1 Introduction

1.1 Problem

When identifying intruder aircraft from the perspective of an opposing aircraft, both image classification and object detection systems need to be robust and consistently perform at their maximum potential. Data augmentation presents a promising approach for enhancing performance, particularly when training data is limited. However, research that directly compares the effects of data augmentation on image classification versus object detection is scarce. It is not yet clear whether augmentation techniques that significantly improve image classification also boost object detection to a comparable degree, highlighting the need for investigation.

1.2 Goals

The goal of this project is to examine whether the most effective data augmentation techniques from image classification can enhance object detection performance, particularly for detecting intruder aircraft. This will be explored using samples of varying sizes from the AVOIDDS dataset [1], which simulates scenarios with limited training data where data augmentation would be more beneficial. Furthermore, the project will compare Image Classification performance using equivalently sized non-augmented and augmented training sets to determine if augmentation leads to better results when the number of training data points is consistent.

1.3 Scope

This project will investigate Data Augmentation with Image Classification, with different methods and percentages of Augmentation. Only the most effective augmentation configurations will be applied to object detection. The research will employ a custom model for image classification, constructed using TensorFlow's Keras framework, and will leverage a pre-trained YOLO model for object detection.

2 Literature Review

2.1 Key Machine Learning Concepts

2.1.1 Over-fitting and Under-fitting

Over-fitting occurs when a used model is too complex, it may perfectly fit the random noise of the data while attempting to catch the regularities [2]. A model that overfits the training data will perform well on the training set but poorly on the test set [3]. Under-fitting occurs when a given model is too simple to capture all regularities in the signal component [2].

Figure 1: Types of model fitting. [3]

There are many ways to prevent over-fitting [4]. These include:

- **Training with more data**.
- **Data Augmentation** (Section 2.3)
- **Adding noise to the input data**.
- **Feature selection**.
- **Cross-Validation**
- **Simplifying Data**
- **Regularization**
- **Ensemble Learning**

2.1.2 Testing and Training Machine Learning Models

A fundamental principle in machine learning is to avoid using the same dataset for both training and evaluation. To develop a reliable model, the dataset should be divided into training, testing, and validation sets [5]. The training set is used to train the model and learn the data patterns, requiring a diverse range of inputs to ensure comprehensive training. The validation set, distinct from the training set, is utilized to evaluate model performance during training. The test set is used to assess the model's performance after the training phase is complete. [5]

There is no optimal split percentage for deciding these splits. However, there are two major concerns when determining the optimal data split:

- Insufficient training data can lead to high variance in model training.
- Limited test/validation data can result in greater variance in model evaluation and performance statistics.

Despite this, there is a rough industry standard for splitting data sets shown in Figure 2.

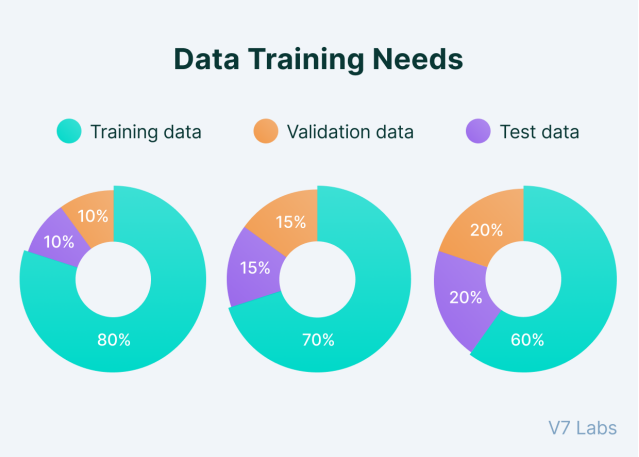

Figure 2: Standard Data Percentage Splits [5]

The process of splitting the dataset into these separate parts is also important to ensure a robust and fair method for testing machine learning models [5]. These advanced techniques for data splitting include:

- **Randomise**. However, a major drawback occurs when training with class-imbalanced datasets, as random allocation can create bias.
- **Stratified**. This method addresses class imbalance by maintaining the distribution in the train, test, and validation sets [5].
- **Stratified K-Fold Cross-Validation**. A robust data splitting technique where the dataset is divided into K folds. The model is trained on K-1 folds and tested on the remaining fold, repeated K times, while preserving class ratios to prevent distribution skewing [5].
- **Boot-Strapping**. Bootstrapping is a technique that involves generating multiple new samples from an existing dataset by repeatedly sampling with replacement. The samples can again be stratified to address class imbalance [6].

2.1.3 Convolution Neural Networks

One type of Neural Network architecture is the Convolution Neural Network (CNN). CNNs are similar to traditional feed-forward networks, however, they are designed to work with grid-structure inputs (i.e., a 2D image) [7]. The convolution operation involves performing a dot product between a grid-structured set of weights (kernel) and grid-structured inputs. This method is particularly useful for data with "a high level of spatial or other locality" like image data [7].

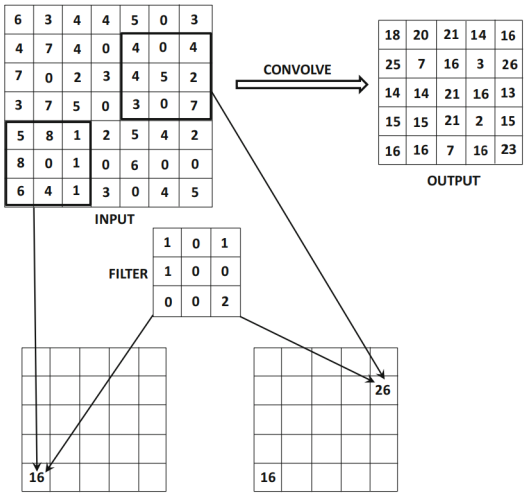

Figure 3: Image Convolution. [7]

2.1.4 Image Classification

Image classification is a technique that categorizes images into predefined classes based on their visual content [8]. There are many different approaches to Image Classification:

- **Support Vector Machines (SVMs)**. SVMs classify images by finding a hyperplane that best separates different classes in a high-dimensional space [9].
- **K-Nearest Neighbor (K-NN)**. Classifies images based on the majority vote of their nearest neighbours within a feature space [9]
- **Random Forest**. A method that ensembles decision trees to perform classification [9].
- **CNNs, Section 2.1.3**. CNNs automate feature extraction, using convolution and pooling layers to process images and classify them with high accuracy [9].

Two common performance metrics to evaluate Image Classifiers are Accuracy and F1 Score. Accuracy represents the proportion of correct predictions made by the classifier, while the F1 Score is a weighted average of precision and recall. [10]. These metrics allow for evaluating a model's peak performance by identifying the epoch that achieves the highest Accuracy or F1 Score.

2.1.5 Object Detection

Object detection combines image classification and localization to determine what objects are in the image or video and specify their locations [11].

Figure 4: Left - Classification. Right - Object Detection [11]

Object detection algorithms are generally classified into two categories: Single-Shot (One-Stage) and Two-Stage detectors (Figure 5). Single-shot detection involves a single pass of the input image to predict the presence and location of objects. This makes them very efficient but generally less accurate than other methods. Two-stage detection involves two passes of the input image. The first pass generates a set of potential object locations, and the second pass refines these proposals to make final predictions. Consequently, two-stage algorithms tend to be more accurate. [12].

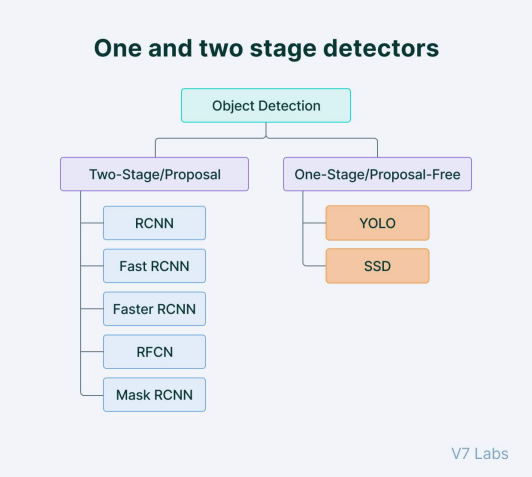

Figure 5: Object Detection Categories. [12]

The two most common evaluation metrics for comparing the predictive performance of object detection models are Intersection over Union (IoU) and Average Precision (AP). IoU is used to measure localization accuracy and identify localization errors. To calculate IoU between the predicted and ground truth bounding boxes, you divide the area of their intersection by the area of their union. (Figure 6) [12].

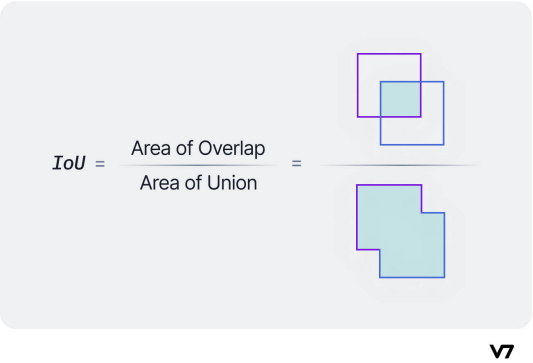

Figure 6: Intersection over Union. [12]

AP is calculated using a precision versus recall curve from a set of predictions. Recall is the proportion of actual positive class samples correctly identified by the model (e.g., if the test set has 100 positive samples and the model identifies 60, the recall is 60%). Precision is the proportion of true positives among the total predictions made by the model. The area under the precision versus recall curve gives the AP for each class. The average of these AP values across all classes is called the mean Average Precision (mAP). [12].

For object detection, precision and recall are used for measuring the decision performance (not for predictions). Instead, an IoU value > 0.5 is considered a positive prediction and IoU < 0.5 is negative [12].

2.2 AVOIDDS Dataset

The AVOIDDS dataset is comprised of 72,000 images and corresponding labels, depicting encounters with intruder aircraft from the perspective of the ownship in the airspace [1]. Data generation is carried out using X-Plane 11, enabling programmatic control over environmental conditions and intruder locations. The images are evenly distributed across six weather types, three aircraft types, and four regions. (Figure 7).

A baseline YOLOv8 model was trained on the AVOIDDS dataset for 100 epochs. The training process took 73 hours using an NVIDIA GeForce GTX 1070 Ti [1]. The model achieved an mAP of 0.866 overall with a precision of 0.990 across all categories, the results are shown in Figure 8. The largest difference in performance arose as the aircraft moved further away from the point of view, as well as this, the model has a higher mAP for the larger aircraft (Boeing 737-800, KingAir C90).

