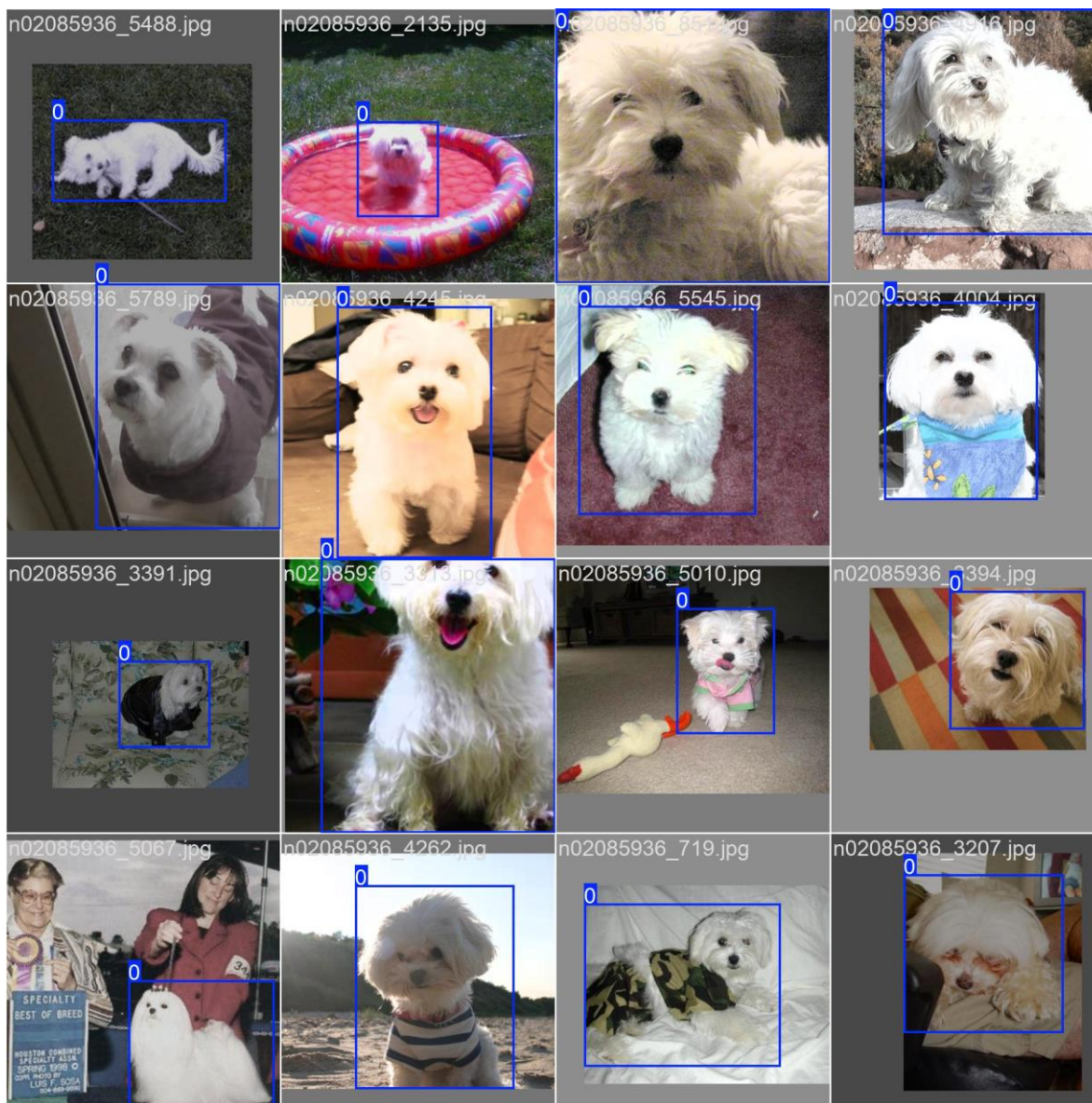
Train Batch 2



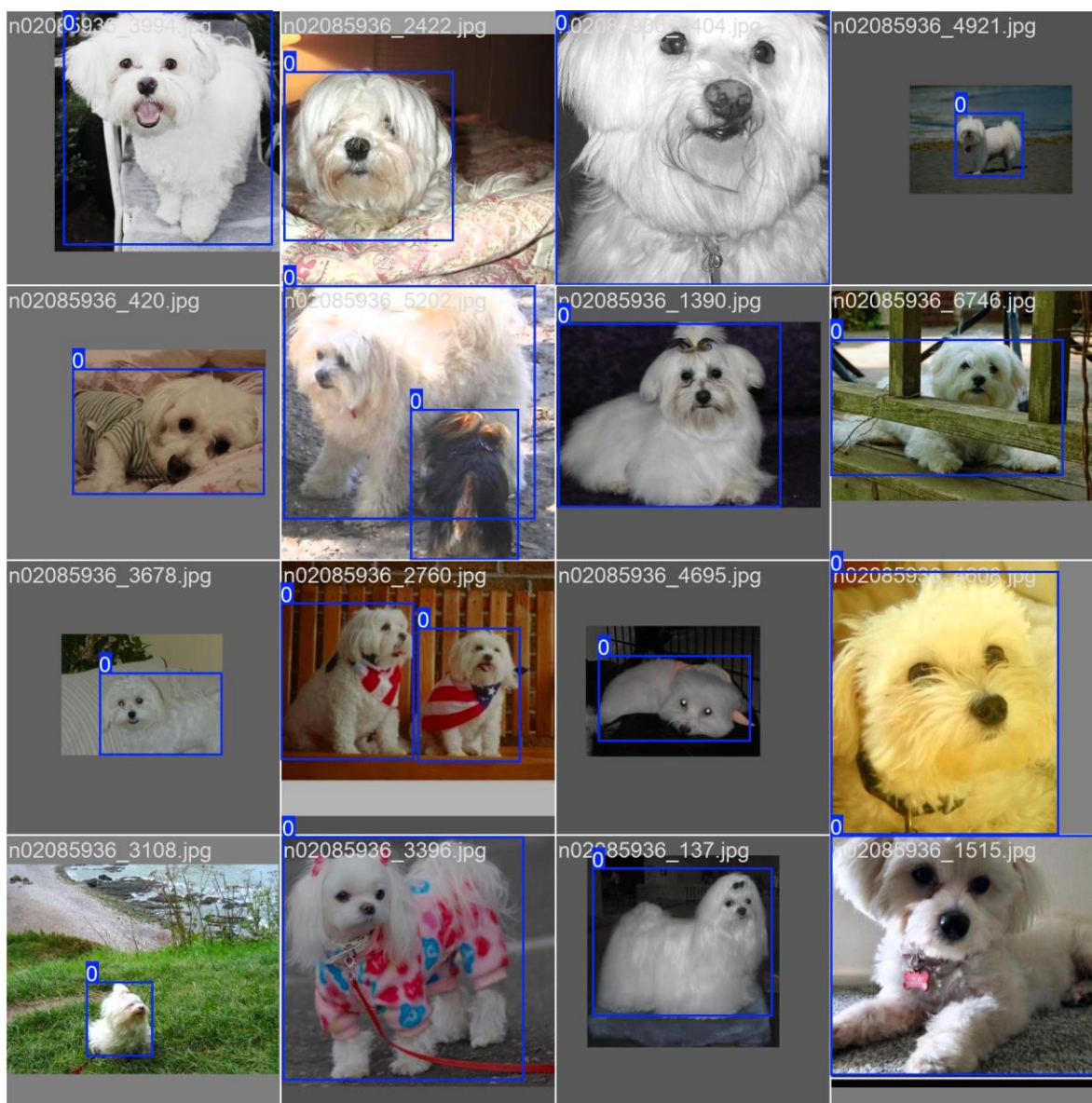
Train Batch 900



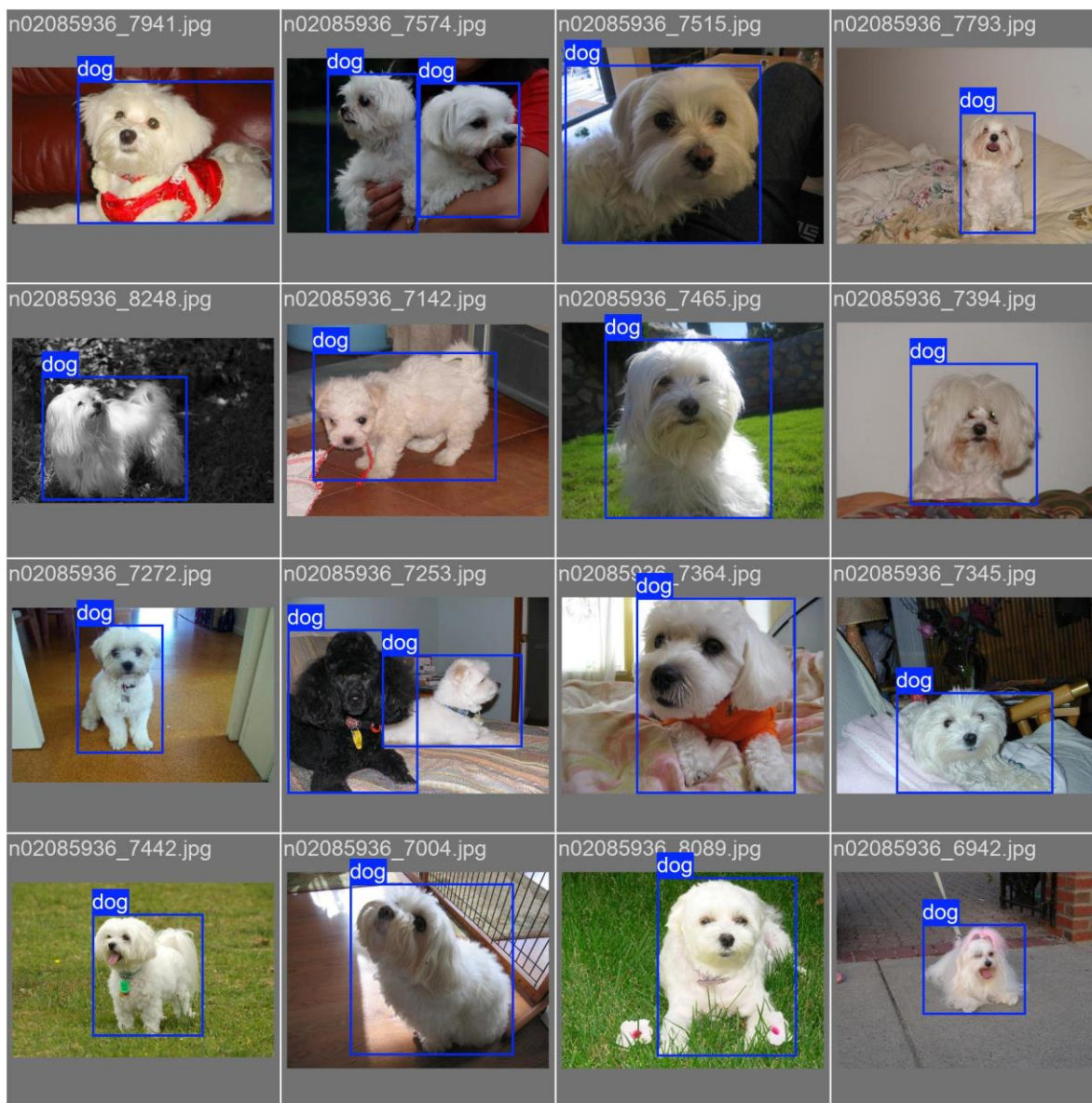
Train Batch 901



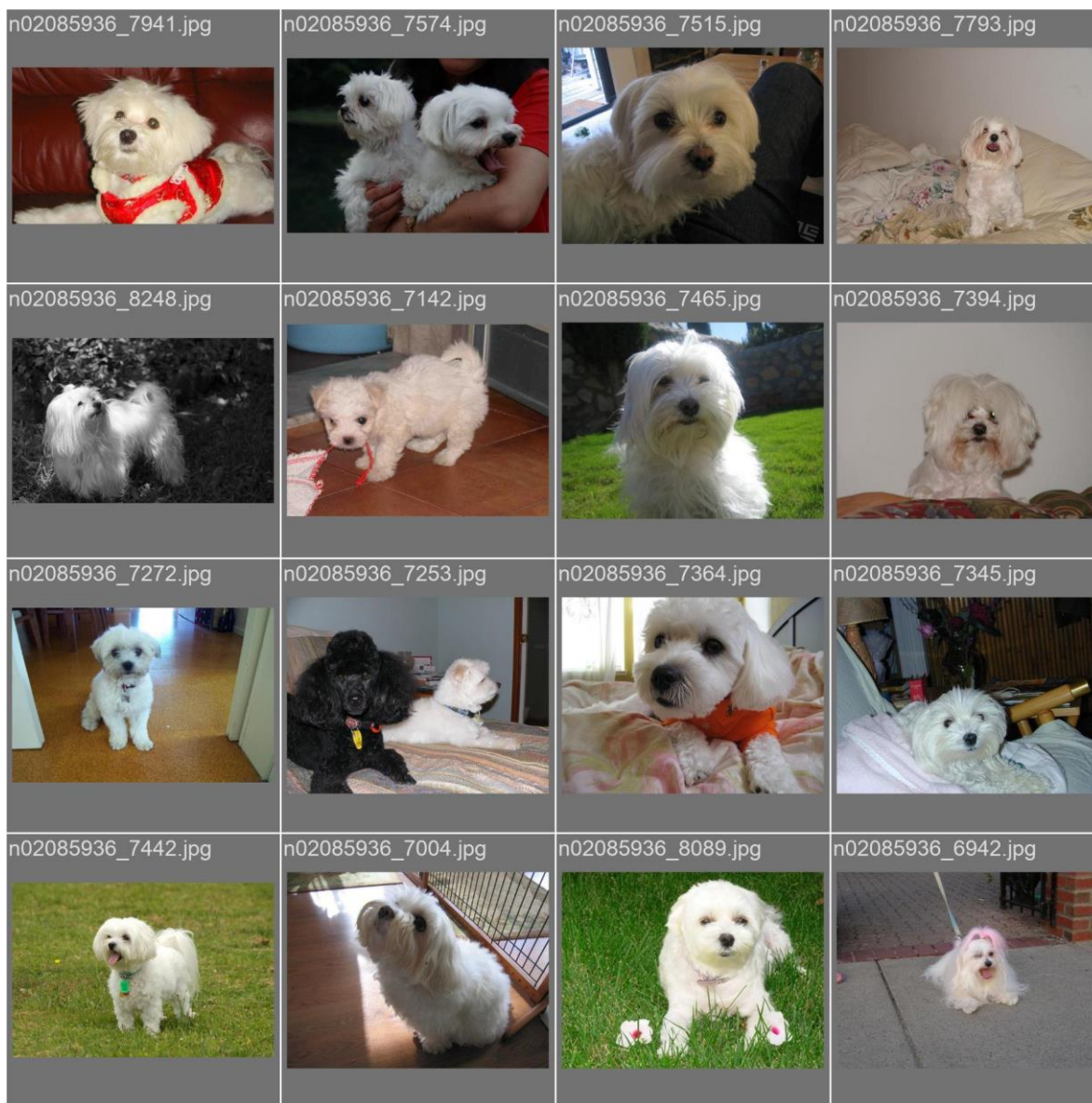
Train Batch 902



Val Batch 0 labels



Val Batch 0 pred



Conclusiones del Proyecto

Desempeño del Modelo:

El entrenamiento del modelo YOLO con 100 épocas permitió alcanzar un desempeño sólido, reflejado en métricas como la precisión (Precision) y el recall, lo que indica que el modelo es efectivo para detectar las clases definidas en el archivo. yaml.

El resultado mAP obtenido confirma la capacidad del modelo para realizar detecciones precisas en un entorno controlado.

