BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION Computer Science

DIPLOMA THESIS

ILLUSION: Image classification by light pollution levels and sky-glow informed constellation detection

Supervisors

Dr. CĂLIN Alina, University Assistant
Dr. COROIU Adriana Mihaela, Lecturer

Author
Murariu Alexandra

2023

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA Informatică

LUCRARE DE LICENŢĂ

ILLUSION: Clasificarea de imagini după nivelul de poluare luminoasă și detectarea constelațiilor în medii poluate

Conducători științifici

Dr. CĂLIN Alina, Asistent Universitar
Dr. COROIU Adriana Mihaela, Lector

Absolvent
Murariu Alexandra

2023

ABSTRACT

Light pollution has emerged as a significant issue in recent years, drawing attention to its adverse impact not only on the environment but also on our daily lives. It is crucial to implement measures to mitigate these negative effects by minimizing light pollution in areas where it has experienced exponential growth. Currently, there are no algorithms available that can determine light pollution from ground images. Therefore, the objective of this thesis is to classify night sky photos according to the Bortle scale and subsequently utilize computer vision to identify discrete constellation patterns while accounting for pollution noise. The proposed approach achieved promising results, with 99% accuracy in light pollution classification and a mAP50 score of 88.6% for constellation detection, considering the challenges posed by sky-glow, missing stars, and variations in constellation shapes near the horizon. These results open new horizons for addressing light pollution and provide motivation for future research in this field.

Contents

1 Introduction **1**

1.1 Subject overview . . . 1

1.1.1 Light pollution . . . 1

1.1.2 Constellation detection . . . 3

1.2 Research questions . . . 4

1.3 Original contribution . . . 4

1.4 Thesis overview . . . 5

2 Theoretical aspects **6**

2.1 Feature engineering . . . 6

2.1.1 Feature extraction . . . 6

2.1.2 Feature selection . . . 6

2.2 Traditional classification methods . . . 7

2.2.1 Logistic regression . . . 8

2.2.2 Support vector machine (SVM) . . . 8

2.2.3 K-Nearest Neighbors (KNN) . . . 8

2.3 Deep Learning . . . 9

2.3.1 Neural networks . . . 9

2.3.2 Convolutional Neural Networks (CNNs) . . . 10

2.3.3 YOLOv5 object detection . . . 13

2.4 Ensemble methods . . . 13

2.5 Evaluation metrics . . . 16

3 Literature review **18**

3.1 Light pollution classification . . . 18

3.1.1 Statistical, mathematical and astrophysics models used in light pollution related papers . . . 20

3.1.2 Classification of tabular data by light pollution characteristics 20

3.1.3 Classification of images by light pollution characteristics . . . 21

3.2 Constellation detection . . . 22

3.2.1 Constellation detection algorithms . . . 22

3.2.2 Star identification algorithms . . . 23
3.2.3 Applications of constellation detection . . . 27

4 Methodology: Classification on the Bortle scale 28
4.1 Dataset . . . 28
4.1.1 Data generation . . . 28
4.1.2 Data analysis . . . 32
4.2 Experiments using simple models on mean intensity values . . . 35
4.2.1 Model 1: Logistic regression . . . 35
4.2.2 Model 2: SVM . . . 36
4.2.3 Model 3: KNN . . . 36
4.3 Experiments using deep learning models . . . 36
4.3.1 Discussion of results: interpretation and analysis . . . 38
4.4 Feature engineering . . . 38
4.4.1 Feature extraction using deep neural networks . . . 38
4.4.2 Feature selection . . . 39
4.4.3 Discussion of results: interpretation and analysis . . . 39
4.5 Ensembling . . . 40
4.5.1 Ensembling of CNN models . . . 40
4.5.2 Ensembling of feature extraction models . . . 42
4.6 Testing on real data . . . 43

5 Methodology: Constellation detection 45
5.1 Dataset . . . 45
5.1.1 Image generation . . . 45
5.1.2 Automatic bounding boxes generation . . . 45
5.1.3 Data exploration . . . 49
5.2 Image preprocessing . . . 49
5.2.1 Contrast stretching . . . 50
5.2.2 Image filtering . . . 51
5.3 Object detection algorithm . . . 52
5.3.1 Custom preprocessing layer . . . 52
5.4 Training . . . 53
5.5 Results & discussion . . . 55

6 Methodology: Mobile application 57
6.1 Functionalities . . . 57
6.2 Architecture . . . 58
6.2.1 Diagrams . . . 58
6.2.2 Backend . . . 59

6.2.3 Frontend 61
6.2.4 Design patterns 61
6.3 Testing 62

7 Conclusion **63**
7.1 Comparison with SOTA 63
7.2 Answers to research questions 64
7.3 SWOT analysis 65
7.4 Future improvements 66

Appendix **66**

A Mathematical models that describe light pollution **67**

B CNN models' architectures **69**

C History of training processes and confusion matrices of CNN models trained on raw data **73**

D Confusion matrices: stacking of CNN models **76**

E Confusion matrices: ensembling of models trained on extracted features **79**

F Constellation detection metrics **81**

Bibliography **86**

Chapter 1

Introduction

1.1 Subject overview

1.1.1 Light pollution

Light pollution is a widespread and far-reaching issue that has transformed the night sky around the globe. Regrettably, only fragments of our world now retain the pristine darkness that once enveloped us each night. In 2016, the New World Atlas of Night Sky Brightness [FCD+16] used a multitude of satellite images to create a comprehensive map (Figure 1.1), which sheds light on the extent of this phenomenon. Currently, complete darkness is predominantly experienced in the most remote regions of the world, such as Siberia, the Sahara, and the Amazon.

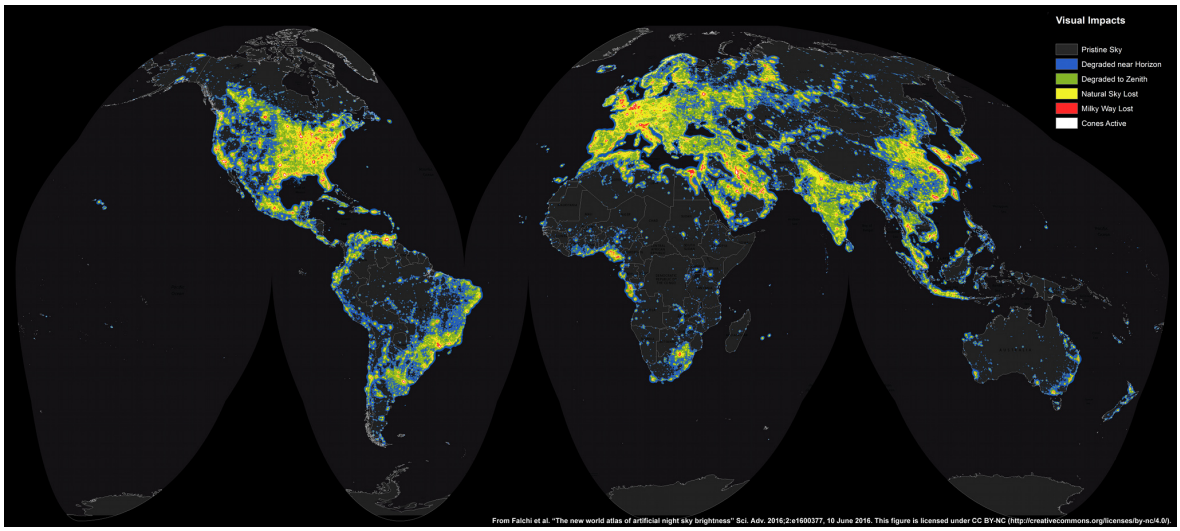


Figure 1.1: The color-coded light pollution map provides a visual representation of the varying degrees of pollution across different areas. Regions highlighted in red and white indicate the highest levels of pollution, while black areas represent locations where the night sky remains closest to its natural, unadulterated state. [FCD+16]

The Bortle scale, introduced by John E. Bortle in 2001 [Bor01], serves as a widely adopted metric for assessing the level of light pollution. This scale categorizes light pollution into nine distinct levels of intensity, taking into consideration the naked-eye limiting magnitude of the night sky (dimness of stars that our eyes can see on that sky). A visual representation is presented in Figure 1.2.

Figure 1.2: Bortle scale (1 represents a very dark sky, while 9 a very polluted one) [MH23]

A study [FH23] from 2023 presents the multitude of factors that contribute to light pollution. These factors include the extent of residential and industrial coverage, as well as the quality of lighting practices adopted by the population. Additionally, the presence of water bodies, such as rivers, can exacerbate light pollution by reflecting artificial lights. Vegetation coverage, on the other hand, can help mitigate light pollution by diffusing the reflection and reducing its impact. Economic indicators, such as GDP, can also play a role in driving the adoption of greener industrial practices that aim to minimize environmental pollution, including light pollution. Furthermore, air pollution contributes to the scattering of light, further intensifying the effects of light pollution. Collectively, these factors contribute to the complex and multifaceted nature of light pollution in our modern world.

The impact of light pollution extends far beyond what may initially come to mind. One significant consequence is the loss of cultural heritage as the dazzling sight of the Milky Way and a sky adorned with stars becomes increasingly rare. This cultural deprivation robs us of a profound connection to the vastness of the Universe that lies above us. Furthermore, light pollution disrupts the circadian rhythms of animals, leading to confusion among migratory birds in their navigation

and interfering with communication among various species. Humans are also not immune to its effects, as light pollution contributes to sleep deprivation and fatigue, affecting overall well-being. Additionally, it hampers astronomical research and obscures our ability to observe celestial phenomena. The ecological balance, energy consumption, and even human health and safety are all affected by the widespread presence of light pollution. These consequences underscore the urgent need to address and mitigate the harmful effects of light pollution on our environment and society.

Although numerous methodologies have been developed to measure and assess light pollution, none have specifically focused on using night sky images and AI for classification purposes. This thesis attempts to fill this gap by employing innovative techniques that leverage AI to classify light pollution levels from nocturnal sky imagery. This can facilitate the formulation of targeted strategies to mitigate light pollution and preserve the integrity of our natural environment.

1.1.2 Constellation detection

There are 88 constellations on the night sky, some of them fully visible from only one hemisphere. The problem of detecting constellations comes from ancient times, when it was essential for navigators to know where they are headed based on the stellar map above them. Over the centuries, civilizations have developed various methods to recognize and interpret these celestial patterns, all with the shared goal of orientation and spatial awareness.

Today, because of advances in spacecraft technologies, the lost-in-space problem becomes crucial for the determination of attitude (orientation of the spacecraft). It entails identifying celestial objects without any location information, and is widely used in star sensors (tools used by satellites, for example, to identify stars and thus determine the attitude). Furthermore, the development of an automated constellation detection tool serves the purpose of enhancing people's knowledge and understanding of constellations, by using it.

In this thesis, I developed an object detection algorithm for constellation identification, which integrates information about pollution levels to accurately determine constellations. By incorporating pollution data, the algorithm is able to effectively detect constellations in both polluted and non-polluted images, providing a comprehensive and adaptable solution for constellation detection under various environmental conditions.

1.2 Research questions

During the development of my models, I wanted to answer the following questions:

- **RQ1**: Can machine learning models effectively classify different levels of light pollution from night sky images?
- **RQ2**: How does feature engineering with traditional models compare to deep learning approaches in classifying light pollution?
- **RQ3**: How do object detection algorithms perform in identifying discrete patterns such as constellations?
- **RQ4**: Can the same model be trained to detect constellations in images with varying levels of light pollution?
- **RQ5**: What are the limitations and challenges encountered in classifying light pollution and detecting constellations using the proposed approaches?
- **RQ6**: What are the practical implications and potential applications of the models developed for light pollution classification and constellation detection?