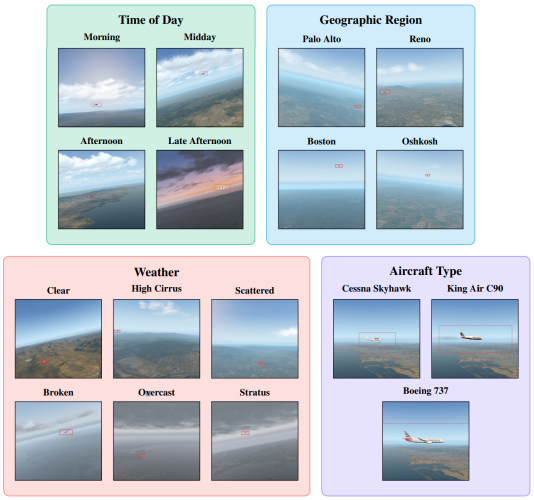

Figure 7: Variation in AVOIDDS images. [1]

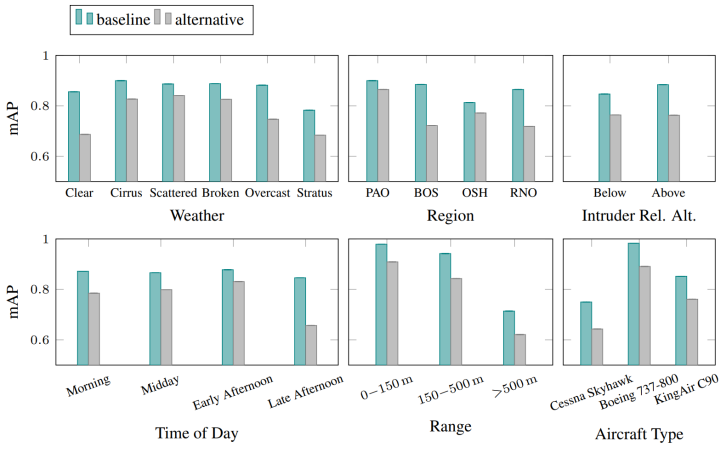

Figure 8: Results of YOLOv8 baseline model on AVOIDDS. [1]

2.3 Data Augmentation

Data augmentation involves generating additional data from the original dataset through various transformations. This augmented data is then added back to the training set, creating a more diverse set of data points and minimizing the difference between the training and validation sets [13]. Krizhevsky employed data augmentation in their experiments, increasing the dataset size by a factor of 2048 [14] by horizontally flipping images and altering the intensity of RGB color channels using PCA color augmentation. These augmentation techniques reduced the model's error rate by over 1% [13].

2.3.1 Different Data Augmentation Methods

A popular practice of data augmentation is combining affine image transformations, such as rotation, reflection, scaling and shearing (Figure 9), with colour

modification. Colour modification includes geometric distortions or deformations such as histogram equalisation, contrast or brightness enhancement, white-balancing, sharpening and blurring (Figure 10). These methods have been proven effective for expanding the training dataset. [15].

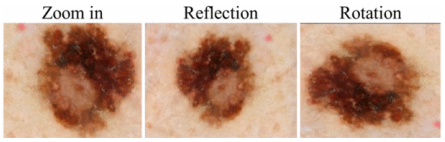

Figure 9: Affine Transformations. [15]

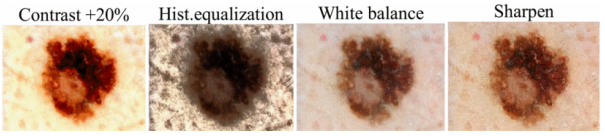

Figure 10: Colour Modifications. [15]

When investigating data augmentation for a deep learning-based model in pathological lung segmentation, M. S. Alam, D. Wang, and A. Sowmya suggest that standard affine transformations provide limited performance improvement within Deep Neural Network (DNN) architecture [16]. Instead, they generated new images using Gaussian filters and applied varying levels of contrast and brightness to create images with high opacity and low contrast. They obtained the best performance when using 250% augmented images (from 506 to 1,265). Similarly, Y. Mi, S. Tabirca, and A O'Reilly augmented their data by adding Gaussian noise, creating subtly varied instances to add to the dataset. They found this to mitigate the risk of overfitting in their models [17].

2.3.2 Data Augmentation in Python

Albumentations [18] is a Python package - "widely used in industry" - which can be used to perform several different image transformations for Data Augmentation, including both affine transformations and colour modifications. For Object Detection, Albumentations has methods for manipulating bounding box coordinates: the **bboxparams()** method can be implemented into a transformation pipeline to augment the bounding box with a specified format - such as YOLO (Section 2.5) [18]. When tested against augmentation frameworks from Keras, Pytorch and 'imgaug', Albumentations was "consistently faster than all alternatives" [18].

Below demonstrates how a selection of the previously mentioned data augmentation methods (Section 2.3.1) can be implemented using Albumentations

- **Reflections** can be performed using the **HorizontalFlip()** and **VerticalFlip()** methods [18].
- **Rotation** can be performed using the **Rotate()** method. This method takes an angle limit parameter in the form of a tuple (minimum rotation,

maximum rotation), an angle is randomly generated between these limits, ensuring that each augmented image is unique [18].

- The **Crop()** and **Resize()** methods can be used to perform **Zoom** transformations, however, crop coordinates will need to be manually calculated before the method can be called in an augmentation pipeline [18].
- Traditional Histogram equalisation can cause contrast issues between the background and foreground [19]. **Contrast Limited Adaptive Histogram Equalization (CLAHE)** solves this problem by dividing an image into small blocks [19]. This can be implemented using the **CLAHE()** method in Albumentations [18].
- **Gaussian Noise.** Albumentations contains the **GaussNoise()** method to add Guassian Noise to an Image. Similarly to Rotation, there is a tuple parameter for the variance limits of the noise [18].

Other methods do not require the Albumentations python package:

- **Sharpening** an image can be performed using image convolution with a specific 2D kernel [20], figure 11 shows various sharpening kernels. The *cv.filter2D()* function in OpenCV can be used to perform convolution efficiently with a given kernel [21].

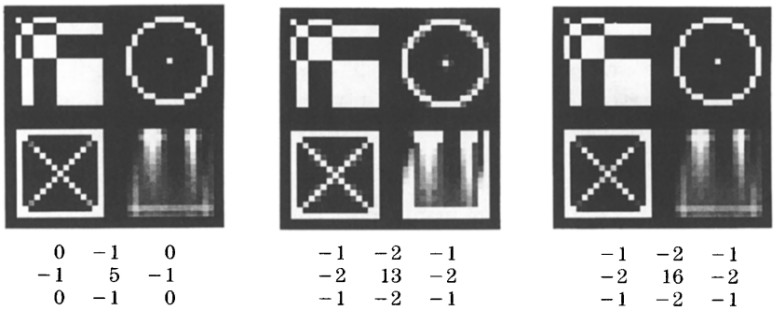

Figure 11: Kernels for Image Sharpening. [20]

- The point processes of multiplication and addition with a constant can be used to control the **contrast and brightness** of images.

$$g(x) = \alpha f(x) + \beta$$

  Adjusting the gain(α) and bias(β) will control the contrast and brightness respectively [22].

- **White Balancing** can be performed in using the Grayworld Algorithm [23].

2.4 TensorFlow Image Classification

TensorFlow is an open-source library developed by Google for numerical computation and large-scale machine learning [24]. Keras is a powerful deep learning library [25] that serves as TensorFlow's high-level API [24], designed to facilitate

the building and training of neural network models, such as CNNs for Image Classification. TensorFlow 2.10 was the last version of TensorFlow to support GPU usage on native Windows, from TensorFlow 2.11, TensorFlow for WSL2 must be used for Windows GPU support [24].

2.4.1 Building a Keras Model

Datasets can be loaded using the **tf.keras.utils.image_dataset_from_directory** utility [24], converting a directory of images on disk into a **tf.data.Dataset**. The required directory format for a TensorFlow Dataset is shown in Figure 12.

```
main_directory/
...class_a/
......a_image_1.jpg
......a_image_2.jpg
...class_b/
......b_image_1.jpg
......b_image_2.jpg
```

Figure 12: Directory format for TensorFlow Dataset. [24]

The Keras Sequential Model can build a CNN for Image Classification [24], The Keras Sequential Model can build a CNN for Image Classification in Figure 13. It starts with a 'Rescaling' layer to normalize pixel values between 0 and 1, enhancing training stability. Conv2D layers with ReLU activation then extract features, followed by MaxPooling2D layers that reduce spatial dimensions to lower computational demands and mitigate overfitting. A 'Flatten' layer converts these maps into a flat vector, feeding into Dense layers that output the model's final predictions.

```
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Figure 13: Example Keras Sequential Model for Image Classification. [24]

After the Keras Model has been built and compiled, it can be trained using the **model.fit()** method [24]. Following this, the **model.predict()** method

can be used to predict the validation set using the trained model, to evaluate performance.

2.5 You Only Look Once (YOLO)

YOLO is an open-source object detection system. It uses a single CNN (Section 2.1.3) to predict multiple bounding boxes and their corresponding class probabilities simultaneously, as the name states, 'You Only Look Once' at an image [26]. This makes YOLO a one-stage object detection system (Section 2.1.5) as opposed to previous two-stage systems such as the Region-Based CNN (R-CNN), which requires additional refinements, duplicate elimination and re-scoring of boxes based on other objects in the scene [26]. Due to its simplicity, YOLO is extremely fast [26]. The base network operates at 45 frames/images per second (fps) when training on PASCAL VOC 2007 [27] without batch processing. This performance was achieved using a Titan X GPU, with the fast versions of YOLO exceeding 150 fps [26] (Figure 14).

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Figure 14: Object Detection Systems Speed Comparison. [26]

Error analysis of the VOC 2007 run demonstrates YOLO's accuracy in comparison to R-CNN. For each category during testing, the top N predictions for that category are considered. The results of this test are shown in Figure 15 with the correctness of predictions being classed as the following [26]:

- **Correct:** correct class and IOU > 0.5
- **Localization:** correct class, $0.1 >$ IOU > 0.5
- **Similar:** class is similar, IOU > 0.1
- **Other:** class is wrong, IOU > 0.1
- **Background:** IOU > 0.1 for any object

Fast R-CNN is narrowly better at correctly predicting objects, however, it makes far more background errors (due to R-CNN using a sliding window and region proposal-based techniques). YOLO struggles to localise objects correctly, with over twice as many localisation errors [26].
There have been 8 iterations of YOLO since its initial release in 2016, each iteration incrementally improves the speed and accuracy of the system, often by improving the CNN architecture [12]. Version 2 also introduced anchor boxes; these are predefined bounding boxes with various aspect ratios and scales which

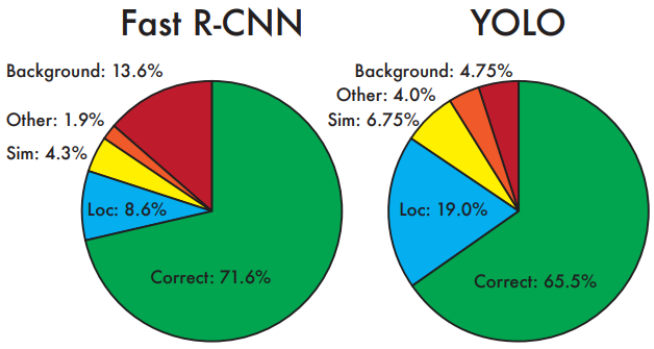

Figure 15: Error Analysis: Fast R-CNN vs YOLO. [26]

help to determine final bounding boxes when combined with predicted offsets [12]. YOLOv7 utilizes nine anchor boxes, enabling it to detect a broader range of object shapes and sizes, thereby reducing the number of false positives [12]. YOLOv7 also increased the resolution that it processes images at to 608 by 608 pixels, helping it to detect smaller objects and increase overall accuracy [12]. YOLOv8 is currently the latest version of YOLO; it was created by Ultralytics. The code for YOLOv8 is open source and licensed under a GPL license [28].

2.5.1 Limitations of YOLO

Some limitations of YOLO are important to consider:

- YOLO struggles to detect small objects or objects that appear in groups [26] (the latter is unlikely to affect this project as each image contains one aeroplane [1]).
- YOLO struggles generalizing to objects with unfamiliar aspect ratios or configurations [26].
- YOLO can be sensitive to variations in lighting or other environmental conditions [12].
- YOLO (like all object detection) is computationally intensive, especially on resource-strained devices [12].

2.5.2 How to use YOLO

YOLO is written in Python and can be accessed via installing the Ultralytics package (YOLOv8 [28]) or by cloning the corresponding git repository (YOLOv7 [29] and YOLOv8 [30]). YOLO can be implemented using both the command line interface (CLI) for quick and easy training as well as the Python interface so that YOLO can be implemented into larger projects [31].

Custom datasets must be formatted correctly to work with YOLO:

- The dataset must be arranged in the correct folder structure (Figure 16) [31].