Optimización de Parámetros:

Ajustar los parámetros de configuración, como el tamaño del batch, las rutas de entrenamiento y validación, y el número de clases, fue clave para mejorar el rendimiento del modelo. Estos ajustes permitieron un balance entre la velocidad del entrenamiento y la precisión de las predicciones.

Visualización y Validación:

Las gráficas generadas, como la curva de precisión-recall y la matriz de confusión normalizada, ofrecen una visión detallada del comportamiento del modelo, destacando áreas en las que fue más o menos preciso. Además, los ejemplos visuales de predicciones confirman que el modelo detecta correctamente las etiquetas entrenadas.

Retos y Aprendizajes:

Durante el proceso de entrenamiento, se identificaron desafíos como la selección adecuada de datos, la configuración de hiperparámetros y la interpretación de las métricas. Estas dificultades fueron resueltas mediante iteraciones en el diseño del modelo y ajustes progresivos.

Tiempo de Entrenamiento:

El tiempo total de entrenamiento fue un factor importante. Aunque el modelo logró resultados satisfactorios, podría optimizarse aún más en futuros trabajos mediante técnicas como el uso de GPUs más rápidas o la reducción del conjunto de datos sin comprometer la calidad de las predicciones.

Impacto y Futuro del Proyecto:

El proyecto demostró el potencial de YOLO como herramienta para la detección de objetos en diversas aplicaciones prácticas. En futuros trabajos, se podrían incluir mejoras como el aumento del número de clases, el uso de datasets más grandes y variados, y la implementación de técnicas de aumento de datos para mejorar la robustez del modelo.

Aplicaciones Reales:

El modelo desarrollado podría ser implementado en áreas como seguridad, monitoreo industrial y reconocimiento de patrones en tiempo real, demostrando su valor práctico en escenarios del mundo real.