1.3 Original contribution

Through extensive experimentation and analysis, several key contributions have been made:

- I generated two datasets: one with 4800 images for light pollution classification, split in 12 classes of artificial brightness values, and one with 24000 images, with the same even distribution of light pollution, that was used for constellation detection.
- I implemented an algorithm that can calculate the constellations' bounding boxes' coordinates from Stellarium generated images, given the coordinates of the observer.
- I tried to classify light pollution based on night-sky images, an approach that is original for this problem, since related work only focuses on SQM or satellite data.

- I used convolutional neural networks and feature engineering for light pollution classification, while also testing models on the pure mean intensity values of the pixels.
- I made a light pollution informed constellation detection algorithm, by adding a preprocessing layer at the beginning of the neural network.

1.4 Thesis overview

The thesis is structured into several chapters that provide a comprehensive exploration of the research topic. Chapter 2 covers the theoretical aspects of machine learning, including feature engineering, machine learning models, ensembling techniques, and evaluation metrics. Chapter 3 delves into related work, examining previous studies on light pollution estimation and classification. It also investigates research on constellation detection and celestial object identification, discussing the algorithms employed in these areas. In Chapter 4, the methodology for classification of light pollution is presented, along with the corresponding results. Chapter 5 focuses on the methodology for constellation detection and its results. Chapter 6 explores the development of the mobile application, detailing its architecture, diagrams, and testing. Finally, Chapter 7 concludes the thesis, summarizing the key findings, discussing possible future improvements, and providing a comprehensive conclusion.

Chapter 2

Theoretical aspects

In this chapter, I will present some key theoretical aspects that are used in this thesis.

2.1 Feature engineering

Feature engineering represents the extraction and/or modeling of different features from the input data, to prepare them for the machine learning steps to follow. This is a crucial step sometimes, especially when dealing with missing data, or with data that cannot be processed by the learning algorithms and needs to be transformed in some way.

2.1.1 Feature extraction

Feature extraction is a part of feature engineering. Especially when working with images, it is a challenge to transform the raw input data (pixel values) into something significant for the models that perform the learning task. There are feature extraction techniques that require a good domain knowledge, for example edge detector filters, making use of histogram distributions for estimating shapes of objects present in the image, or Scale-Invariant Feature Transform (SIFT). However, deep learning models (Section 2.3) can also be used to extract relevant features from images. In Figure 2.1, it is shown how convolutional neural networks extract features from an input image: first, low-level features, then mid-level and in the end, high-level features.

2.1.2 Feature selection

Out of the extracted features, only the relevant ones should be used in the learning task, in order to reduce computational costs, but also prevent the model from

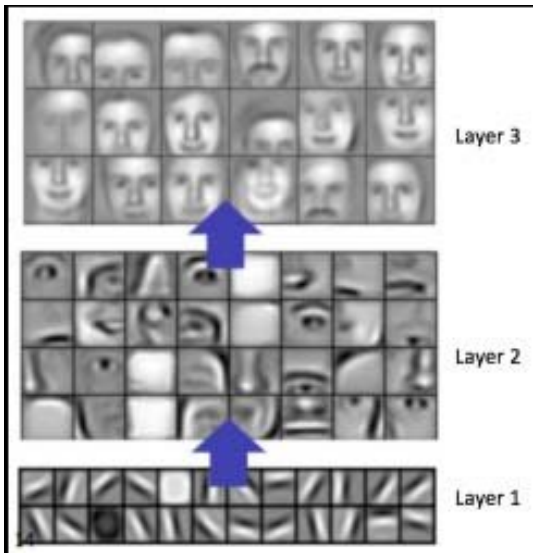


Figure 2.1: Feature extraction with a convolutional neural network [AAMA17]

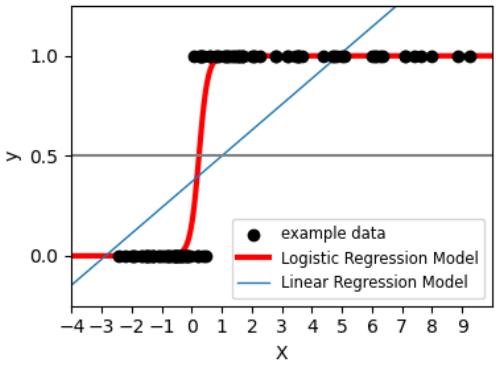


Figure 2.2: Logistic regression: 2D data [PVG+11]

learning from irrelevant characteristics.

Feature selection using a Random Forest selector

In random forests, each of the trees is trained on a subset of the initial data. By measuring the average increase of purity (a measure of how well a node separates data into classes) that a feature yields across all trees from the random forest, one can measure how important that feature is.

Feature selection using an Extremely randomized trees selector

Extremely randomized trees (extra trees) exhibit similarities to random forests, but have some key differences. In both of them, decision trees are built in order to make predictions. However, extra trees do not use bootstrapping (sampling with replacement) when sampling the data, and also nodes are split randomly, without taking into account a "best" split, like in random forests. Thus, extra trees are faster and have lower variance, as the data is generalized through the introduced randomness.

2.2 Traditional classification methods

Traditional classification methods have roots in classical statistical techniques. They are very interpretative approaches that are effective in classification tasks. In the following, I will present the algorithms that I used in this paper.

2.2.1 Logistic regression

This statistical model is typically used in binary classification. It tries to find the parameters for the logistic function that best fits the data points [Gé19]. In Figure 2.2 an example of such a fitting is presented on a 2-dimensional feature dataset.

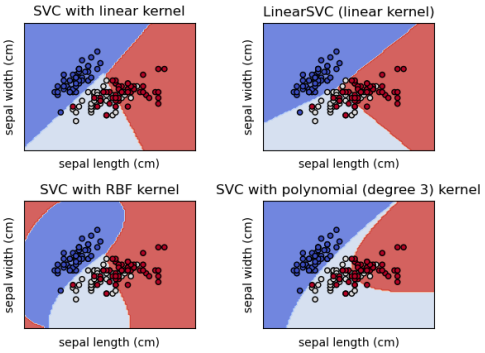


Figure 2.3: Illustration of SVC kernels algorithm [PVG+11]

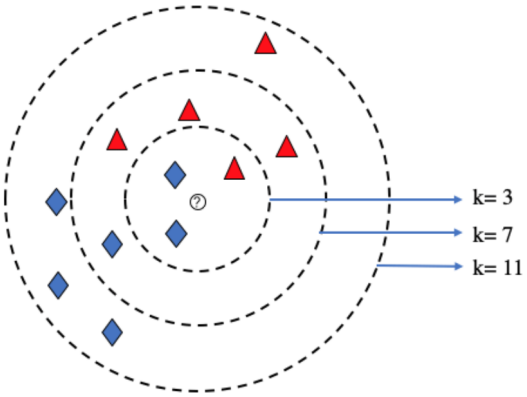


Figure 2.4: Choice of k for the KNN [ZCLX20]

2.2.2 Support vector machine (SVM)

This algorithm is very popular for classification tasks. It tries to find the perfect boundary between classes, even in high-dimensional feature spaces. The support vector machines solve the constrained optimization problem of finding hyperplanes (defined as a set of points x that are solutions to the equation $w^T \cdot x = b$, where b represents a bias term and w defines the orientation of the hyperplane - normal vector to it) that best separate the data into classes (this is the constraint of the optimization problem)[Gé19]. SVM basically tries to find, by transforming the data using kernels, a separation of classes. Figure 2.3 depicts different types of kernels and how they manage to divide the 2D data points by class.

2.2.3 K-Nearest Neighbors (KNN)

This is a simple, yet flexible model that establishes a class correlation among data points that exhibit proximity in the feature space. It calculates the distance between each new point and the others and, depending on the parameter k (number of neighbors), assigns the new data point to the class that is most common among the k closest data points [Gé19]. Figure 2.4 shows a visual representation of the choice of k, in a two-dimensional feature space.

2.3 Deep Learning

Deep learning represents a subset of machine learning, focused on artificial neural networks with multiple layers. It eliminates the need for feature engineering, as the models are capable of extracting high-level features by themselves. They are split into several categories [AZH+21]:

- **deep supervised learning**: when the training set contains labeled data
- **deep semi-supervised learning**: data is only semi-labeled
- **deep unsupervised learning**: no labels, the learners need to find relationships in the input data
- **deep reinforcement learning**: the learner interacts with an environment and adapts to it, by gaining feedback

My focus will be on deep supervised learning, specifically on the use of different convolutional neural networks, which will be presented below in Subsection 2.3.2.

2.3.1 Neural networks

Neural networks work as a normal brain, in the sense that there is no central unit of information. They learn based on the weights that are updated at each iteration through the network [PG17]. Feed-forward multilayer neural networks have a simple structure, as they consist of an input layer (input data), hidden layers (these model some nonlinear functions of the input features), and an output layer. They learn from backpropagation, where, based on gradient descent, the weights are updated in order to minimize a chosen loss function. Weights represent coefficients that are a quantification of the importance of that feature in the learning task (a weight of zero would mean that the feature is ignored). Even if the signal is low, which means that most of the weights have really low values, a bias term enables the model to explore new approaches, as it is added to the weighted feature, regardless of the weight value. By an activation function, the output value of a layer (the weighted features) is passed to the next one (forward propagation). In Figure 2.5, some of the graphs and equations of the most used activation functions are presented.

Learning optimization

Sometimes in backpropagation, gradient descent might make too big or too small steps in the direction of minimizing the loss function. However, there are some optimization techniques that can prevent this from happening.

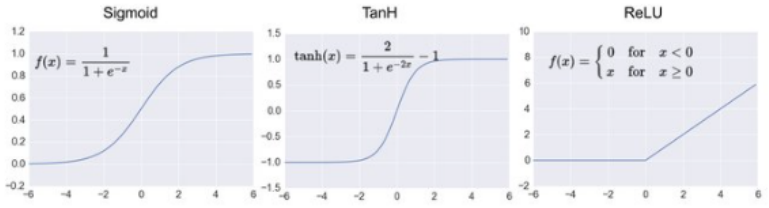


Figure 2.5: Most commonly used activation functions [NB21]

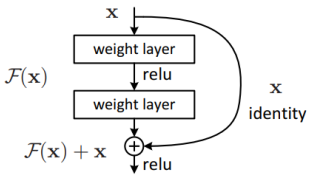


Figure 2.6: Residual block [HZRS16]

Adam optimizer (Adaptive Moment Estimation) is based on stochastic gradient descent and combines other optimization algorithms: AdaGrad and RMSProp. It adjusts a separate learning rate for each parameter, and uses a history of past gradients and their squares to update the parameters [KB14].

Dropout works by randomly dropping a neuron from an iteration of the neural network. It helps speed up the training process, while also controlling overfitting.

2.3.2 Convolutional Neural Networks (CNNs)

CNNs are very popular among deep learning networks. They are particularly employed in the field of computer vision, specifically for tasks involving image classification and object detection, and achieve the best results in competitions. Their advantages come from being able to extract relevant features from images, regardless of positions or rotations. This works because usually, images have a kind of local similarity between features whose patterns can be detected, in contrast to tabular data, in which rows do not commonly have relationships between themselves.

Layers typically used in CNNs

Convolutional layers are considered the building blocks of CNNs [PG17]. They use convolution, which works by sliding a kernel through the image and multiplying the kernel values by the pixel values, resulting in an activation map. Convolutional layers are able to capture important features from the image.

Pooling layers usually are placed between successive convolutional layers to reduce the size of the data. They prevent overfitting by reducing the feature dimensionality. Max-pooling outputs the maximum value present in a kernel, while average-pooling computes the average of those values.

Fully-connected layers are typically used as output layers, in order to compute the results (final predictions). Some CNN architectures have multiple fully connected layers at the end of the network.