```
dataset/
├── train/
│   ├── images/
│   └── labels/
└── val/
    ├── images/
    └── labels/
```

Figure 16: Dataset Folder Structure. [31]

- Labels must be provided in the form of a single ***.txt** file per image (if there are no objects in the image, no file is necessary). Each text file should be formatted with one row per object in the format: class x_{centre} y_{centre} *width height*. Box coordinates must be in normalized **xywh** format (ranging from 0 to 1). An example label text file for a specific image is shown in Figure 17 [31].

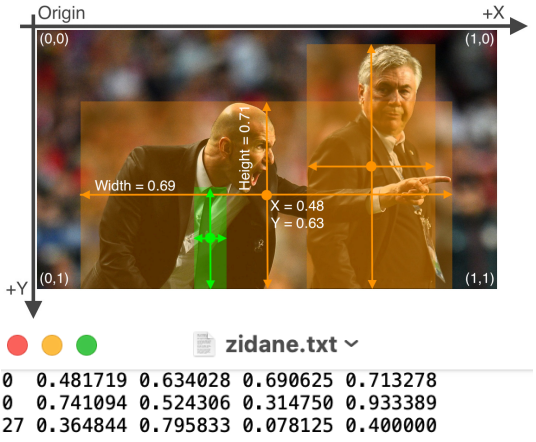

Figure 17: Example label file for image. [31]

- There must be a **data.yaml** file in the root directory of the dataset that describes the dataset (shown in Figure 18) [31].

```
path: ../datasets/starter_dataset
train: images/train
val: images/valid
names:
  0: aircraft
```

Figure 18: Example .yaml file for AVOIDDS dataset. [1]

- Images of a dataset can be optimized to reduce the size of the dataset for more efficient processing. The tools required for this are part of the Ultralytics Python package [31].

Figure 19 shows example Python code for training using YOLOv8. A pre-trained YOLO model is used to train the on the **coco128** dataset [32]; the model is trained on the training set for 3 epochs. The epoch count specifies how many times the training dataset is processed by the learning algorithm [33]. The model (weights) used in this code is the **yolov8n.pt** file however there are many pre-trained (default) models available in YOLO (Figure 20).