MobileNetV2

This model was developed by Google researchers back in 2018. It is intended to achieve a balance between computational costs and accuracy. Thus, it is an ideal candidate for mobile applications, where the inference time needs to be as low as possible. It uses certain techniques to achieve a fast training time, such as inverted residuals, linear bottlenecks, and shortcut connections [SHZ+18]. The 53-layered architecture behind this model is described in Figure B.1 (Appendix B).

EfficientNetB7

This is a state-of-the-art CNN model in image classification tasks, developed in 2019 by Mingxing Tan and Quoc V. Le. It uses a combination of neural network scaling and compound scaling techniques to achieve high accuracy with fewer parameters [TL19]. Based on ImageNet [DDS+09a] dataset accuracy with a variable number of parameters, EfficientNetB7 was proven to be more accurate than many other state-of-the-art models [TL19]. Figure B.2 (Appendix B) presents the 813-layered model architecture.

ResNet50V2

Although not considered a state-of-the-art model anymore because of the multitude of new discoveries since its appearance, this variant of the ResNet (Residual Network) is still a very powerful image classification model. It was created by Microsoft researchers in 2016. It consists of 50 layers (hence the name), which are illustrated in Figure B.3 (Appendix B). The uniqueness of this model consists of the incorporated shortcut connections, a solution to the ”degradation problem”: performance drops as the model has a deeper architecture. These connections can skip some layers of the model, allowing the input of each residual block (Figure 2.6) to be added to its output. During backpropagation, the gradients flow back on this shortcut path [HZRS16].

YOLOv8 classifier

YOLOv8 is a state-of-the-art model developed by Ultralytics. It is capable of classification, object detection, and pose tasks. The model architecture is divided into two parts: the backbone and the head. The backbone consists of 53 convolutional layers and uses cross-stage partial connections to disperse information across layers, in order to improve learning capabilities. It also processes images at different scales to capture both fine-grained (edge, corners, textures) and coarse-grained (larger patterns, such as shapes) information effectively. The head performs the final

classification task, having only a classification layer with a number of output channels equal to the number of classes (it outputs probabilities for each class) [JCQ23]. Figure B.4 (Appendix B) shows the model architecture (note: the head in the figure is more complicated because the figure shows the architecture for the object detection model, not the classifier model).

InceptionV3

InceptionV3 was created by Google back in 2015. It is made of 42 layers as presented in figure B.5 (Appendix B). This model uses asymmetric convolutions, as illustrated in figure 2.7. Basically, instead of using just a simple 3x3 kernel, by using a 1x3, 3x1, and 3x3 kernel the model can capture features at different scales more efficiently, while also being more cost-effective. InceptionV3 instead uses a combination of 1x1 convolutions and 3x3 convolutions to perform the grid size reduction. The model is made of several modules (building blocks). The application of asymmetric convolution on those modules is represented in figures 2.8 (before) and 2.9 (after).

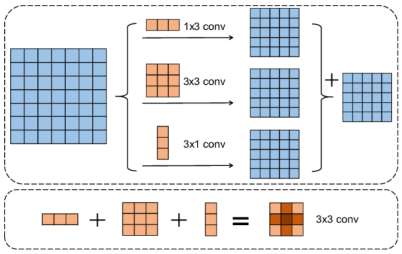


Figure 2.7: Asymmetric convolution
[ZZL+20]

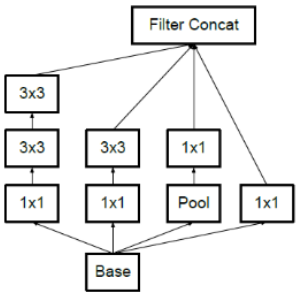


Figure 2.8: InceptionV3 - base module
[SVI+16]

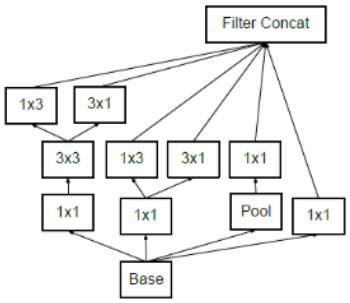


Figure 2.9: Base module - after applying asymmetric convolutions
[SVI+16]

VGG16

The VGG16 model was introduced in 2014 by an Oxford team. It consists of 16 weighted layers (hence the name). This is one of the first models to have started adding more layers for improving accuracy, instead of having more hyperparameters [SZ14]. Its architecture is presented in figure B.6 (Appendix B). This model was proved to offer very good results as a feature extractor, compared to other convolutional neural networks [SZ14].

2.3.3 YOLOv5 object detection

YOLOv5 [JCS+22] is a powerful real-time object detection model developed by Ultralytics. It is popular for its high speed and accuracy among other object detection models. YOLOv5 is the fastest of the YOLO models in terms of real-time detection inference times (measured in fps) [Ste21], and is also one of the fastest object detection models in general [MRMS23]. The v5s model is also less computationally expensive than the bigger model variants, so it enables a faster training process while also having decent performance.

Model architecture

YOLOv5 is a single-stage object detection model that consists of three parts: backbone, neck, and head. The backbone extracts image features using the CSP-Net strategy, and thus reduces the space complexity of the problem. The neck tries to make feature pyramids, which help the model detect objects of different sizes in different-scaled images. It selects the most relevant features from the backbone and aggregates them. Finally, the head is responsible for outputting the boundary box and the label predictions [XLL+21]. YOLOv5 uses the same head architecture as its last 2 predecessors, which consists of 3 convolutional layers. The entire architecture is presented in figure B.7 (Appendix B).

2.4 Ensemble methods

Ensemble learning became more popular in recent years due to its capabilities. It represents combining several machine learning algorithms in order to increase the overall accuracy of the model. Ensembling consists of mixing training data, combinations of model outputs, or models as a whole.

Random forest

Random forests are ensembles of decision trees (weak learners, if/then type of trees that consist of a sequence of ”decisions” based on different criteria to a final leaf, which represents the output class - Figure 2.11), that are trained on different subsets of the dataset, which are selected with replacement (bootstrapping). Each decision tree splits the data based on certain criteria, until a minimum amount of data is left at a leaf or a maximum depth is reached. The predictions of each decision tree are then aggregated, typically using a voting mechanism [KJ20]. The architecture of the random forest is depicted in figure 2.10.

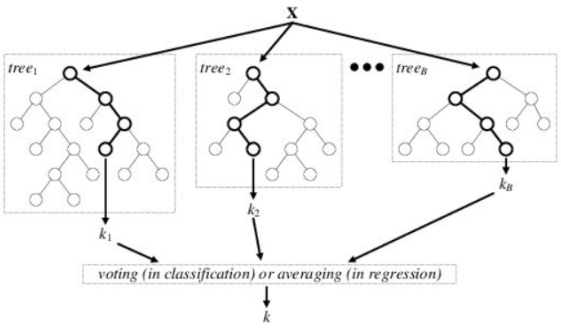


Figure 2.10: Random forest - architecture [VVG+16]

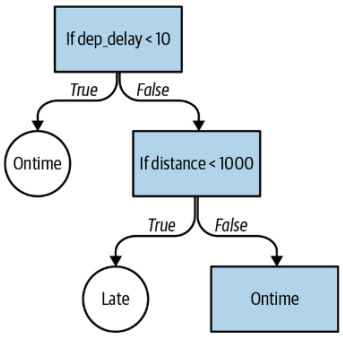


Figure 2.11: Example of decision tree that decides whether the flight will be on time [Lak19]

Voting

Voting represents an ensemble algorithm that takes into account the outputs of multiple models to reach a consensus or agreement. Considering the choices made by each individual model, voting aims to improve overall prediction accuracy and robustness.

Hard voting

Hard voting works like an election [KJ20]: basically, each model ”votes” the predicted class, and then the majority vote is the winner: thus, if 3 of the 4 models decide that class ”poor” is the correct class of the image, this shall be the final decision of the ensemble system.

Soft voting

The difference between hard and soft voting (averaging) is that the latter uses the probabilities of the image being each of the classes from all models (not just the predicted class) in a classification task, and then performs a weighted sum of all these probabilities, based on the model's importance [Bey17].

Stacking

Stacking involves some base learners (models) that are pre-trained and a meta-learner that learns how to aggregate the predictions of the base learners in order to make the best decision. A very nice analogy is presented in the book [KJ20]: suppose that a person is on a game show and is asked a history question to which he does not know the answer. However, he has the option to ask one of his friends for input.

He knows that one of his friends is a history major, while the others do not know as much history. Thus, he is able to make the decision to trust the answer that his history major friend gives. This is the exact same way stacking works: if one of the base models excels in predicting, let us say, class "poor", the meta-learner will learn to trust that model's input if it predicts the "poor" class.

Boosting

Boosting is an ensembling technique which aims to improve the accuracy of some given learning algorithm. Some learners are trained on subsets of the data, and, if one of them does not perform well for some input, its training becomes concentrated on that specific defect, in order to improve it and thus the overall performance of the ensemble.

AdaBoost

AdaBoost (adaptive boosting) is a boosting algorithm that constructs several decision tree classifiers, which will learn from misclassified data from previous ones. Basically, each time AdaBoost gives a bigger weight to the misclassified images, in order to increase their importance at the next training iteration. The decision is taken in the end by voting between the generated decision tree classifiers [KJ20]. The strategy of this model is presented in figure 2.12.

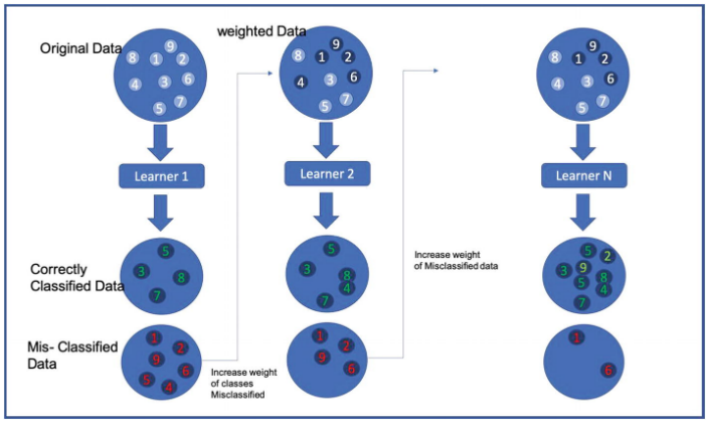


Figure 2.12: AdaBoost algorithm architecture [KJ20]

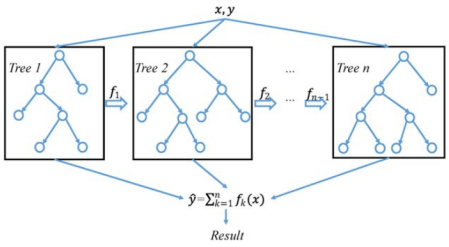


Figure 2.13: XGBoost architecture [WPZ+19]

Gradient boosting

This algorithm is similar to AdaBoost, as it boosts the weak predictions of previous decision trees. The multiclass classification problem is translated into several binary classifications. For each of these problems, the algorithm starts with a single

leaf that "predicts" an initial value for each image. This initial value is calculated by taking the logarithm of the odds that an image will be of a specific class. Next, the pseudo-residual is calculated, which represents the difference between the predicted value and the actual one (the class label value). The next tree that will be built will take as input this residual instead of the actual predicted value and will contribute to the next prediction limited to a learning rate value, to ensure a slow but steady increase in accuracy. In the end, the trees will learn to predict the errors of their predecessors better and better, thus improving the overall accuracy of the ensemble [Ser18].