```
from ultralytics import YOLO

# Create a new YOLO model from scratch
model = YOLO('yolov8n.yaml')

# Load a pretrained YOLO model (recommended for training)
model = YOLO('yolov8n.pt')

# Train model using 'coco128.yaml' dataset for 3 epochs
results = model.train(data='coco128.yaml', epochs=3)

# Evaluate the model's performance on the validation set
results = model.val()

# Perform object detection on an image using the model
results = model('https://ultralytics.com/images/bus.jpg')

# Export the model to ONNX format
success = model.export(format='onnx')
```

Figure 19: Example Python Code for training with YOLO. [31]

The **YOLOv8n** model is the fastest (80.4ms with for CPU ONNX test) as well as the least accurate (mAP of 37.3). Whereas, **YOLOv8x** is the most accurate (53.9 mAP) as well as the slowest (479.1ms with for CPU ONNX test).

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 20: Different Models in YOLOv8. [30]

2.6 Other Object Detection Systems

There are other possible frameworks for implementing Object Detection, allowing developers to build custom models or use pre-existing. For example, PyTorch offers robust capabilities distinct from specific model architectures like YOLOv8 [34]. PyTorch allows for object detection models such as Faster R-CNN to be built from scratch (similar to TensorFlow) [34].

3 Risk Analysis

This section analyses each risk with a probability & severity rating and how we plan to mitigate each one.

ID	Risk	P (1-5)	S (1-5)	RE (P × S)
R1	Training time is high	5	4	20*
R2	Models over-perform datasets	2	3	6
R3	Data Augmentation does not improve performance	3	2	6
R4	Bias arising from unwanted sources	3	4	12
R5	Pressure from other modules	4	2	8

Table 1: Risk Analysis. P - Probability, S - Severity, RE - Risk Exposure *Largest Risk Exposure

R1: Plan training and test strategy in detail; use the prototyping phase to get an estimate of the time cost of training.

R2: The Keras Image Classifier can be tweaked to be slightly less effective. If YOLOv8 performs too well on initial tests, leaving less room for improvement, then an older version can be used or model features and parameters can be adjusted/disabled to worsen performance.

R3: Important to document and make conclusions on all results to find possible underlying reasons for this.

R4: Make sure variables such as training/test set sizes, dataset stratification, number of epochs, models used and the number of bootstraps are controlled between tests and are documented.

R5: Organise the schedule (Section 9) so that it allows time for other module commitments when necessary.

4 Specification

4.1 Requirements

The system requirements are listed below:

1. The system must be able to create subsets of the AVOIDDS Dataset of a chosen size.
2. The system must be able to augment datasets.
3. The user must be able to specify the augmentation metadata for an augmented dataset.
4. The system must build and compile a model for image classification with Keras.
5. The system must be able to train an image classifier model with augmented and non-augmented (pure) datasets.
6. The system must evaluate image classification using F1 Score.
7. The system must enable the user to determine an optimal data augmentation configuration for image classification, to be leveraged with object detection.
8. The system must utilize the YOLOv8 system.
9. The system must be able to train YOLOv8 with augmented and pure datasets.
10. The system must be able to evaluate YOLOv8 using mAP at different thresholds and F1 Scores.

4.2 Experiment Specification

A detailed individual experiment design is described in Section 5.3, this section highlights key details. The experiment will use datasets with train sizes of 1000, 500, and 2500, as agreed with the Project Supervisor. This decision is based on the following reasons:

1. The need for Data augmentation is greater with limited training data, which can be simulated using smaller samples from the AVOIDDS dataset.
2. The models performed well on smaller datasets (Section 5.1), so using smaller samples will highlight the impact of augmentation on various dataset sizes.
3. Smaller sample sizes reduce test time, enabling more tests to be conducted.

Additionally, the Project Supervisor emphasized the importance of using specific stratification with the AVOIDDS metadata when employing smaller sample sizes.

The experiment will use augmentation percentages of 50%, 150%, and 250% spanning from mostly pure images to mostly augmented images. The following

augmentation techniques will be used: Reflections, rotations, brightness and contrast modifications, histogram equalisation, white balancing, sharpening, Gaussian noise, and zooming.

Results from the Image Classifier (CLF) will be used to propose an Augmentation Strategy for Object Detection. For the image classifier, F1 Score is utilized to assess performance. This metric is preferred over accuracy because accuracy can be misleading, particularly in datasets with an even distribution of classes, as it only reflects the proportion of correct predictions without considering false positives and false negatives. F1 Score is more suitable as it ensures the classifier maintains a high level of accuracy in both detecting true positives and correctly ignoring negatives.

Object detection will be evaluated using F1 Score and Mean Average Precision (mAP). The F1 Score facilitates a consistent assessment of classification performance across both systems. Additionally, mAP is employed to evaluate the precision of object localization at varying IoU thresholds, which is crucial for measuring how well the model delineates and positions bounding boxes around detected objects. Notably, YOLOv8 does not include accuracy as a default metric.

Metrics for both systems are obtained from the validation set, ensuring they reflect performance on unseen data. The percentage increase from Pure and Augmented results will be calculated - using the same dataset sample - to compare relative improvements. Part of Hypothesis 1 in Section 4.3 will be evaluated using Validation Accuracy vs. Epoch graphs, generated with ClearML [35]. The gradient of the curve during the learning phase and convergence epoch will be compared for pure and augmented sets to determine which 'Learned Quicker'.

4.3 Hypotheses

H1: Data Augmentation will improve Image Classification F1 Scores, with some methods performing better than others. Data Augmentation will cause the Keras Image Classifier to learn quicker and converge earlier.

H2: Data Augmentation will have a greater impact on smaller datasets, achieving a greater increase in F1 Score and mAP between pure and augmented sets.

H3: Data Augmentation will have a greater impact on datasets with 'cloudy' and 'disruptive' weather (Section 5.3).

H4: Increasing the number of training data points will improve F1 scores more significantly than specific augmentation methods; when comparing equal-sized datasets, augmented data will not yield higher F1 scores than non-augmented data. Increasing the augmentation percentage will lead to a greater increase and F1 score.

H5: Augmentation Configurations obtained from the Image Classifier will improve YOLO Object Detection, but to a lesser extent.

5 Design

5.1 Prototyping

Prototyping made use of Jupyter Notebooks for their interactivity, instant feedback, and visualization. They allowed seamless switching between Python environments, avoiding compatibility issues. During prototyping, PyTorch was used to implement a Faster RCNN with a pre-trained ResNet50 model, requiring the conversion of YOLO-formatted datasets (AVOIDDS) to PyTorch-compatible ones. Due to infeasible training times and inconsistent performance, this concept was abandoned. While prototyping with YOLOv8, disabling training features and using an untrained model were considered, theorising that this would lead to a greater performance imrpovement from pure to augmented experiments. However, after consulting with my project supervisor, we decided to use a standard YOLOv8 implementation to ensure realistic performance results.

After the frameworks were decided, the prototyping phase involved testing dataset handling and augmentation functionality. It also involved determining key experiment details such as:

- Dataset sizes and augmentation percentages to use for the experiment (Section 4.2)
- The epoch count to use for experiments. 30 epochs were chosen for experiments to ensure consistent convergence across models with different dataset sizes and augmentation methods while keeping time costs reasonable.
- Model hyperparameters were tested to ensure consistent performance, deliberately allowing room for potentially better-performing future runs (optimal augmentation runs).

5.2 System Design

Multiple classes and modules have been designed to facilitate the specified experiments. These handle various functions including dataset creation & manipulation, augmentation, training, and evaluation. Figure 21 contains the class diagram for the final proposed design. The Augmenter module contains the individual methods for augmentation logic, each of these methods will perform the specified augmentation type to a single image (and label) and return the augmented versions. This module also contains the augment_image method which selects and applies the appropriate augmentation method based on the specified type in the augmentation metadata. The Dataset Module contains methods for all dataset handling within the system. Key functions include importing the AVOIDDS dataset into a Pandas DataFrame, creating dataset subsets from AVOIDDS, correcting AVOIDDS labels to represent multiple classes, and generating augmented datasets. Additionally, the module will support multithreading to enhance the speed of file operations.

The train_YOLO and train_CLF scripts will be used to execute the training and evaluation procedure for YOLO and the Keras Image Classifier respectively. Both scripts will create a subset of AVOIDDS, then train and evaluate using the

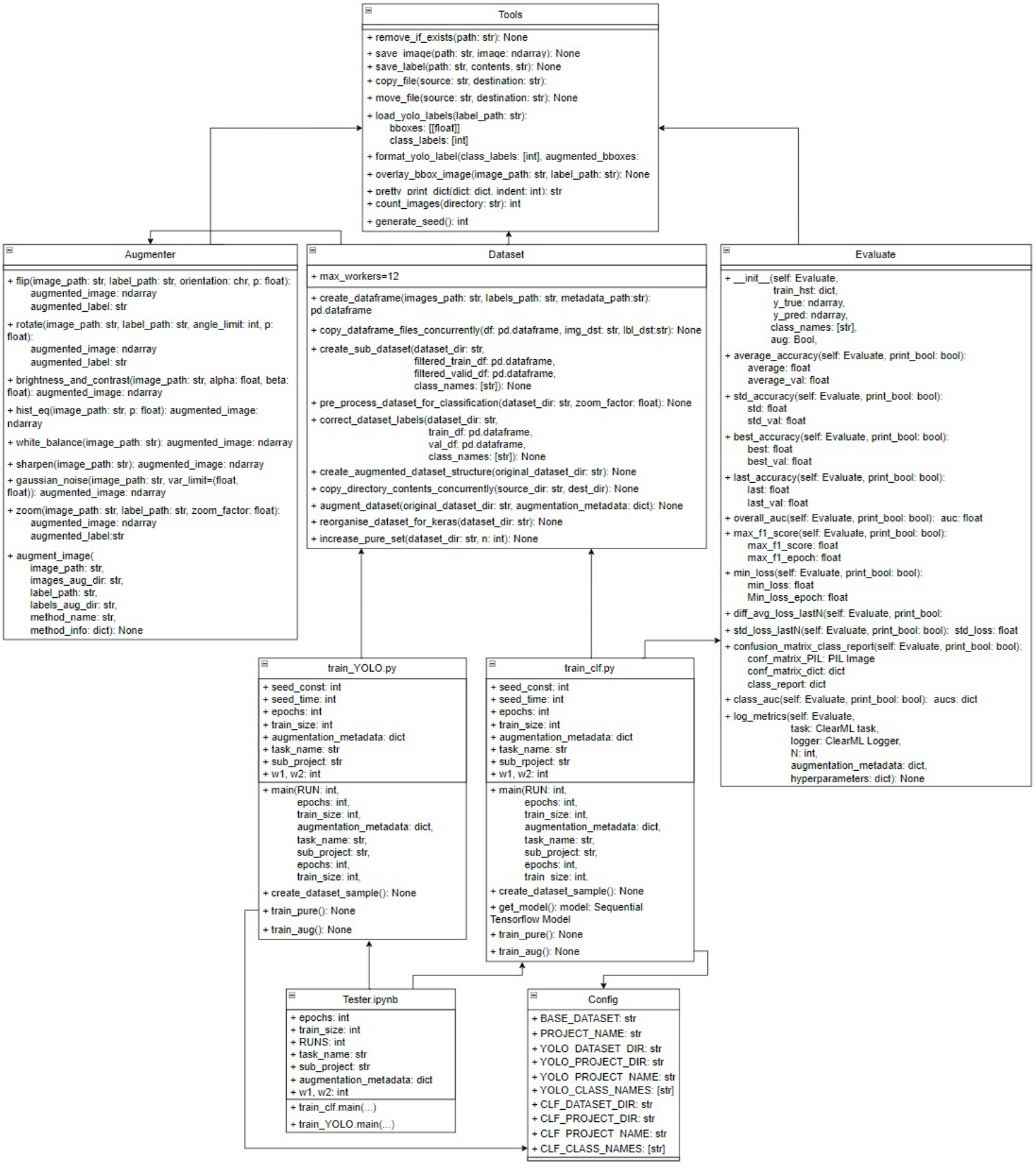

Figure 21: Class Diagram for Proposed Design

specified models for both pure and augmented sets. The YOLO system has built-in functionality to calculate performance metrics, however, since CLF has been custom-built for this system, the Evaluate class will be used to calculate various performance metrics for image classification. Metrics from both frameworks will be logged using ClearML. The Tester.ipynb notebook is used to execute runs of both YOLO and CLF. Each code block will contain unique tests with the specific metadata for that test.

5.3 Individual Experiment Design

The proposed structure of an individual experiment (for both YOLO and CLF) is detailed below:

- An experiment will be initialised with the specific parameters and metadata, including Augmentation Metadata, Number of Epochs, train dataset (pure) size, number of Bootstrap samples, and weather type.
- Each experiment will only sample from the 'Palo Alto' region of the AVOIDDS dataset, acting as a control variable. Due to AVOIDDS' size, enough images are available to sample dataset sizes of 500, 1000 & 2500 from the Palo Alto region (25% of the AVOIDDS dataset), even after stratification. Palo Alto was chosen because it represents a larger range of land features than the other locations, including coastlines, mountainous areas and urban landscapes.
- For all experiments, the 3 aircraft from AVOIDDS will be the classes: 'Cessna Skyhawk', 'Boeing 737-800', 'King Air C90'. Each dataset sample will be stratified to ensure equal class distribution.
- The dataset samples will be stratified by weather type, consolidating the 6 weather categories from AVOIDDS into 3 groups based on visual similarity for more efficient testing. These groups will be referred to as 'Clear' (combining 'Clear' & 'High Cirrus' - 01), 'Cloudy' (combining 'Scattered' & 'Broken' - 23), and 'Disruptive' (combining 'Overcast' & 'Stratus' - 45). Each experiment will focus on one of these groups, and the samples within the selected category will be further stratified to ensure an equal distribution between the two original weather types that constitute each new group. This approach simplifies the evaluation process while maintaining a balanced representation of weather conditions.
- Each dataset sample will be generated using a random key based on the current time in milliseconds and will be set at experiment initiation, ensuring keys are unique. A separate static key - constant across experiments - will be used for training to ensure consistent training and evaluation.
- Each experiment will create an augmented dataset using the augmentation metadata provided at initiation. This dataset will include the pure train set, determined by the train_size parameter; the augmented train set, created by applying the specified augmentations to the pure train set; and a validation set sampled from the 'Valid' directory of the AVOIDDS dataset. The validation set size will be 25% of the pure train set size, and the same validation set will be used to evaluate for both pure and augmented.

- Both YOLO and CLF will use the same 2 base models for every experiment. As mentioned previously, the same random seed will be used for both models for every experiment. The pure model will be trained before the augmented model, and the models used for each will be completely separate.

- The results of all experiments will be logged to ClearML, each Task within each experiment will have a unique identifier with the form:

  (*aug_method*)-(*weather*)-(*epochs*)-(*train_size*)-(*run_id*)-(*type*).

  For example: *he*-45-30-1000-3-*aug*. ClearML automatically generates plots and saves metrics as artefacts, facilitating objective comparison between experiments. This feature enables analysis to determine which models, pure or augmented, achieve faster learning, as detailed in Section 4.2.

- Each experiment will be run a total of 5 times per weather type, totalling 15 bootstrapped samples, the average and standard deviation of the metrics will be taken across these.

5.4 Test Strategy

Figure 22 outlines the proposed test strategy. The flowchart categorizes experiments by type using colour with red, orange, green, dark blue, and purple representing the Keras Image Classifier experiments, while light blue represents YOLO experiments. Each experiment will follow the individual experiment design defined in Section 5.3; as experiments progress, the number of augmentation methods tested should generally decrease, while the augmentation percentage will generally increase. This is because worst-performing methods will not proceed to future experiments. The goal of the red, orange, green and dark blue experiments is to evaluate all augmentation methods and varying percentages to propose the best augmentation configurations. The next stage will involve testing the proposed best configurations in 2 types of experiments: testing with equal-sized pure and augmented sets (purple) and testing with YOLO Object Detection (light blue).

For the equal pure and augmented experiments, the train dataset sample will initially contain 1000 images (the default dataset size). Each experiment will then augment this set by increasing augmentation percentages (using the proposed configurations) to create Augmented sets of sizes 1500 (50%), 2500 (150%) and 3500 (250%). Following this, the new unique images will be appended to the train set to match the size of the augmented sets. These new images will be unique and follow the same stratification parameters that were initially used to sample the train set. For YOLO experiments, the proposed best configurations will be tested with all 3 dataset sizes. The evaluation and success metrics are as stated in Section 4.3.

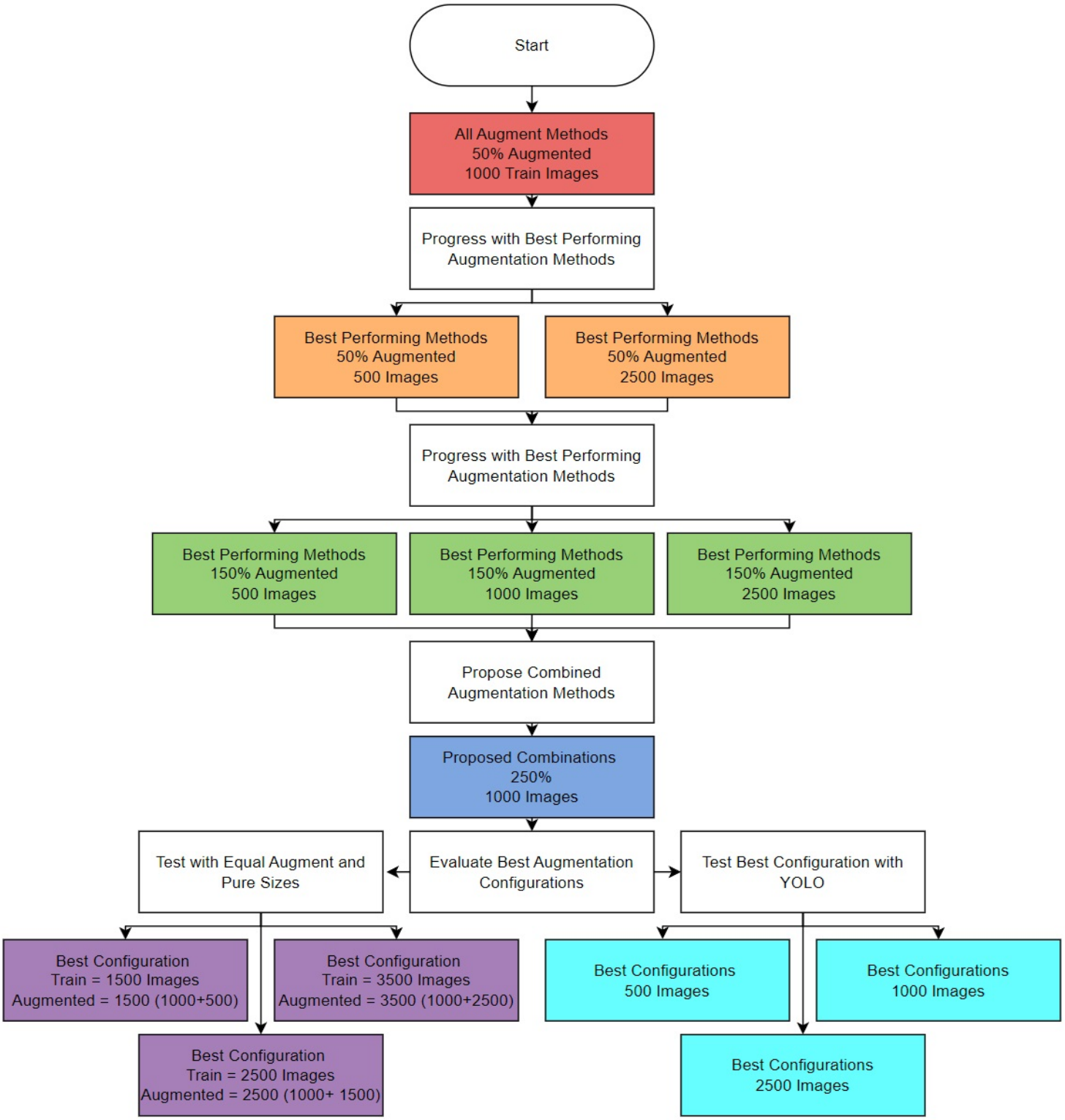

Figure 22: Flowchart for Proposed Test Strategy

6 Implementation

6.1 Documentation

Python was used for prototyping and the final implementation, as it is the primary language for TensorFlow and YOLOv8. All code was documented using pydocs, adhering to standard conventions. The HTML documentation is included in the project archive.

6.2 Testing Platform

Testing platforms for running TensorFlow experiments were evaluated, and ultimately, Windows CPU was chosen. TensorFlow on Windows provided faster CPU performance for image and dataset manipulation, though training took longer due to the lack of GPU support. On the other hand, TensorFlow with GPU on WSL had significantly hindered CPU performance, possibly due to using a mounted C drive (/mnt), and ClearML implementation was slow and buggy, taking 25 minutes to reach the training section with 500 images, compared to 1.25 minutes on Windows. YOLO leverages GPU capabilities through PyTorch, which is optimized for use on Windows. After resolving issues with CUDA installations, this setup performed effectively for this system.

The Lyceum Iridis Cluster was considered as an alternative platform [36], however, was not used in the final implementation. The "Iridis 5 Software Repository" [36] contains all installed packages on the Lyceum system; this repository does not contain TensorFlow/Keras or Ultralytics/YOLOv8 (or any previous version). Packages can be installed using the Spack Package manager [36], however, the installation of TensorFlow failed due to a conflict with the pre-installed GCC package, despite attempts with multiple compilers. The Ultralytics installation also failed because several required dependencies could not be installed. Debugging on Windows revealed significant challenges in establishing a reliable and consistent environment with these frameworks; issues included numerous CUDA errors, problems with relative paths, and compatibility issues with other packages, particularly within Conda or Linux environments. Therefore, it was decided not to pursue an implementation with Lyceum, as the indefinite time required to build the system could jeopardize the completion of all experiments.

6.3 Augmenter Module

This module contains methods which perform the specified augmentation techniques on single images and corresponding labels. The following methods, **flip**, **rotate**, **hist_eq** and **gaussian_noise**, use the corresponding Albumentations (A) functions described in Section 2.3.2. Each of these methods creates an augmentation pipeline using the **A.Compose** method which augments the image initially, before augmenting the corresponding label (if necessary) using the **A.bboxparams** function - specifying the 'YOLO' format to match the AVOIDDS dataset. This function expects and returns the bounding box coordinates for each label, therefore the methods **Tools.load_yolo_labels** and **Tools.format_yolo_labels** were created. These helper functions convert string labels into the appropriate data types ([[**float**]] for coordinates and [**int**] for class labels) and then convert the augmented bounding box coordinates back into a

string format with the corresponding class labels.

The **zoom** method (Figure **??**) zooms into the aircraft in the image using the **A.Crop** and **A.Resize** to 'crop' a portion of the image before resizing it back to the original image size of 1920x1080. To calculate the crop dimensions, the method first converts the bounding box coordinates from relative to absolute values based on the image size. Using the bounding box's centre coordinates and dimensions, it determines the crop dimensions by dividing the bounding box size by the zoom factor. The crop coordinates are then calculated to ensure they remain within the image boundaries, ensuring the aircraft is centred and properly framed. This approach avoids the risk of losing the aircraft from the image, as might happen with random or centre crops.

The **brightness_and_contrast**, **white_balance** and **sharpen** methods do not use Albumentations. The **brightness_and_contrast** method uses the equation stated in Section 2.3.2 to adjust the brightness or contrast of an image; the alpha value controls contrast while the beta value controls brightness. The function parameters define the limits for these variables, and a random value between the limit and the baseline (no adjustment) is calculated to apply the adjustment. The **white_balance** method uses the gray world algorithm to adjust the image's colour balance. It calculates the mean value for each colour channel (red, green, and blue) and then computes the overall mean ('mgray'). Each channel is adjusted based on the gray world assumption that the "average of all channels should result in a gray image" [23]. This is achieved by scaling each channel's values proportionally to the ratio of the overall mean to the channel means. The **sharpen** function sharpens an image using the **filter2D** method from OpenCV, as stated in Section 2.3.2.

The module also includes the **augment_image** method, which calls the appropriate augmentation method with given parameters and saves the resulting augmented images and labels.

6.4 Augmentation Metadata

The augmentation metadata is a dictionary variable which is set by the user for each test. It contains the methods to be used, their parameters and their augmentation percentage, figure **??** shows an example.

6.5 Dataset Manipulation

The dataset module contains all dataset-related code, including functions that are used to arrange dataset manipulation pipelines which create and prepare dataset samples for training. Where possible, this module implements multithreading using Pythons **ThreadPoolExecutor** class, to increase the speed of operations such as copying and transferring files.

The module imports the AVOIDDS dataset into 2 pandas DataFrames (DF) (train & validation) using the **create_dataframe** method, storing paths to images and labels rather than the files themselves. It uses the *metadata.json* file from AVOIDDS to include information about each image, such as weather conditions and aircraft type. These DFs are filtered to include only images from

the Palo Alto region, and then further filtered based on the weather category specified for an individual experiment. Following this, a 'stratify_key' column is added to the filtered DFs by concatenating the aircraft and weather columns. The **train_test_split** function from Scikit-Learn is then used to create the final training and validation DFs from the filtered ones. The DFs are stratified using the 'stratify_key' column, and a random seed based on the current time in milliseconds is used to generate the samples, as described in Section 5.3. The **create_sub_dataset** method takes the train and validation DFs and creates the dataset directory, concurrently copying the correct files from AVOIDDS.