XGBoost

XGBoost (Extreme Gradient Boosting) is a state-of-the-art gradient boosting algorithm that provides very good results in a variety of classification tasks. It was introduced by Tianqi Chen and Carlos Guestrin in 2016 [CG16]. This algorithm combines some weak learners in a way that makes the final predictions more accurate. It creates decision trees from subsets of data and aggregates their results to produce the verdict (Figure 2.13). XGBoost also uses regularization, in order to prevent overfitting, it adds randomization parameters and dynamically determines the depth of decision trees, which improves the performance [KJ20].

2.5 Evaluation metrics

Accuracy

Accuracy is an evaluation metric that represents the percentage of correct predictions made by the model. However, it is not always insightful, and usually other metrics are also needed to understand the performance of a model.

Confusion matrix

Confusion matrices are visual representations of the differences between actual and predicted classes, per class. A binary classification example can be seen in Figure 2.14. TP (true positives) represents the number of predictions that were correctly assigned to the positive class, FN (false negatives) represent how many were positive but were assigned negative class, FP (false positives) are the number of negative inputs that were assigned the positive class, and TN (true negatives) represent the number of correctly assigned negative class inputs.

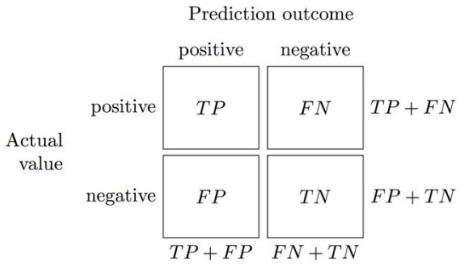


Figure 2.14: Confusion matrix on binary classification (classes ”positive” and ”negative”)

[MVM+16]

Object detection metrics

The most common object detection algorithms' evaluation metrics are presented below [Sha19]:

- box loss: how much the predicted boxes differ from the true bounding boxes
- object loss: the probability of the predicted bounding box to contain an object, compared to the truth
- class loss: the probability that the predicted class is correct, compared to the truth
- precision: percentage of correctly detected objects from the total number of positive detections
- recall: percentage of true positives from true positives and false negatives
- AP (average precision): the area under the precision-recall curve
- IoU (intersection over union): the area of overlap over the area of union between the predicted bounding box and the ground truth one
- mAP50 (mean average precision at 50% IoU): average precision (AP) where a detection is considered correct if the IoU between the predicted and true bounding boxes is greater than 50%.
- mAP50-95: the average mAP calculated over the range of thresholds from 50% to 90%.

Chapter 3

Literature review

3.1 Light pollution classification

Despite its relevance in modern times, light pollution has received relatively limited research attention in recent years. Many studies focus mainly on highlighting the negative impacts of light pollution, rather than delving into its analysis and developing a comprehensive model to characterize it.

A study conducted in 2022 [RIU+22] examines the frequency of articles related to light pollution data analysis, and the approach used to investigate this pollution effect. They found a total of 54 published and peer-reviewed English papers on the matter, from 1980 until 2022. Next, they categorized the papers based on certain criteria:

- **Research objective**: Most articles (47.76%) concentrated on evaluating the quality of light pollution measurement means, while the next two most frequent objectives were regression for prediction purposes and correlation between different pollution characteristics, or between measurement methods. Classification comes only fourth in line, with 6.78% of articles focusing on clustering various pollution data and trying to find similarities between them. The papers that concentrate on classifying areas by light pollution seek to find a connection between its intensity and the negative impact on the ecosystem, or search for proper dark sites that can be used for stargazing.

- **Data preprocessing**: Of the found articles that use preprocessing on input data, the majority use image preprocessing (51. 61%), while the next are statistics and machine learning methods.

- **Data collection**: Most of the datasets used in the selected literature are collected from ground-based sensors (31.91%) - most commonly, SQMs, but also wide-frame photography, while the next preferred methods, in this order, were

remote sensing (data from satellites or aircraft), other, survey/sampling, laboratory/experiment.

- An important thing to note here are the differences between satellite images datasets, which are presented in article [YLT+20]. Most common nighttime satellite photos are taken from the DMSP (Defense Meteorological Satellite Program) or VIIRS (Visible Infrared Imaging Radiometer Suite). However, they have a coarse resolution, which makes local region analysis harder. Recently, more satellite types have appeared, such as EROS-B and JL1-3B, which have a finer spatial resolution. However, the costs of purchasing images taken by them are high and thus discourage research done on proper satellite data.
- A study [KL16] from 2016 compares field measurements of light pollution with EROS-B images over the city of Jerusalem. It expresses the need to consider the three-dimensional characteristics of light propagation and proves this by measuring the brightness values in all three directions and getting very different numbers (Figure 3.1). Usually, SQM measurements are taken in the upward direction, but they proved that the correlation with EROS-B brightness values is the lowest when compared in this direction of measurement. Although SQM (Sky Quality Meter) data is commonly utilized in studies as a reference to assess the accuracy of models or as a foundation for model development, it is important to note that its numbers may not always serve as the most suitable descriptors of light pollution.

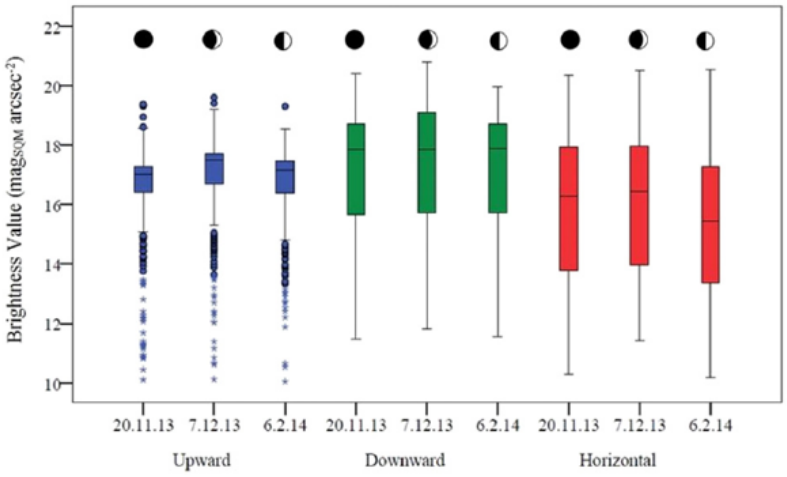


Figure 3.1: Comparison between brightness values measured by SQM in different directions. The phase of the moon for each data point is also illustrated. [KL16]

3.1.1 Statistical, mathematical and astrophysics models used in light pollution related papers

Many papers used various types of mathematical models in order to solve light pollution related tasks, however only some of them did so in order to better model light pollution, disregarding other matters. Most articles concentrated on finding a correlation factor between light pollution data and environmental problems, not on better describing and understanding light pollution itself. However, some research has also been done on this part, and multiple models have been developed for a better approximation of the light pollution variables. In a study from 2020 [SYT+20] and another from 2009 [SLSMP09], such models are presented and evaluated. More details about the following models can be found in Appendix A.

Walker's model [Wal73] was the first model specifically created for measuring light pollution. The Garstang model [Gar86] has been widely used in research of the ecological effects of light pollution. It was adapted to create the first World Atlas of Artificial Night Sky Brightness in 2001 [CFE00]. Treanor's model [Tre73] takes into consideration the scattering of light by suspension molecules, while Berry's model [Ber76] addresses the limitations of Treanor's model by accounting for upward glare from city lights. Aubé's model [AFFRSH05] incorporates variable city shapes and luminosity. Kocifaj's model [Koc07] is a comprehensive mathematical model that can be applied in various atmospheric conditions. Duriscoe's model [Dur13] incorporates atmospheric extinction and provides better results for areas with less pronounced light pollution. It was used in the creation of the New World Atlas of Artificial Night Sky Brightness, composed in 2016 [FCD+16].

Although many mathematical models were created to represent the effect of light pollution, the use of machine learning for this task is sparse. Most articles use machine learning or statistical analysis to find a correlation between light pollution and some of its consequences [XLZ+20, LSHJ+22].

3.1.2 Classification of tabular data by light pollution characteristics

In the paper by Sainger et al. [SYT+20], machine learning models were used to create a hierarchy of locations based on their levels of light pollution.

Data was taken from the website of the International Dark-Sky Association (IDA) (`darksky.org`), from 2011 until 2018. It was captured through the Defense Meteorological Satellite Program (DMSP), which consists of several satellites launched by the United States Air Force to monitor different meteorological events [SS]. Locations from different states were considered. The attributes of the dataset are: state,

location, latitude, longitude, SQM, natural brightness, artificial brightness, Bortle scale value, elevation of the place, and population.

The paper presented the following separation of classes for brightness values (in mcd/m^2), which represents an inspiration for my own separation of classes in the developed algorithm:

- < 1 (> 20.08 mag/$arcsec^2$),
- 1-5 (18.33-20.08 mag/$arcsec^2$),
- 6-9 (17.69-18.13 mag/$arcsec^2$).

Two models were used for classification: random forest and decision tree (using the Iterative Dichotomiser 3 algorithm [Qui86], which creates several decision trees that correctly predict the output, while ultimately taking the simplest one of them). They reported 99.7% accuracy for the decision tree and 100% accuracy for the random forest algorithm, a 98.715% cross-validation score for the decision tree, and 99.917% for the random forest.

This paper also conducted a study where they compared the outputs of the two algorithms with Berry's model. They used the RMSE value, which is a metric that indicates how far the predicted value is from the observed one. They concluded that the machine learning approach was more efficient, given the smaller deviation from the true values.

3.1.3 Classification of images by light pollution characteristics

In a paper [YLT+20] from 2020, a study was conducted to classify the area of Wuhan, China into 3 categories, based on night-time satellite images: stable non-urban class (area from a non-urban place to another non-urban place), stable urban class (area from an urban location to another urban location), and expanding urban class. The study used two images, one taken from Luojia1-01, the first Chinese night-time light imaging satellite with open-source data, while the other was taken from the International Space Station (ISS).

In order to determine whether the pixels are part of a "lit" zone, they used Otsu threshold segmentation [GW18] (which separates pixels into two classes based on a threshold intensity value), applied on the HSV transformed images. Gaussian filter (an algorithm used to reduce noise from images) with size k=9 and standard deviation σ was applied to the ISS photograph. Then, they took the pixels that were part of the lit area in both images. Four datasets were created, one with pixels of the LJ1-01 image, and one for each band (red, green, and blue) from the ISS image. After that, they trained a regression model depending on pixel values from each of

the datasets on the remaining pixels, and removed any outliers that appeared from all datasets.

Next, they investigated 3 approaches of getting features from different images, in order to achieve the best class separation: they started by trying only with the initial two images, then by also using the Otsu segmented images, and by adding another derived band from the segmented image. To check how well the classes are separated, they used the Jeffries-Matusita distance as a metric, calculated on the digital number (DN) value of the pixels (this is a cuantification of the brightness level of the pixel). A larger distance essentially means a better separation of classes. Two datasets containing 1000 pixels each were created, one for training and one for validation. The classification was done using SVM and BP-Neural Network with 3 layers and logistic activation function.

The overall accuracy on the validation set for the SVM was 89.6%, while for the neural network it was 90.7%. Most misclassified pixels were the ones of stable non-urban class and expanding urban class.

3.2 Constellation detection

I was able to find just two articles that investigate the specific task of constellation detection. However, there exist many implementations which focus on space objects' detection. Most of them are focusing on pattern recognition, but more recently, as deep learning has become more and more advanced, its algorithms are also present in such tasks.

3.2.1 Constellation detection algorithms

A paper [GM21] from 2021 studied the effectiveness of using neural networks for the constellation detection task. They generated a dataset of 1 284 780 images of 240x240 resolution from the Tycho-2 star catalog, which are labeled with an 89 binary vector, where each bit represents the presence or absence of a certain constellation in that image. For the test images, 48 772 generated images were used. The team used a ResNet-like architecture with 26 trained layers and an output layer of size 89, resembling the probability of each constellation being in the image. During training, they used Adam optimizer and a custom loss function, based on the area of each constellation. They obtained a F_1 score of 0.927. However, this is more of a classification task, since the location of the constellation is not present in this output, only information about its absence/presence inside that image.

A final report [JWL15] from Stanford University also followed the constellation detection task. They took a template matching approach, in which they prepro-

cessed images by binarization to keep the stars only, and then compared the image to template images for each of the constellations. For each template constellation from the dataset, they put all stars in a coordinate system, in which the brightest star was at position (0,0) and the second brightest star at (1,0). Then, all distances from all stars inside the constellation are scaled to the distance between the first and the second brightest star into that coordinate system. An example of how the scaling of the template constellation Gemini (figure 3.2) is performed is presented in figure 3.3. On the test image, they find the brightest 2 stars by region, and then try to find the matching constellation by searching in the templates by order of stars in each constellation. They state that they eliminated the "undetectable constellations" which contain only 2-3 stars, and that they also used information about the neighbors of each constellation in order to make improvements on speed and accuracy. However, the average speed is very high (85s/image), and the accuracy is 71. 4%.

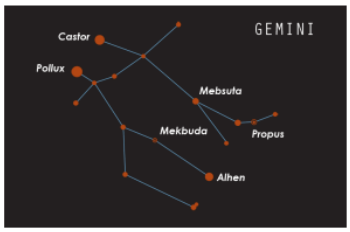


Figure 3.2: Template image of Gemini [JWL15]

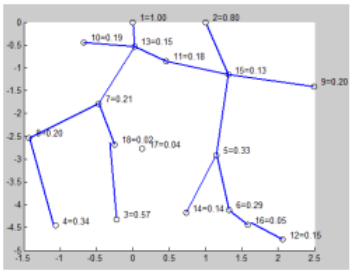


Figure 3.3: Stars of Gemini plotted into the scaled coordinate system [JWL15]

3.2.2 Star identification algorithms

These types of algorithms are especially used for autonomous navigation systems. Usually, in order to determine the orientation and attitude of a spacecraft, star sensors are used, which have a very high accuracy. The star identification part of a star sensor can be done using a variety of algorithms. The first two and most used algorithms that will be presented rely on a database (star catalog) which has the coordinates and brightness of each star.

First, the subgraph isomorphism algorithms attribute a subgraph to the stars in the image. Basically, stars are vertices, and the angular distances between stars are weights of edges. The entire graph is built from the star catalog. Then, the identification of stars becomes a trivial problem, that of finding the subgraph in the big graph of all stars of the catalog. This search can be performed using the triangle algorithm [Lie92], which is very time-consuming, because there are a multitude of triangle subgraphs within the catalog graph. However, some optimizations can be

applied to this exhaustive search process, such as eliminating some redundant triangles. Still, when adding noise of the data, such as false stars, these algorithms ultimately fail to preserve their precision, especially when the number of false stars increases.

Pattern matching algorithms create a unique pattern for each guiding reference star (GRS), which contains information from the neighboring stars (GNSs). Then, in order to detect stars, the pattern is searched through the ones in the catalog. The grid algorithm [PKD97] is the most widely used pattern-based approach, which identifies the correct part of the sky in 99.7% of the image orientations, where the accuracy of the positions of stars is 1 pixel, and the magnitude precision is 0.4 units of stellar magnitude. It transforms the celestial sphere into a grid, and then tries to match grid cells of the photo with grid cells of the celestial sphere (catalog), by testing some hypotheses, in which combinations of stars are put in the same pattern in order to try and ensemble a pattern that can be found in the catalog grid. However, this algorithm, even in papers where it was improved, is sensitive to the presence of ”false” stars.

Genetic algorithms [PSAWS06] were also used for the star identification task. They represent an alternative to the classical optimization problem algorithms that search for a global minimum (in this case, for the best fit to the image pattern inside the star catalog). In this approach, a random population of solutions (star patterns) is selected, they are evaluated, and the best of them will be ”parents” in the next generation, which essentially resembles an epoch from the machine learning terminology. The ”child” solution will have features from both of his parents, but also some mutations. This process is repeated, until a template that is close to one from the catalog is reached. However, their speed and lack of robustness evolve into lower accuracies when compared to other approaches, which will be presented below.

A paper [DCT+22] from 2022 focuses on Space Surveillance and Tracking, which implies monitoring space data to identify objects that orbit our planet. They used datasets from Space Surveillance Optical Telescopes, whose images are sometimes affected by light pollution or clouds, and a Detectron 2 architecture based on Faster R-CNN and ResNet-18, in order to classify between ”point” and ”streak” objects in the night sky. Next, the area of each object around the centroid point is computed, and, if under a specific threshold value, the object is discarded. When an object was only partially visible in an image, it was discarded because it was more probable for it to be fully visible in the next frame. The training process was done using K-fold cross-validation, and relabeling at each step, as not all objects are initially labeled. In the end, 98% precision is received for the second classification task that validates detected objects (as in the beginning the dataset is only partially annotated), and

0.62 mAP for object detection.

Gendared [PVS+21] (The Generic Data Reduction Framework for Space Surveillance), a software created in the context of an ESA Romanian Industry Incentive Scheme project, that receives raw telescope images and generates astrometric data for them using AI and traditional methods, was used in the previous paper as the "ground truth" source. The pipeline of this software consists of background elimination, detection of all objects, classification into object of interest, star or cosmic ray, and astrometry, which finds the position of some reference stars, and trajectory estimation of the detected objects. AI4GENDARED, a project led by GMV, is based on this software and involves improving it by using deep learning, for the classification task, and pattern recognition for completing data based on the orientation of the telescope.

In a 2020 paper by Rijlaarsdam et al. [RYB+20], the focus is on star identification in a lost-in-space context (without location information). The study proposes the use of a small neural network and feature extraction methods as an alternative to template matching algorithms that require a database of star characteristics. By utilizing relevant feature extraction, the neural network can focus on other characteristics of the stars. Traditional algorithms like the Grid Algorithm and distance-based approaches are prone to errors due to image noise or wrong star selection. In this study, normalized binned distances to the polestar are used as inputs for the neural network, which is trained to classify the data into different star classes. Stars that are too dim to be detected are eliminated from the classification task. For creating the training data, images are rotated, and results from picking different guide stars in different scenes are calculated, and also false stars are introduced. The used evaluation metric is the identification rate, which represents the number of correctly identified stars from the test set over the incorrect ones. Two different networks are used, one with 4096 nodes in both hidden layers, and one with 32 in the first layer and 64 in the second. This paper shows that even a small architecture can achieve great results. The study also investigates the impact of false stars and positional noise (shifts in the centroids of stars due to thermal deformations of lenses, or errors in the centroids' detection algorithm) on the classification performance, showing that the proposed approach remains robust and accurate under varying testing conditions. The results are shown in figures 3.4 and 3.5. However, in order to achieve good results, this algorithm is based on the assumption that a real guide star is present in the center of the image.

In a state-of-the-art approach presented in a paper [XJL19] from 2019, they compare a RPNet deep learning architecture with the grid algorithm presented before. RPNet consists of a star pattern generator (SPG), which generates star patterns and finds the one that creates the best separation of classes, and a star pattern classifier

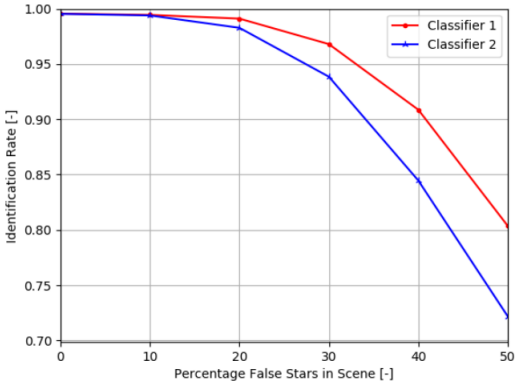


Figure 3.4: Comparison between the two classifiers when false stars are introduced [RYB+20]

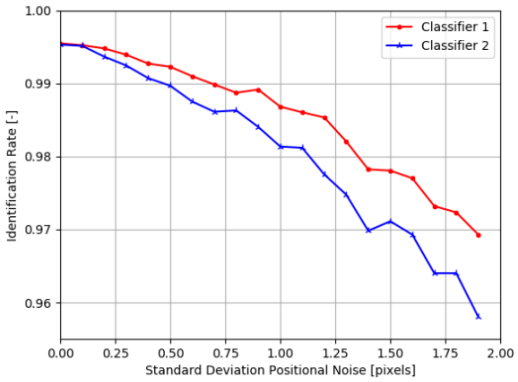


Figure 3.5: Comparison between the two classifiers when positional noise is introduced [RYB+20]

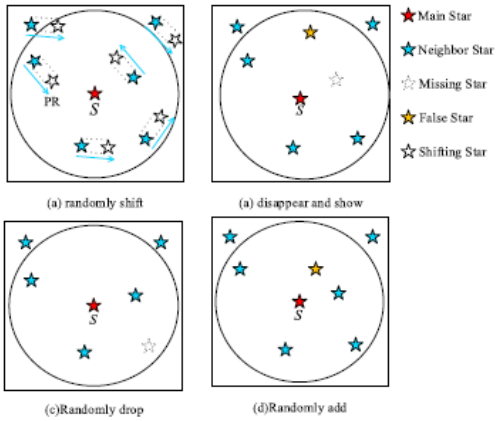


Figure 3.6: Different methods of adding noise to the data [XJL19]

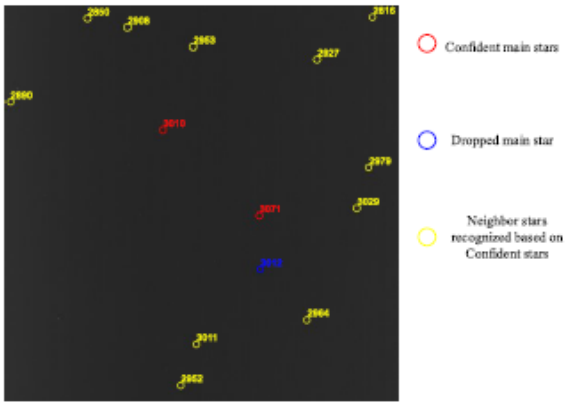


Figure 3.7: Results of RPNet on a real image [XJL19]

(SPC), which performs the classification task. The algorithm filters stars from the star catalog based on factors like observability and angular distance. It is based on pattern-based identification algorithms. First, the celestial coordinates (right ascension and declination) are transformed into image coordinates. Neighboring stars are chosen only from a fixed pattern radius limit of angular distance to the reference star. Noise (Figure 3.6) is also added to the training set, together with the corresponding clean data. The SPG has two fully connected layers, and encodes the chosen star patterns. SPC is a feedforward network that assigns probabilities to each star to be of a specific class. The training process consists of two phases: in the first, the SPG learns to generate clean patterns by learning from noisy and clean data. In the next stage, SPC trains on the output of the SPG in order to detect the stars. This algorithm achieves 99.23% accuracy in identifying stars from simulated images, while on real images the accuracy is still higher than 98%. The study also compares different al-

gorithms by noise level (figures 3.8, 3.9). Complexity is also evaluated: Table 3.1. Compared to the previously discussed paper, they report no performance decline in position changes up to 1 pixel standard deviation, while accuracy drops to 94% when reaching 1.6 pixels of positional noise. The results for a real star image can be seen in Figure 3.7.