The **augment_dataset** function takes augmentation metadata as a parameter and augments an entire dataset sample. It reconstructs the dataset's directory (Figure 23), adding an empty 'train-aug' directory alongside the train and validation ones for images and labels. Additionally, it creates a new YAML file that designates this new augmented set as the YOLO training set. The method then copies all images and labels from the train set to the train-aug set before applying the augmentations. For each augmentation method specified in the metadata, the code calculates the number of images to augment based on a given percentage. It randomly selects the specified number of images, allowing for repetition as percentages may surpass 100%, and determines the corresponding label path for each selected image. These augmentation tasks are then submitted to another **ThreadPoolExectuor** instance and handled concurrently, with the executor applying the specified augmentation method to each image.

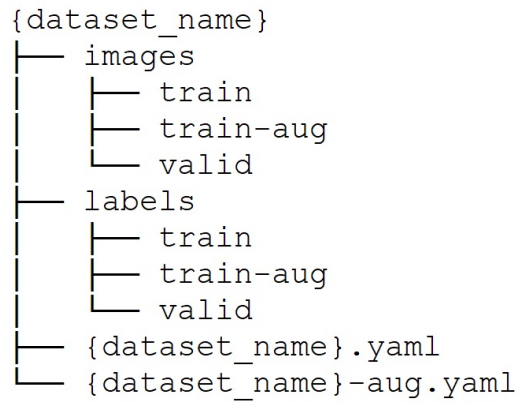

Figure 23: Dataset Directory Structure for Augmented Dataset

Other crucial functions in the Dataset Module are:

- **correct_dataset_labels**. This method adjusts the labels in dataset samples to represent the classes as integers 0,1,2, corresponding to the three aircraft in AVOIDDS. The original labels only had one class - aircraft - since every image contains an aircraft, this was unsuitable for image classification.
- **reorganize_dataset_for_keras**. This method organizes images within train, train-aug and valid to match the Keras Dataset structure described in Section 2.4.1.

- **pre_process_dataset_for_classification**. This method applies a minor zoom augmentation to every image in a dataset sample, including train and train-aug (on top of any augmentations). This was necessary as the classifier initially performed poorly due to aircraft appearing very small in some images. A zoom factor of 1.5 was chosen and used for every experiment, improving the performance of the classifier to a consistent and reliable state. The YOLO experiments will not use this method to ensure the system is tested on realistic ownship POVs. This will be considered when evaluating the performance of the YOLO system.

- **append_new_train_images**. This method appends new unique images to the train directory to match the size of train-aug. These images follow the same stratification rules as the original sample as described in Section 5.4. This method also corrected the labels of the new images.

6.6 Keras Image Classifier Implementation

All Keras CLF experiments will follow this dataset manipulation pipeline to create dataset samples (ds refers to the Dataset Module):

1. **ds.create_sub_dataset**
2. **ds.correct_dataset_labels**
3. **ds.augment_dataset**
4. **(ds.append_new_train_images)** - only used in experiments with equal train and train-aug sizes
5. **ds.pre_process_dataset_for_classification**
6. **ds.reorganize_dataset_for_keras**

Figure 24 outlines the Keras CNN model architecture inspired by the model described in Section 2.4.1. Initially, the model applies a rescaling layer to normalize pixel values of input images sized at 256x256 pixels with 3 color channels. This size was chosen to balance computational efficiency with the retention of sufficient image detail, representing roughly a quarter of the original image's height. The model includes sequences of Conv2D and MaxPooling2D layers, which reduce spatial dimensions and increase feature depth. Specifically, it features two convolutional layers with 16 and 32 filters each followed by max pooling that halves the feature map dimensions, and another convolutional layer with 64 filters followed by max pooling to further reduce the dimensions and enhance feature depth. To prevent overfitting, a dropout layer randomly sets input elements to zero during training. The flattened output feeds into a dense layer with 128 units and a final dense layer with 3 units, corresponding to each class. An 'Adam' optimizer function was used with a 'categorical_crossentropy' loss function because 'Adam' efficiently handles sparse gradients and adapts well to different problems, while 'categorical_crossentropy' is optimal for multi-class classification, ensuring effective model convergence and accuracy.

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 256, 256, 16)	448
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout (Dropout)	(None, 32, 32, 64)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 128)	8,388,736
dense_1 (Dense)	(None, 3)	387

Figure 24: Model Architecture for Keras Image Classifier

6.7 YOLO Object Detection Implementation

All YOLO experiments will follow this dataset manipulation pipeline:

1. **ds.create_sub_dataset**
2. **ds.correct_dataset_labels**
3. **ds.augment_dataset**

The pre-trained **YOLOv8n** model was used for training in the YOLO experiments, due to it being the fastest model available with YOLOv8 (Figure 20). A summary of the model is shown in Figure 25. YOLO performance metrics are automatically calculated by the system, however, the macro F1 score was calculated manually using the F1 scores for each class.

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
16	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	123648	ultralytics.nn.modules.block.C2f	[192, 128, 1]
19	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
22	[15, 18, 21]	1	751897	ultralytics.nn.modules.head.Detect	[3, [64, 128, 256]]

Model summary: 225 layers, 3011433 parameters, 3011417 gradients, 8.2 GFLOPs

Figure 25: Model Architecture for YOLOv8n

7 Results

This Section contains the experiment results, all results are stored in the Excel Spreadsheets listed in the file archive.

7.1 All Augmentation Methods

Table 2 presents the percentage increase from pure to augmented datasets for both the F1 score and the 'Learned Quicker' percentage (percentage of runs where augmented learnt quicker than pure for each method).

Method	**% Increase F1 Score**	**Learned Quicker (%)**
Brightness(-60)	10.75	86.67
Brightness(30)	-5.47	53.33
Brightness(90)	-46.35	20.00
Contrast(0.5)	49.72	60.00
Contrast(1.25)	-0.81	46.67
Contrast(2.0)	-41.88	13.33
Gaussian(100,200)	48.07	80.00
Gaussian(10,50)	40.74	66.67
Gaussian(1,5)	34.56	66.67
HFlip	33.60	66.67
HistEq	57.51	60.00
Rotate(20)	62.22	86.67
Rotate(45)	20.31	40.00
Rotate(5)	113.69	93.33
Sharpen	7.21	86.67
VFlip	-8.00	40.00
WhiteBal	-13.27	33.33
Zoom(0.4)	50.64	80.00
Zoom(1.2)	6.96	40.00
Zoom(2.0)	-9.33	33.33

Table 2: Performance Metrics using Image Classifier against all Augmentation Methods, Dataset Size: 1000, Augmentation Percentage: 50%

The F1 score's percentage increase varied significantly, ranging from a decrease of 46.4% with Brightness(90) to an increase of 113.7% with Rotate(5), the most effective methods are highlighted in Table 3. Conversely, the least effective methods were Brightness(90) and Contrast(2.0), which saw decreases of 46.4% and 41.88% respectively. The Rotate(5) method achieved an average F1 score of 0.53 with pure datasets and 0.80 with augmented datasets, with standard deviations of 0.281 and 0.013 respectively. For methods with multiple parameters, the parameters causing the least extreme change to the image generally achieved the highest increase in the F1 score. Specifically, Contrast with an alpha value of 0.5 significantly outperformed alpha values of 1.25 and 2.0. Similarly, Rotate with angle limits of 5 and 20 outperformed Rotate with an angle limit of 45. Minor Zooms with a zoom factor of 0.4 also outperformed greater zoom values. All Gaussian parameters performed similarly, indicating that Gaussian noise does not significantly impact the image from the model's perspective. Additionally,

negative brightness adjustments performed better than positive brightness adjustments, though neither significantly improved performance.

For 12 out of 20 methods, runs with augmented datasets learned quicker more than 50% of the time. The highest percentages were achieved by Rotate(5) (93%) and Rotate(20) (87%), indicating a correlation between learning rate and F1 score increase. However, Sharpen and Brightness(-60) also achieved 87% but had much lower increases in F1 score. Brightness(90) and Contrast(2.0), which had the worst percentage increases in F1 score, also had the lowest "learned quicker" percentages (20% and 13%), reinforcing the correlation.

Method	**% Increase F1 Score**	**Learned Quicker (%)**
Contrast(0.5)	49.72	60.00
Gaussian(100,200)	48.07	80.00
HistEq	57.51	60.00
Rotate(20)	62.22	86.67
Rotate(5)	113.69	93.33
Zoom(0.4)	50.64	80.00

Table 3: Best Performing Augmentation Methods, Dataset Size: 1000, Augmentation Percentage: 50%

Figure 26 shows the percentage increase in F1 score for each of the best-performing methods, split into the three weather conditions. 100% of these methods saw the greatest increase with Cloudy (23) or Disruptive (45) weather. The percentage increase for Cloudy with Rotate(5) is more than twice the next highest, this outlier was considered when evaluating methods and choosing configurations. For all methods, the worst-performing methods yielded the greatest percentage decrease with Cloudy and Disruptive weather conditions (Figure **??**).

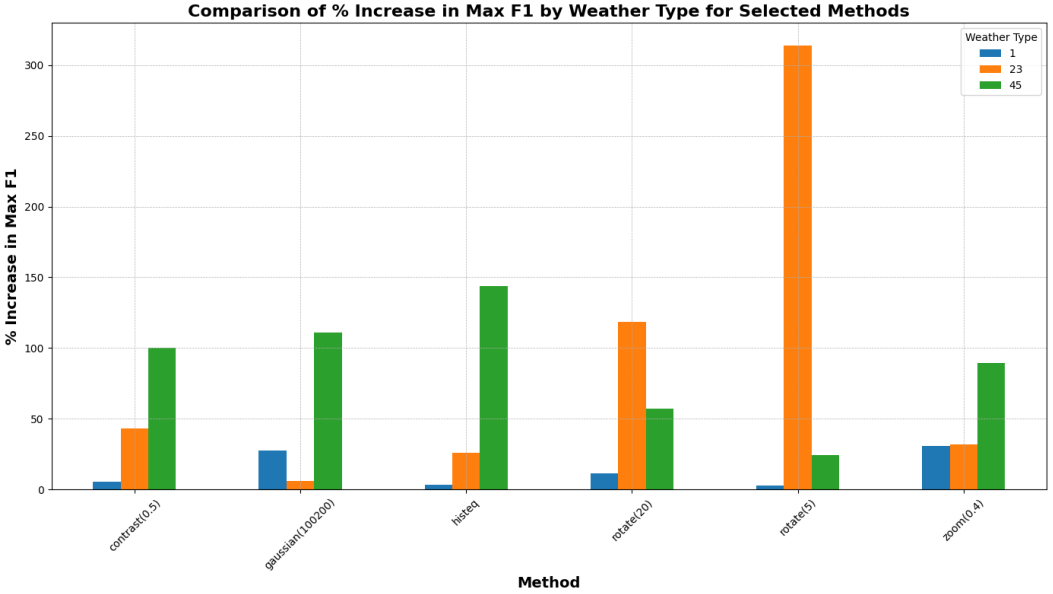

Figure 26: Percentage Increase in F1 Score for Best Performing Methods with Weather Conditions

7.2 Lower and Upper Bound Dataset Size

Tables 4 and 5 present the results of the best-performing methods with 2500 and 500 images, respectively, at 50% augmentation. Similar to the experiment with 1000 images, Rotate(5) achieved the greatest increase in F1 score in both experiments, with increases of 50.0% and 54.1%, respectively.

Method	**% Increase F1 Score**	**Learned Quicker (%)**
Contrast(0.5)	14.26	53.33
Gaussian(100,200)	30.35	60.00
HistEq	2.45	60.00
Rotate(20)	20.02	66.67
Rotate(5)	50.07	93.33
Zoom(0.4)	39.12	60.00

Table 4: Best Performing Augmentation Methods with Dataset Size: 2500, Augmentation Percentage: 50%

Method	**% Increase F1 Score**	**Learned Quicker (%)**
Contrast(0.5)	20.12	64.29
Gaussian(100,200)	5.99	60.00
Gaussian(10,50)	39.85	86.67
HistEq	29.26	60.00
Rotate(20)	14.14	80.00
Rotate(5)	54.11	86.67
Zoom(0.4)	41.81	66.67

Table 5: Best Performing Augmentation Methods with Dataset Size: 500, Augmentation Percentage: 50%

Figures 27 and 28 show the Average Percentage Increase of F1 Score and Average Learned Quicker Percentage (across best-performing methods) against the 3 dataset sizes. 1000 images yielded the greatest increase in F1 score, with 2500 images achieving the smallest increase.

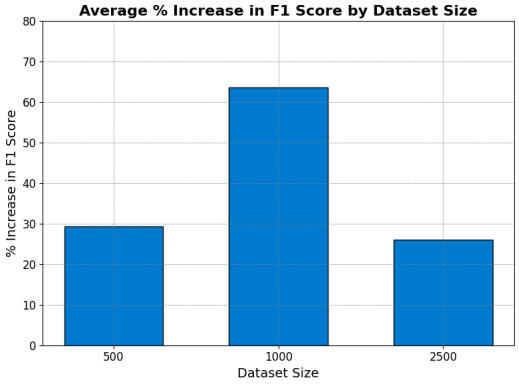

Figure 27: Total Average Percentage Increase in F1 Score at 50% Augmentation for all Dataset Sizes

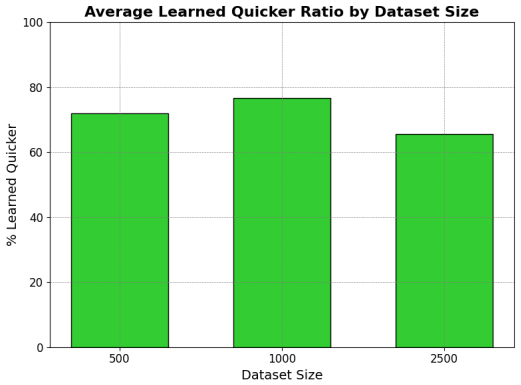

Figure 28: Total Average Learnt Quicker at 50% Augmentation for all Dataset Sizes

All dataset sizes achieved a Learned Quicker Percentage around 70%, with the 1000-image dataset demonstrating the highest percentage and the 2500-image dataset the lowest. For 500 images, 66.7% of the best-performing methods achieved a greater increase in F1 score under Cloudy and Disruptive conditions. For 2500 images, this figure was 83.3%.

7.3 Increased Augmentation Percentage

Tables 7, 8 and 6 display results for all dataset sizes with a 150% augmentation percentage. The zoom method was replaced by Gaussian(10,50) due to its inability, like HFlip and VFlip, to create multiple unique augmentations per image, rendering it ineffective for augmentation rates above 100%. Again, Rotate(5) performed achieved the largest increase in F1 score for all 3 dataset sizes. Both configurations of Gaussian performed the worst for all sizes. Therefore, only rotates, histogram equalisation and contrast will be used in subsequent experiments