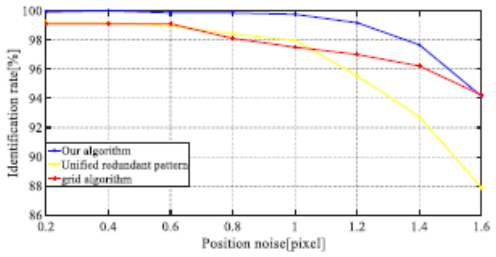


Figure 3.8: Comparison of models in different levels of position noise [XJL19]

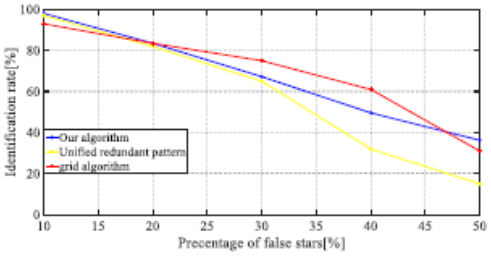


Figure 3.9: Comparison of models in different numbers of false stars [XJL19]

Type of complexity	Triangle algorithm	Grid algorithm	RPNet
Time complexity	O(n^3)	O(cN)	O(N)
Space complexity	O(n^3)	O(N)	O(1)

Table 3.1: Time and space complexities comparison [XJL19]

3.2.3 Applications of constellation detection

There are many available apps that help people recognize constellations in the night sky, such as Stellarium, SkySafari, Star Tracker, Skyview, Star Walk and many more. These enable the user to point the device at the night sky and see the constellations that lie above. However, these apps rely on pre-existent data, the location of the user and device sensors. They are not capable of identifying constellations from a single image, without location or orientation information, or stellar databases.

In addition, there is a website, Astrometry, which is documented in this article [LHM+10]. It enables users to input a night-sky image, and it will output the detected celestial objects and constellation, but also objects that are not visible in the image but are present in that field of view. The algorithm solves a generalized "lost-in-space" problem, as not even the image scale is known. Hashes are computed from star asterisms (four or five stars) and searched for in previously labeled hashes of star groups. The success rate is greater than 99.9%, and the error only comes from the incompleteness of the star catalog that is used. However, the search takes a lot of time, compared to a neural network approach.

Chapter 4

Methodology: Classification on the Bortle scale

4.1 Dataset

4.1.1 Data generation

Due to the unavailability of large datasets specifically designed for the light pollution classification task from night sky images, I created one on my own. I used a free open-source planetarium software called Stellarium [Ste23] [ZHW+21], version 1.1.

View settings

Images were generated using a perspective projection, as it presents the three-dimensional world as it would appear to an observer from a specific location. It maintains the correct relationships between shapes and angles, but distorts the sizes of objects depending on their distance from the observer [FvDFH13]. This was optimal for my task, since the distortions are minimal at the edges and because it can provide a more realistic and accurate view of the sky as it would appear to an observer from a specific location. On the contrary, other types of projection available in Stellarium, such as the Mercator projection, which is commonly used for maps of the Earth [Vis18], because it preserves shapes and orientation, or a stereographic projection, which is used in star charts [dC16], can cause significant distortions in the sizes of objects at different locations. In figure 4.1, these projections are simulated in Stellarium GUI.

I wanted the images to look as real as possible, in order to be able to generalize my model for a real-world application, so I checked some options in the GUI or in the script, such as:

Mercator projection Stereographic projection Perspective projection

Figure 4.1: Different types of projections in Stellarium GUI

- dynamic eye adaptation (in order to dim faint stars when a very bright object is visible, as this is how our eyes and cameras work)
- zodiacal light brightness
- Milky Way brightness
- twinkle effect on stars: a combined effect of refraction and interference of light, in turbulent atmosphere (especially near the horizon) [Ure82]
- elimination of the moon from images, as the glare effect highly depends on the camera specifications and could lead to wrong results if pictures are not taken with the same setup

I also changed the refraction settings. They consist of 3 customizable factors: pressure, temperature, and extinction coefficient. Temperature and pressure affect the refractive index of the atmosphere [Bru81], which causes the position of the celestial object to change (especially near the horizon). The extinction coefficient quantifies the amount of light that is absorbed or scattered when passing through the atmosphere [And12]. When this value gets higher, it means that the intensity of light coming from a celestial object decreases, thus making it harder to observe fainter objects. While this coefficient is not a direct measure of light pollution, it does have an impact on the visibility of celestial objects in polluted areas. A higher extinction coefficient can make it even more challenging to observe faint celestial objects in light-polluted areas, as it reduces the overall brightness of the sky and the contrast between the objects and the sky background. Also, particles and aerosols that contribute to the extinction coefficient can help scatter artificial light from the ground up, escalating the effects of light pollution even more. This phenomenon is called skyglow [KTB+15]. I used a pressure of 1013.00 mbar, $15\,^{\circ}\mathrm{C}$ temperature and an extinction coefficient of 0.13k. These values are standard, as I did not want them to have an effect on the already complex feature space of the data. Also, my observer's position was always oriented to the zenith, so the effects of refraction would have been minimal.

Movement of the observer

In order to generate images of the night sky in random locations from an observers' point of view, I set the movement of the celestial sphere to an alt-azimuth mount (Figure 4.2). It consists of a vertical axis of rotation (altitude) and a horizontal one (azimuth). The difference from an equatorial mount is that this type of mount cannot compensate for Earth's rotation, but my interest was only to generate random images of the night sky, not to track certain objects of the celestial sphere. However, to generate different positions, I used random values for equatorial coordinates (Figure 4.3), because for the constellation detection task (Chapter 5) it was much easier to work with these coordinates as they are the standard celestial coordinate system.

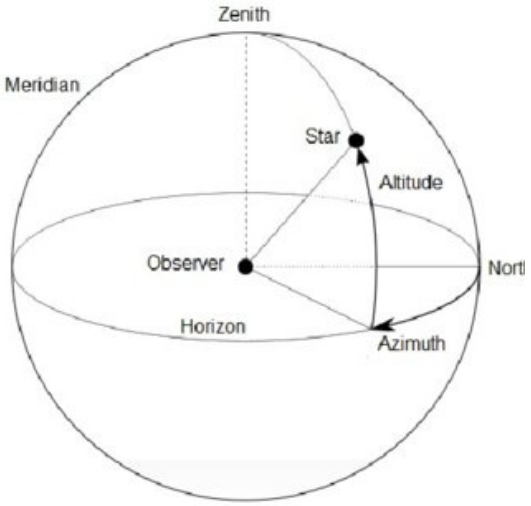


Figure 4.2: Horizontal coordinate system
[SCP15]

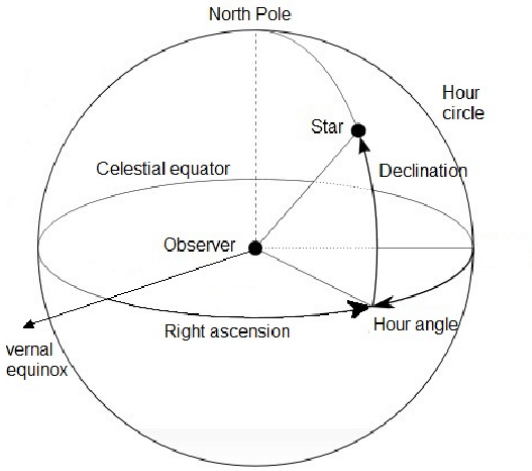


Figure 4.3: Equatorial coordinate system
[SCP15]

In a for loop, I generated random values that define the observer's position in an equatorial system:

- **Right Ascension (RA)**: a random value between $-180°$ and $180°$ (relative to the vernal equinox). This value is the equivalent of Earth's longitude, but for celestial objects. It is the angle formed between the hour circle (an imaginary circle on the celestial sphere passing through the celestial poles and a given object) of the object and the hour circle of the vernal equinox, measured in the opposite direction of the apparent diurnal motion [Ure82].
- **Declination (DEC)**: a random value between $-90°$ (south celestial pole) and $90°$ (north celestial pole). This is similar to Earth's latitude, and paired with RA they define the observer's position in the equatorial coordinate system.

It is the angle formed by the radius vector of the body with the plane of the equator [Ure82].

- **Rotation angle (THETA)**: a random value between $0°$ and $360°$. This represents the rotation angle of the observer's view around the zenith (the point directly overhead).
- **Field of View (FOV)**: a random value between $30°$ and $90°$. This determines how much of the sky is visible at once. It is measured in degrees, and higher FOV values correspond to a wider view of the sky [ZW19].

Next, I changed the observer's location to the new coordinates and set the altitude to the zenith to be able to capture strictly what is directly overhead.

Surface brightness settings for light pollution effect

Surface brightness is expressed in magnitudes per square arcsecond ($mag/arcsec^2$) and represents the amount of light in a given area of the sky [Śc13]. It is a key metric in classifying pollution on the Bortle scale, as it provides a quantitative measure of the sky's brightness. The most prominent negative effect of light pollution is artificial skyglow, which is caused by artificial light dispersed in the atmosphere, resulting in an increase in the brightness of the night sky [FCD+16]. Through the GUI, one is able to change the value of surface brightness, with a manually selected one (Figure 4.4) or automatically from Stellarium's locations' database (it contains light pollution information for several places).

I generated 3600 images of 12 different values of surface brightness that are depicted in table 4.1.

Table 4.1: Values of surface brightness per class of the generated images

	Very good [class 0]				Good [class 1]				Poor [class 2]			
Surface Brightness	22.2	21.9	21.7	21	20.2	19.3	20.6	19.5	18.8	18	17	18.3
Limiting magnitude	6.8	6.6	6.5	6.1	5.5	4.8	5.8	5	4.4	3.7	2.8	4

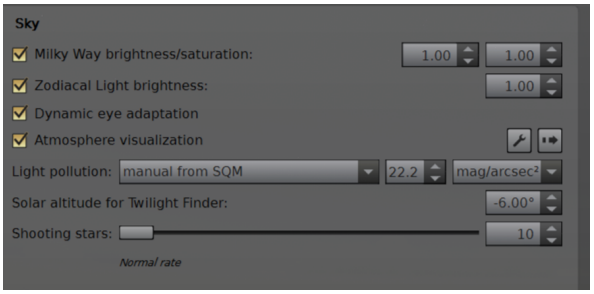


Figure 4.4: Manual selection of surface brightness from the GUI

4.1.2 Data analysis

Images overview

The dataset is balanced, as I generated 300 images for each of the aforementioned surface brightness values, resulting in a total of 3600 images. In figure 4.5 there are some example images, along with their corresponding surface brightness, naked eye-limiting magnitude and Bortle scale value. Magnitude is a logarithmic scale that defines the brightness of a celestial object. The lower the magnitude, the brighter the star. The naked-eye limiting magnitude represents a measure of the faintest object that can be seen with the unaided eye under a dark sky. Usually, humans can see until a magnitude of 6, but this depends on various weather conditions [Dic16].

22.2 / 6.8 / bortle1

21.0 / 6.1 / bortle4

19.5 / 5.0 / bortle6

18.0 / 3.7 / bortle8

Figure 4.5: Different pollution levels in dataset images

Mean intensity values. Separation by class

First, I started looking at the mean intensity values of the images. This value represents the average brightness of the image, so it is an indicator of the skyglow, of how much artificial light gets projected in the atmosphere from the ground. In this article [CFE00], they also correlate the brightness of the sky with the light pollution scale when drawing light pollution maps.

Initially, my dataset was split into 9 classes, each of them corresponding to a different Bortle scale value. I plotted the mean intensity value per class in a box plot depicted in figure 4.6. There are many outliers corresponding to the first five classes of the Bortle scale, and there is also no clear separation between the first four classes. By simply looking at the dataset images, it is very hard to tell the difference between the initial 4 classes, as different regions of the sky are depicted, with different celestial objects and with a different density of them.

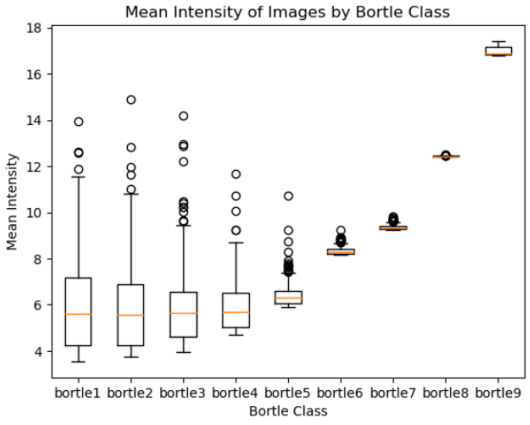


Figure 4.6: Mean intensity values for each Bortle class

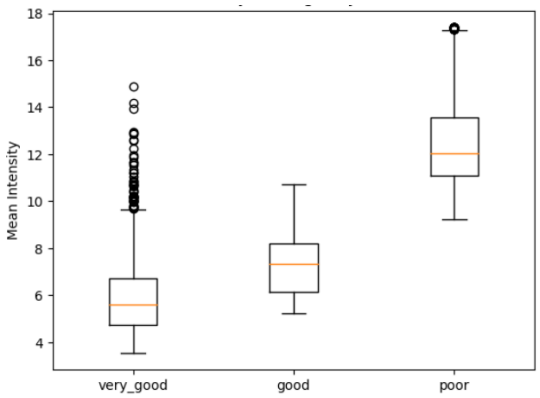


Figure 4.7: Mean intensity values for each new class

Thus, by also taking into account the fact that I wanted my model to generalize well on real data, and that Stellarium might not be able to accurately depict such fine differences between similar pollution levels, I tried to split my dataset just in 3 classes, and see if I can get a more solid separation from there. This approach was also followed in two related articles that were presented in chapter 3. I chose the following separation of classes representing the quality of the night sky, with regards to light pollution intensity:

- very good (naked-eye limiting magnitude > 6)
- good (naked-eye limiting magnitude between 4.5 and 6)
- poor (naked-eye limiting magnitude < 4.5)

I chose these values since the ”very good” class corresponds to dark, rural areas located more than 100 miles away from major cities, the ”good” class represents rural, suburban or small city areas, and the ”poor” class some bright, urban areas [Bor01]. I wanted to create a real-world application capable of adapting to different cameras and locations and still being able to differentiate between a highly polluted sky and a typical suburban area one. In figure 4.7 I plotted the mean intensity values for those new classes.

Now, the separation of classes is much clearer from the sky brightness point of view. However, the ”very good” class has many outliers, because the brightness of the stars and their density in the sky make the mean intensity grow larger, even though it is not an effect of the skyglow. Thus, I needed a more advanced model to extract features and be able to classify images in a correct way, making the distinction between a highly polluted, lightly colored sky, and a sky full of stars that also has a high mean intensity, but not coming from the background glow of the image.

Preprocessing

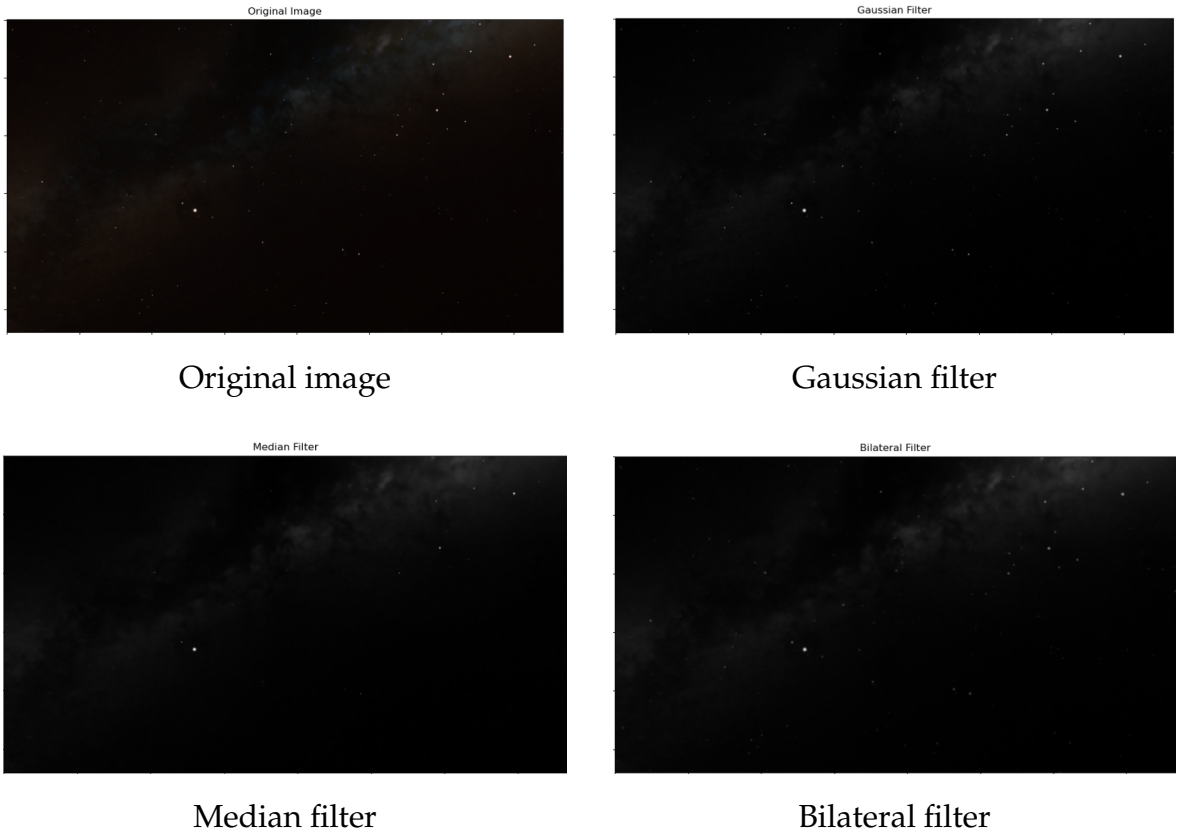
I also tried preprocessing the images with different filters placed on the normalized and grayscaled image, but when I did this, several features that could have been present, such as the Milky Way or different background hues, were suppressed.

The Gaussian filter, used to eliminate noise from images and smooth out details [BK08], considered the background color (which is a good indicator of light pollution) noise, so when filtering it changed it regardless of parameters.

The median filter, which replaces each pixel value with the median value of the neighboring pixels [BK08], blurred stars excessively, so when calculating the mean intensity afterward, the values got larger, but only because the stars were surrounded by a glare.

The bilateral filter, which should smooth the image while retaining edges [BK08], considered some of the dimmer stars to be noise and completely eliminated them when given a higher neighborhood value.

I wanted background and certain ”noise” information to remain preserved, as it is a key indicator of how polluted the sky is (if the Milky Way is visible, then it is clear that the sky is not polluted, and also different hues in the background are visible when the sky has a lower pollution factor). In figure 4.8 different filters are applied on a ”very good” class image. I tried different parameters, but it seemed that background information and fine details, such as the galaxy’s hue, were more and more suppressed. For the images in the figure I used a kernel of (5,5) for the Gaussian blur, a 5-sized kernel also for the median filter, and for the bilateral blur I used a neighborhood of 9 with 75 for the filter sigma color and space. The more I increased those parameters, the worse the effect got, and with lower values of the parameters I could not see such a significant difference that it would be worth using those filters in my classification task.



Original image | Gaussian filter

Median filter | Bilateral filter

Figure 4.8: Different filters applied on a ”very good” class image

4.2 Experiments using simple models on mean intensity values

In subsection 4.1.2 I illustrated how the mean intensity of images is a representative feature and has the potential to efficiently separate classes. I tried some simple models first to see how they are able to learn from this feature, before moving on with more advanced methods.

Note: For the following models in this section, I united the validation and test set, as there is no need for a validation set (these models do not have any hyperparameters to be tuned).

4.2.1 Model 1: Logistic regression

The scikit-learn logistic regression classifier also supports multiclass classification tasks by default. I used the one-vs-rest approach (OvR), in which for each class a separate (binary) logistic regression model is trained. For each image, it calculates the probability of that image being a specific class. The class with the highest probability is the resulting class of the input image [ld]. I obtained 78% accuracy on the test set. The confusion matrix is illustrated in figure 4.9. The ”poor” class is perfectly

classified. However, the model struggles with the distinction between the two other classes.

4.2.2 Model 2: SVM

I used the scikit-learn SVC (multiclass) implementation and tried to fit the data with multiple kernels: linear, polynomial, and rbf. In Figure 4.10, the confusion matrix for the rbf kernel is depicted. From the confusion matrices of logistic regression and SVM, it appears that SVM performs better on all classes except for the "very good" (label 0) class.

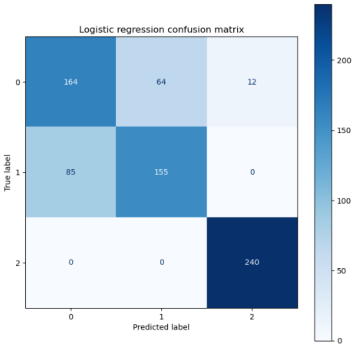


Figure 4.9: Logistic regression confusion matrix

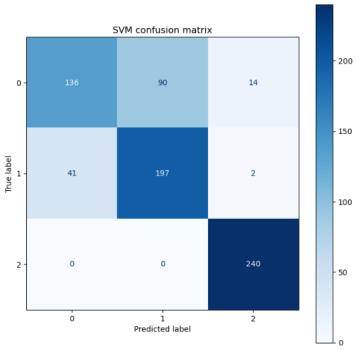


Figure 4.10: RBF kernel confusion matrix

Kernel type	Accuracy
Linear	78%
Poly (degree=3)	78%
RBF	80%

Table 4.2: Accuracy of different kernel types

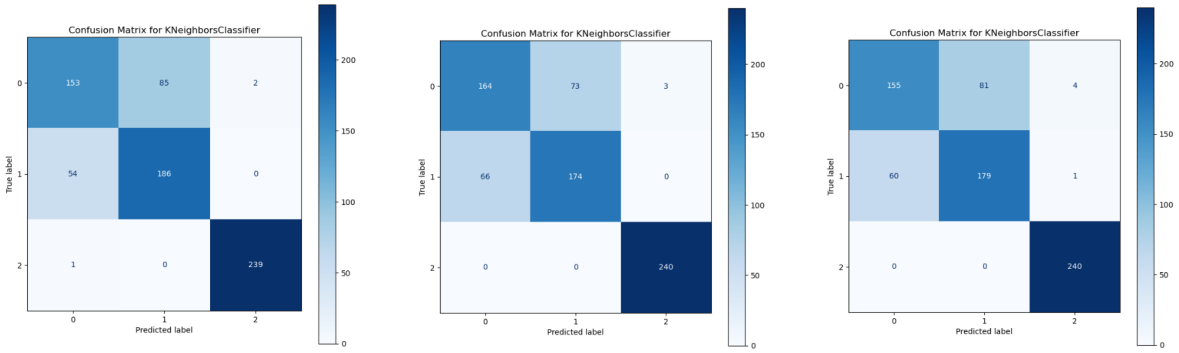
In table 4.2, I presented the results of the SVM classification using different kernels. The results are similar, with the RBF kernel being slightly more precise on the "good" (label 1) and "very good" (label 0) classes.