Method	**% Increase F1 Score**	**Learned Quicker (%)**
Contrast(0.5)	19.46	66.67
Gaussian(100,200)	8.18	100.00
Gaussian(10,50)	1.56	66.67
HistEq	21.29	100.00
Rotate(20)	28.54	100.00
Rotate(5)	34.19	100.00

Table 6: Best Performing Augmentation Methods with Dataset Size: 2500, Augmentation Percentage: 150%

Method	% Increase F1 Score	Learned Quicker (%)
Contrast(0.5)	36.71	100.00
Gaussian(100,200)	18.97	73.33
Gaussian(10,50)	7.89	93.33
HistEq	73.48	100.00
Rotate(20)	40.23	100.00
Rotate(5)	117.14	100.00

Table 7: Best Performing Augmentation Methods with Dataset Size: 500, Augmentation Percentage: 150%

Method	% Increase F1 Score	Learned Quicker (%)
Contrast(0.5)	68.38	71.43
Gaussian(100,200)	27.27	73.33
Gaussian(10,50)	-0.37	60.00
HistEq	52.50	100.00
Rotate(20)	55.03	66.67
Rotate(5)	98.94	100.00

Table 8: Best Performing Augmentation Methods with Dataset Size: 1000, Augmentation Percentage: 150%

Figures 29 and 30 illustrate the average F1 increase and Learned Quicker Percentage for the four best-performing methods: Rotate(5), Rotate(20), HistEq, and Contrast(0.5). Increasing the augmentation percentage significantly improved the F1 score for the 500-image dataset and slightly improved it for the 2500-image dataset. However, it performed similarly for the 1000-image dataset, possibly due to an outlier in the 50% augmentation experiment with Rotate(5). Increasing the augmentation percentage consistently improved the Learned Quicker Percentage, with all dataset sizes performing similarly, ranging from 85-95%.

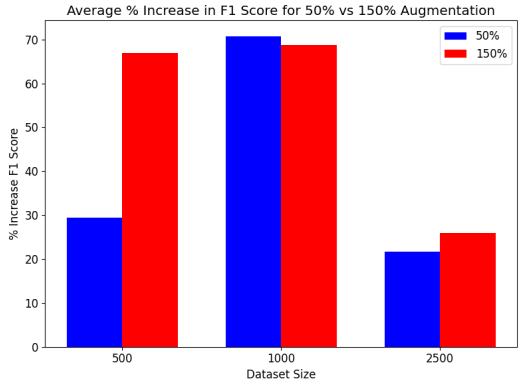

Figure 29: Total Average Percentage Increase in F1 Score at 50% vs 150% Augmentation for all Dataset Sizes.

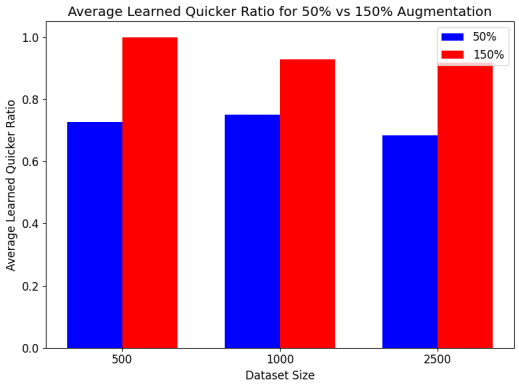

Figure 30: Total Average Learnt Quicker at 50% vs 150% Augmentation for all Dataset Sizes.

Across the 3 150% experiments, 83% of runs yielder a greater F1 increase with Cloudy or Disruptive weather conditions.

7.4 Combining Augmentation Methods

The proposed combinations are listed below, they contain arrangements of the best performing from the previous round of experiments:

- **HistEq, Contrast(0.5), Rotate(5), Rotate(20)**
- **HistEq, Contrast(0.5)**
- **Rotate(5), Rotate(20)**
- **Rotate(5)**

These combinations were chosen to contain both affine image transformations and colour modifications, as well as a combination of both. Rotate(5) was also used independently as a control. Tables 9 and 10 present the results for 1000 and 500 images. Unfortunately, due to memory limitations of the Desktop PC used, the test with 2500 image datasets did not complete, suggesting a need for external computation.

Method	**% Increase F1 Score**	**Learned Quicker (%)**
HistEq, Contrast(0.5), Rotate(5), Rotate(20)	409.74	93.33
HistEq, Contrast(0.5)	76.96	80.00
Rotate(5), Rotate(20)	39.46	100.00
Rotate(5)	45.51	93.33

Table 9: Combination Best Performing Augmentation Methods with Dataset Size: 1000, Total Augmentation Percentage: 250%

Method	% Increase F1 Score	Learned Quicker (%)
HistEq, Contrast(0.5), Rotate(5), Rotate(20)	15.57	100.00
HistEq, Contrast(0.5)	20.14	80.00
Rotate(5), Rotate(20)	37.37	100.00
Rotate(5)	33.24	100.00

Table 10: Combination of Best Performing Augmentation Methods with Dataset Size: 500, Total Augmentation Percentage: 250%

The results show consistent improvement in F1 Score, but no single method consistently outperformed others. Combining methods with different parameters, like Rotate(5) and Rotate(20), did not consistently improve performance compared to using Rotate(5) alone. Figures 31 and 32 illustrate the performance of Rotate(5) across three augmentation percentages. The percentage increase at 250% was notably less than at 150%. 150% achieved a significantly greater increase in F1 scores for 500 images, however, 50% narrowly outperformed 150% for 1000 Images.

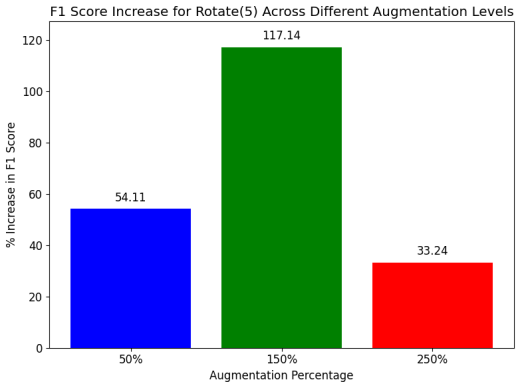

Figure 31: Rotate(5) Percentage Increase in F1 Score at 50%, 150% & 250% Augmentation for 500 Image Datasets.

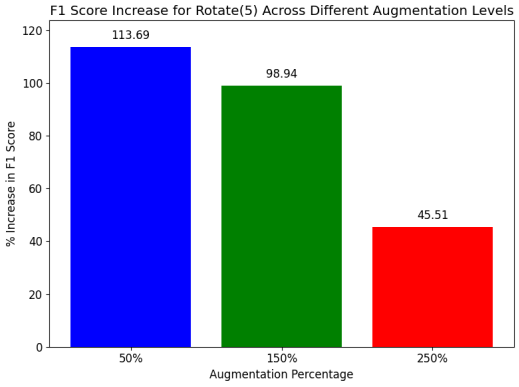

Figure 32: Rotate(5) Percentage Increase in F1 Score at 50%, 150% & 250% Augmentation for 1000 Image Datasets.

Furthermore, 87.5% of experiments at 250% augmentation yielded the greatest

increase in F1 score at Cloudy or Disruptive weather conditions.

7.5 Equal Pure and Augmented Train Sizes

This set of experiments, detailed in Section 5.3, utilized different augmentation percentages for Rotate(5) due to its consistent superior performance in previous tests. Table 11 shows that contrary to H4, there were notable improvements in F1 Score. 150% augmented yielded the greatest increase of 43.9%, whereas, 250% yielded the lowest with 8.1%. The Learned Quicker percentage dropped significantly in this experiment, averaging 66.7% across the three augmentation percentages. This is a notable decrease from the 97.8% average observed in previous experiments with Rotate(5) using 1000 images and the same augmentation percentages

Augmentation Percentage	**% Increase F1 Score**	**Learned Quicker (%)**
50%	30.52	73.33
150%	43.85	66.67
250%	8.13	60.00

Table 11: Experiments with Pure and Augmented Sets of Equal Size, testing with Rotate(5) at varying augmentation percentages.

Figure 33 indicates that under clear weather conditions, the augmented set consistently under-performed. However, notable improvements in F1 score were achieved with cloudy and disruptive weather conditions.

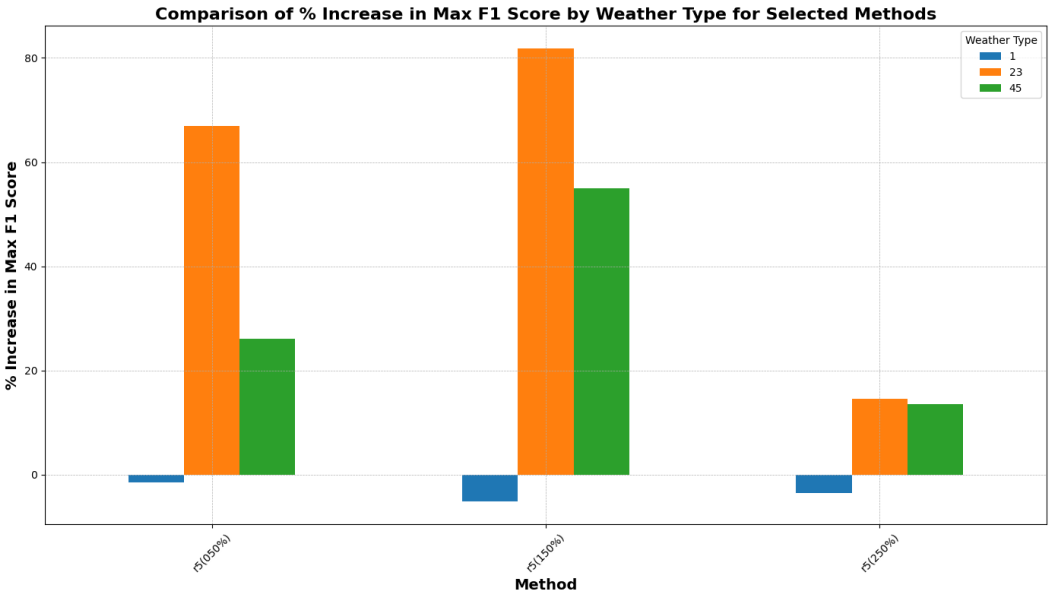

Figure 33: Rotate(5) Percentage Increase in F1 Score at 50%, 150% & 250% Augmentation for 500 Image Datasets.

7.6 YOLO

Proposed Configurations for YOLO:

- HistEq, Contrast(0.5) and Rotate(0.5)
- HistEq, Contrast(0.5)
- Rotate(5)

These methods represent colour modifications, image transformations and a combination of both. Rotate(5) has consistently been the best-performing method, this was chosen over the combination of Rotate(5) and Rotate(20) which saw no notable performance increase. For dataset sizes and augmentation percentages: 150% was used for 500 images, as it caused the largest increase in F1 Score. For 1000 images, 150% was chosen over 50% due to its improved performance in the equal pure and augmented tests (after displaying similar performance in the previous experiments). 50% was chosen for 2500 Images as 150% showed minimal improvement in exchange for the longer training time, risking missing the experiment deadline set in Section 9.1.

Results for the YOLO experiments are shown in Tables 12, 13 and 14. The results indicate consistent performance across methods, though the percentage increase in F1 score was notably lower than for the image classifier. Rotate(5) achieved the highest F1 scores with both 2500 and 500 images, while the combination of all methods yielded the best F1 score with 1000 images. Both of these configurations included Rotate(5), suggesting that Rotate(5) is a crucial component for achieving higher F1 scores.

Method	**% Increase F1 Score**	**% Increase mAP50**	**% Increase mAP50-95**
HistEq, Contrast(0.5), Rotate(5)	16.19	15.18	23.41
HistEq, Contrast(0.5)	16.79	16.31	20.14
Rotate(5)	18.32	16.03	22.17

Table 12: YOLO Experiments with Proposed Augmentation Combinations, Dataset Size: 500, Augmentation Percentage: 150%

Method	**% Increase F1 Score**	**% Increase mAP50**	**% Increase mAP50-95**
HistEq, Contrast(0.5), Rotate(5)	9.97	8.16	14.64
HistEq, Contrast(0.5)	6.37	5.03	12.44
Rotate(5)	8.16	6.63	15.20

Table 13: YOLO Experiments with Proposed Augmentation Combinations, Dataset Size: 1000, Augmentation Percentage: 150%

Method	% Increase F1 Score	% Increase mAP50	% Increase mAP50-95
HistEq, Contrast(0.5), Rotate(5)	1.97	2.00	4.60
HistEq, Contrast(0.5)	1.64	1.46	4.21
Rotate(5)	2.73	2.48	5.57

Table 14: YOLO Experiments with Proposed Augmentation Combinations, Dataset Size: 2500, Augmentation Percentage: 50%

Augmentation had the most significant impact on the 500-image dataset, followed by the 1000-image dataset, and then the 2500-image dataset, supporting hypothesis H2. The largest percentage increase was with mAP50-95 for all methods, suggesting that augmentation improved the model's ability to precisely locate and classify objects, maintaining high accuracy even at very strict IOU thresholds.
Figure 34 illustrates that the Boeing achieved the highest average F1 Score with (0.863 with Augmented), followed by the King Air (0.787) then the Cessna (0.639). The graph highlights the consistent yet moderate improvements in F1 Score that data augmentation caused for 2500 Images.

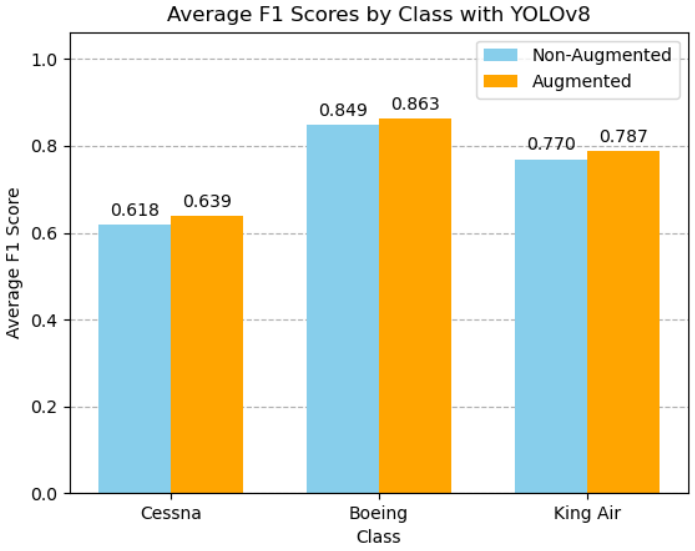

Figure 34: Average F1 Score per class with 2500 Images and YOLO

There was no correlation between weather conditions and performance increase mAP with YOLO. Figures **??** to **??** indicate that clear conditions lead to the most notable improvement in F1 scores, although the improvement is not significant.

8 Evaluation

8.1 H1

The first hypothesis suggested that Augmentation will improve F1 scores achieved with Image Classification, but individual methods will perform differently. This is true, with Augmentation consistently improving F1 scores for all Classification experiments. Rotate(5) was consistently the best individual method, followed by Rotate(20), Histogram Equalisation, Contrast(0.5) and all Gaussian Noise methods. Methods with more extreme parameters such as Brightness(90) or Contrast(2.0) performed significantly worse than other methods, indicating that aggressive transformations can overly distort the images, thereby reducing model accuracy. Methods with minor modifications yielded greater performance.

The hypothesis also states that Data Augmentation will cause the classifier to learn quicker, 12 out of the 20 methods learnt quicker over 50% of the time, suggesting this to be true. A one-sample t-test compared the null hypothesis: Learned Quicker (%)' = 50%, against the alternative that it is greater than 50%. With a p-value of 0.040 at the 5% significance level, the null hypothesis is rejected, indicating that data augmentation effectively enhances the model's learning speed. Subsequent experiments utilized only the best-performing augmentation methods, all achieving a 'Learned Quicker' percentage above 50%. The consistent results in these follow-up experiments further validate that data augmentation reliably improves learning speed.

An equivalent statistical test was used for the F1 score increase for all 20 methods from Table 2. The null hypothesis that data augmentation does not improve F1 Scores was rejected at a 0.05 significance level ($p = 0.027$), indicating a significant positive effect of augmentation on F1 scores.

8.2 H2

This hypothesis suggests that data augmentation will have a greater impact on smaller dataset sizes, with the improvements diminishing as the dataset size increases. Image classification results did not follow this pattern exactly, with 1000 achieving greater increases in F1 Score than 500 images. However, 2500 images consistently achieved the lowest improvement. The unexpected results with 500 images suggest that this dataset size was too small, lacking enough data to learn class-specific details, leading to overfitting with augmented data.

In contrast, results from YOLO aligned H2 with average F1 score increases of 17.1%, 8.17% and 2.11% for 500, 1000 and 2500 images respectively, with mAP values also following this trend. The use of a pre-trained model optimized for diverse data could be the reason for this.

8.3 H3

H3 proposes that Data Augmentation would lead to a greater improvement in Cloudy and Disruptive weather conditions; the image classification results align with this. At all dataset sizes and augmentation percentages, the majority of experiments yielded the greatest percentage increase with Cloudy and Disrup-

tive Conditions, which may introduce more background complexity, allowing data augmentation to enhance the classifier's ability to generalize. Another one-sample t-test was conducted to assess the proportion of performance improvements under varying weather conditions (using the values in Table 15), with a null hypothesis set at 66.7%. The resulting p-value of 0.019 led to the rejection of the null hypothesis at the 0.05 significance level, thus supporting H3.

Method	**Cloudy or Disruptive**
Best Performing Methods, 1000 Images, 50%	100%
Best Performing Methods, 1000 Images, 150%	100%
Best Performing Methods, 500 Images, 50%	71.4%
Best Performing Methods, 500 Images, 150%	83.3%
Best Performing Methods, 2500 Images, 50%	66.7%
Best Performing Methods, 2500 Images, 150%	66.7%
Combination of Best Performing, 500 Images, 250%	75%
Combination of Best Performing, 1000 Images, 250%	100%

Table 15: Percentage of Runs to yield the greatest improvement in Cloudy or Disruptive weather conditions.

There was no significant correlation between weather conditions and YOLO's performance gain (for both F1 scores and mAP), suggesting that object detection relies less on background information or that YOLO's pre-trained model is more robust to background conditions.

8.4 H4

H4 proposed that increasing the augmentation percentage would lead to greater performance improvement. Image classification results provide support for this; increasing the augmentation percentage from 50% to 150% resulted in substantial improvements for 500 images and minor improvements for 2500 images. However, with 1000 images, the 50% augmentation slightly outperformed the 150% augmentation, contradicting H4. This inconsistency might be due to the unusually high improvement seen with Rotate(5), or it could suggest that there is an optimal augmentation level for different dataset sizes that was exceeded at 150%.

H4 also suggests that the volume of training data points will have a greater impact on performance improvements than the specific methods used, implying minimal to no performance improvement with pure and augmented datasets of equal size. However, the results contradict this. All three augmentation percentages led to F1 score improvements with the augmented sets, with 150% showing the highest performance gains and 250% the lowest. Improvements were only observed under Cloudy and Disruptive weather conditions (Figure 33), with no improvements in Clear conditions. This further supports H3, indicating that data augmentation significantly enhances classification performance on noisy or more complex images.

8.5 H5

The hypothesis posited that augmentations derived from image classification would enhance YOLO object detection, increasing F1 score and mAP. The results supported H5 as all configurations showed performance gains across every metric for all dataset sizes. However, the increases in F1 score were less pronounced with YOLO, as the hypothesis proposed. Rotate(5) achieved an average percentage increase in F1 score of 8.16% with YOLO, compared to 98.9% with image classification (1000 images, 150%). This pattern held consistent across datasets of 500 and 2500 images, as detailed in the corresponding results tables. The reduced improvement in object detection could be due to the omission of the image pre-processing described in Section 6.5, which was beneficial in the classification experiments. Additionally, using a pre-trained YOLO model may have diminished the impact of augmentation, as the model was already optimized on the COCO dataset [32]. Image Classification and Object Description are ultimately different tasks; the latter's requirement for object localization might be less responsive to augmentations, which could explain the observed reduction in performance.

All configurations consistently yielded performance increases in both mAP50 and mAP50-95, demonstrating that data augmentation methods which enhance image classification also improve object localization within an image. The mAP50-95 metric showed the greatest percentage increase in performance among the YOLO metrics, suggesting that data augmentation not only enhances overall detection but improves precise localisation at tighter IoU thresholds.

In the YOLO experiments, the mAP50 metric with 2500 images exhibited the least improvement. A paired t-test was conducted on the corresponding raw pure and augmented mAP values from the Excel Result Sheets in the project archive. The test produced a very small p-value of 2.93×10^{-5}, confirming statistical significance for mAP50 with 2500 images. Thus, implying the statistical significance of for all metrics across the YOLO configurations.

8.6 Comparison to Literature

E. Q. Smyers et Al. trained the entire AVOIDDS dataset using the YOLOv8s model and evaluated its performance with mAP [1], however the specific IoU threshold was not specified. Their results generally surpassed those of this project, attributable to the more complex model (Figure 20) and larger dataset. They achieved a total mAP of 0.866 compared to 0.575 (mAP50-95) for this project (2500 images, Augmented). Figure 35 illustrates that in both projects, the 'Boeing 737-800' recorded the highest mAP, while the 'Cessna Skyhawk' had the lowest. The mAP scores appear to be influenced by aircraft size, with larger aircraft like the Boeing outperforming smaller ones such as the 'King Air C90' and the Cessna, demonstrating that object detection more effectively localize larger objects that occupy more of the image.

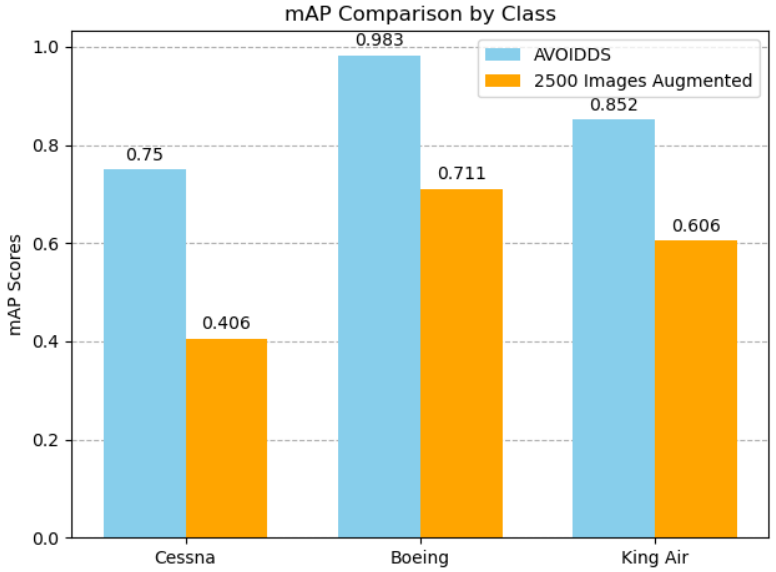

Figure 35: mAP Scores per Class. AVOIDDS vs 2500 Image Augmented Experiment

M. S. Alam et al. found that 250% augmented images yielded the best performance [16], this contradicts the findings of this project which suggest 50% and 150% are both superior augmentation percentages. They also suggested that standard affine transformations provide limited performance improvement [16], further contradicting this project, where Rotate(5) consistently yielded the greatest increase in F1 Score. Y. Mi, et al. found that augmenting using Gaussian Noise mitigated the risk of overfitting in their experiments [17], a result that aligns with this project's observations of consistent improvements in F1 Score with Gaussian Noise.

8.7 Training Time

Experiment	Training Time (Hours)
All Methods, 1000 Images, 50%	65
Best Performing Methods, 2500 Images, 50%	54
Best Performing Methods, 500 Images, 50%	13.5
Best Performing Methods, 2500 Images, 150%	67.5
Best Performing Methods, 500 Images, 150%	16.4
Best Performing Methods, 1000 Images, 150%	24
Combined Methods, 1000 Images, 250%	20
Combined Methods, 500 Images, 250%	11
Equal Pure and Augmented	15.9
YOLO, 500 Images	13.8
YOLO, 1000 Images	22.8
YOLO, 2500 Images	34.5
Total	**358.4 (14.9 Days)**

Table 16: Experiment Training Times

Table 16 highlights the potential need for robust computational clusters like Lyceum, given the extensive training times. Training with more images, such as 2500 at 150% (6250 images) significantly increases training time. Image classification and object detection yielded similar times for experiments with similar parameters.

9 Conclusions

In conclusion, this project demonstrated that data augmentation can effectively enhance training datasets and improve the performance of image classification, yielding greater F1 Scores. Minor affine transformations from rotating the image at small angles consistently yielded the greatest improvement, closely followed by the colour modification techniques histogram equalisation and minor contrast increase, especially with an augmentation percentage of 150%. These methods successfully improved Object Detection performance (measured with F1 score and mAP) however image classification yielded a greater improvement. Furthermore, the results suggest that augmentation is particularly beneficial for improving classification in the presence of noisier or more complex images, achieving notable performance gains even when augmented and non-augmented datasets are of equal size. This project successfully met its primary objectives, while also highlighting the value of data augmentation, particularly in scenarios with limited training data.

9.1 Project Management

9.1.1 Supervisor Contact

Weekly communication with the project supervisor was maintained through Microsoft Teams and face-to-face meetings, utilizing Teams for its professional chat features, video calls, and screen sharing.

9.1.2 Project Management Tools

GitHub was chosen for its robust version control, reliable issue tracking, and seamless transitions between devices, facilitating continuous work and efficient code management. The cloud-based repository also provided a reliable backup.

ClearML [35] was used for efficient metric logging and test comparison, integrating seamlessly with TensorFlow and Ultralytics. It facilitated the tracking of results and monitoring of training progress. ClearML's tools simplified comparing tests with and without data augmentation, enhancing efficiency and organization. ClearML also streamlined test comparisons and organized data augmentation effects. Its artefact storage preserved essential test details such as metrics and hyperparameters. (Figures **??** & **??**)

9.1.3 Project Timeline

Figure **??** outlines the remaining work plan proposed in December 2023. Due to a significant personal circumstance, I was unable to work on the project for a substantial part of Semester 2. Consequently, I applied for and received a deadline extension to ensure successful project completion. This situation was not anticipated in the Risk Analysis defined in December 2023, increasing the need for the extension. In the future, I will reflect on this experience and include provisions for unexpected circumstances in the risk analysis to better prepare for potential disruptions.

After the deadline extension was approved, a new work plan was created (Figure **??**). This plan optimized time by writing the report during ongoing tests, allow-

ing time for other coursework and exams. The 'Implementing System Design' section extended past the initial schedule due to concurrent code development and testing.

9.2 Future Work and Improvements

External Computation: With no setbacks, a functioning Lyceum implementation may have been feasible. In future work, I would like to implement the project to utilise external computational services such as Lyceum or Microsoft Azure [37] - budget permitting - allowing for a more thorough investigation with a wider range of experiment parameters.

More Augmentation Methods: I plan to further explore augmentation techniques, including HSV shifting, Gaussian Blurring, and Mosaic Augmentation.

Beyond Images: I would like to explore the idea of leveraging real-time object detection (video) with YOLO. Since the AVOIDDS dataset was generated using X-Plane, a video dataset could be collected with corresponding metadata (time of day, weather, etc) using the university's X-Plane flight simulator. Applying augmentations in real-time is computationally challenging, however, with future advancements of YOLO and greater computational power via external computation, this may be feasible.