4.2.3 Model 3: KNN

I tried to apply the algorithm with k=5, k=10 and k=20 neighbors. For all of them I obtained 80% accuracy on the test set. However, from the confusion matrices (Figure 4.11) it appeared that for k = 10, the results are on average better compared to the other two values of k. Compared to rbf SVM, the KNN approach does not classify the "good" (1) class that well.

4.3 Experiments using deep learning models

All the models presented before achieved $\approx 80\%$ accuracy on the test set. Outliers identified earlier (Subsection 4.1.2) proved to prevent simple models from achieving higher accuracy. My initial approach was to try out different state-of-the-art or fast and efficient models on the raw images, in order to see if they are able to detect



k=5 k=10 k=20

Figure 4.11: KNN confusion matrices

the pollution-specific features. To gain more insight into the performance of these CNN models, I created a new test set of 1200 images, 400 for each class, with the same distribution of pollution levels as the one used in the initial dataset, presented in Table 4.1. I didn't use any preprocessing on the images, except for resizing them to the sizes the models were pre-trained on, in order to be able to use transfer learning (utilizing the learned knowledge and weights from a pre-trained model on a different dataset as a starting point for training on a new dataset), and the default preprocessing of the model, if this was the case.

All models used transfer learning from ImageNet [DDS+09a] and were trained for 100 epochs. Also, I used Adam optimizer and a dropout layer, in order to prevent overfitting.

The confusion matrices on the test set and the history of the training processes for all models were plotted. They are present in Appendix C. A summary of the results is presented in Table 4.3.

Model	**Accuracy**	**Correctly predicted "good" class (out of 400)**	**Correctly predicted "poor" class (out of 400)**	**Correctly predicted "very good" class (out of 400)**	**Train loss**	**Validation loss**
MobileNetV2	73%	208	361	304	0.3745	0.3741
EfficientNetB7	82%	323	331	327	0.3395	0.2751
ResNet50V2	79%	322	327	300	0.2827	0.2971
YOLOv8	99%	397	400	392	0.04458	0.5262
InceptionV3	77%	274	306	347	0.1887	0.1421

Table 4.3: Summary of results for CNN models

4.3.1 Discussion of results: interpretation and analysis

MobileNetV2 struggles to distinguish the difference between the middle "good" class and the other two classes. However, it is very good at spotting the high levels of pollution found in the "poor" class images. **EfficientNetB7** performed much better than MobileNetV2, as from the confusion matrices it appears that EfficientNetB7 recognizes the "good" middle class much better and the "very good" class a little better, while it is worse at identifying "poor" class images. Also, the validation loss is slightly smaller for this model. An important note is that the accuracy on the validation set is slightly higher almost every time compared to the train accuracy. Probably in the validation set there were images on which this model performed better, and more edge cases appeared in the training set, thus making the performance on the training set worse. Having a bigger validation set would have made it more probable to have a lower validation accuracy. However, the accuracy on the test set is still high, so the model is not overfitting and there is not any concerning noise in the data. Compared to previous models (without taking into account YOLOv8), **ResNet50V2** has consistently good results for each class. Furthermore, the loss values are better than those of MobileNetV2 and comparable to those of EfficientNetB7. The loss obtained on the train and the validation sets for **InceptionV3** is the lowest of all models, while this model also has decent results in all classes.

From the table, it is clear that **YOLOv8** is the most efficient model. However, the loss values were higher compared to the other models (0.52 on the validation set). The model clearly still performs very well on the test set, so I considered it in the following steps. The results are almost perfect, except for the "good" middle class. However, even if the results are so good, the big loss value isn't a good indicator of the model's capability of generalization on a real dataset.

4.4 Feature engineering

My third approach was to use feature extraction to keep only relevant features from the input image and classify based on them.

4.4.1 Feature extraction using deep neural networks

I chose to use some powerful CNNs for this specific task, as most feature extractors require the image to be converted to grayscale, thus losing color information. I am especially interested in background color, so the best solution was to use deep neural networks capable of extracting high-level information from images [ZC18].

For analyzing the quality of the extracted features, I tested them on the "sim-

ple" models presented in Section 4.2, which proved to be efficient for classifying the images based on their mean intensities.

Before testing the relevance of CNN-extracted features in simple models, I selected the most relevant using different selection strategies presented in this book [KJ20].

4.4.2 Feature selection

I applied the random forest/extra trees feature selector using the SelectFromModel scikit-learn class [PVG+]. I used the "mean" and "median" thresholds, which mean that only features with a higher impact on the purity of nodes than the mean / median importance of all features will be selected. I wanted to try both approaches since my data has some outliers and I wanted to also account for their presence.

With each of the CNNs, I extracted the 50 most relevant features and then selected from them. All models used transfer learning from ImageNet[DDS+09b]. Tables 4.4 - 4.7 show the accuracies with different feature extractors and different feature selection techniques.

Random forest selector	**Linear SVM**	**Logreg**	**KNN(n=5)**	**Random forest(n=100)**
no feature selection	91%	90%	87%	96%
Random forest - mean	92%	92%	72%	94%
Random forest - median	92%	91%	73%	92%
Extra trees - mean	92%	90%	73%	93%
Extra trees - median	92%	91%	73%	92%

Table 4.4: VGG16 feature extraction results

Feature selector	**Linear SVM**	**Logreg**	**KNN(n=5)**	**Random forest(n=100)**
no feature selection	91%	89%	76%	83%
Random forest - mean	89%	89%	77%	82%
Random forest - median	91%	89%	76%	81%
Extra trees - mean	90%	89%	76%	83%
Extra trees - median	91%	89%	76%	81%

Table 4.5: MobileNetV2 feature extraction results

4.4.3 Discussion of results: interpretation and analysis

From the results, it appears that the best feature extraction model is ResNet50V2. It achieved the highest accuracies with different classification models and selection techniques. The most promising results were obtained using mean threshold feature

Feature selector	Linear SVM	Logreg	KNN(n=5)	Random forest(n=100)
no feature selection	96%	96%	79%	89%
Random forest - mean	97%	96%	81%	90%
Random forest - median	96%	96%	79%	89%
Extra trees - mean	97%	96%	83%	90%
Extra trees - median	96%	96%	79%	89%

Table 4.6: ResNet50V2 feature extraction results

Feature selector	Linear SVM	Logreg	KNN(n=5)	Random forest(n=100)
no feature selection	88%	88%	84%	89%
Random forest - mean	95%	96%	83%	92%
Random forest - median	95%	95%	82%	90%
Extra trees - mean	95%	95%	84%	91%
Extra trees - median	96%	96%	79%	89%

Table 4.7: InceptionV3 feature extraction results

selection with random forest or extremely randomized trees, with a linear SVM classifier. However, given the small size of the test set, other similar performing models are not to be negligible (i.e., InceptionV3 feature extraction using median selection by extra trees and a linear SVM classifier), as they could generalize better on real data or a bigger test set. This approach performed way better than most CNN models on raw data, achieving accuracies close to the YOLOv8 model's performance.

4.5 Ensembling

Until now, only the YOLOv8 model was able to spot the differences very accurately between the 3 classes. However, the other models also show potential, so I decided to try and ensemble them in order to obtain another good-performing model, to be able to have different accurate systems when testing on real images.

4.5.1 Ensembling of CNN models

Hard voting classifier

First, I combined all pretrained CNN models presented in section 4.3 (except YOLOv8, which showed almost perfect results) into a hard voting classifier. Figure 4.12 shows a diagram of the ensemble voting classifier that I used.

On the same test set the models were trained before, I achieved 80% accuracy with this voting classifier. The confusion matrix on that test set for the hard voting classifier is presented in figure 4.13.

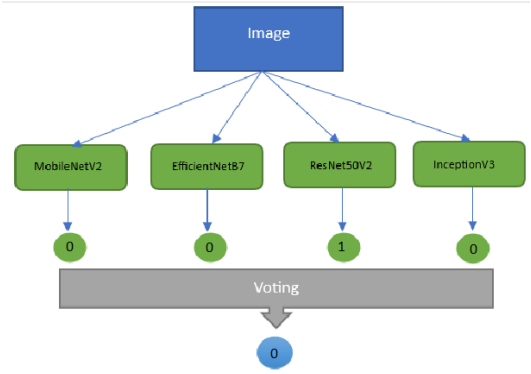


Figure 4.12: Voting classifier of CNN models

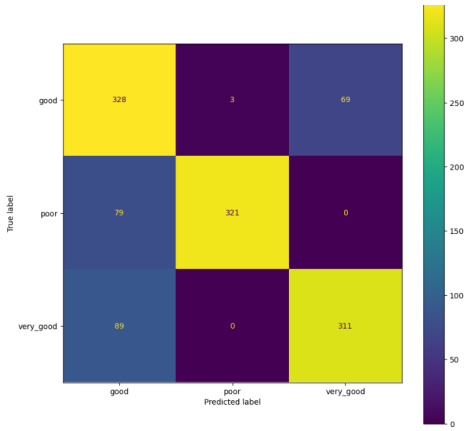


Figure 4.13: Hard voting ensemble: confusion matrix

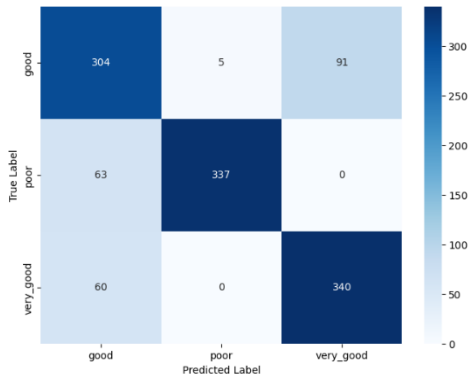


Figure 4.14: Soft voting classifier - confusion matrix

Soft voting classifier

On the same test set as the one the models were evaluated on in Section 4.3, the soft voting classifier achieved 81.75% accuracy. The confusion matrix on that test set for the ensemble system is presented in figure 4.14.

Stacking

I used 3 different meta-learners: AdaBoost, Gradient boost, XGBoost, and Random forest. I used two variants: in the first one, the meta-learners used the predicted classes of the base learners in order to make the decision, while in the second variant, the meta-learners used the probabilities outputed by each of the base learners for each of the classses, making the stacking more robust. The confusion matrices are illustrated in Appendix D. The results are summarized in table 4.8.

Discussion of results: interpretation and analysis

It seems that the stacking ensembling approaches work better than the voting ensembles. And also, generally, the second variant works better than the first, since

Meta-learner	Variant	Accuracy	Correctly predicted ”very good” class (out of 400)	Correctly predicted ”good” class (out of 400)	Correctly predicted ”poor” class (out of 400)
AdaBoost	2	80.42%	323	377	265
Gradient boost	1	83.33%	354	354	292
Gradient boost	2	84.75%	342	378	297
XGBoost	1	83.33%	354	354	292
XGBoost	2	86%	350	379	303
Random forest	1	83.33%	354	354	292
Random forest	2	85.08%	353	374	294

Table 4.8: Summary of results for stacking of CNN models

it has more robust data for the meta-learner to learn from (probabilities for each class instead of just predictions). The best approach seems to be the one that uses an XGBoost classifier as a meta-learner on probabilities generated by each of the base models. It is an increased accuracy compared to the ones of the base models alone. An interesting fact is that when based on the predictions alone, most meta-learners yield the same results, probably because they determine the same correlations between the simple data, that is, predictions of the base models, and the true class values. Also, when I tried the first variant with AdaBoost as a meta-learner, it confused the poor with the very good class, not being able to discern between the different characteristics and model decisions of the ”poor” class images.

4.5.2 Ensembling of feature extraction models

The ”simple” models trained in Section 4.4 showed a lot of potential given their relatively high accuracies. Thus, I wanted to try and assemble them in order to construct a more accurate system. I used the features extracted by the ResNet50V2 neural network and selected by extra trees with mean threshold (see Table 4.6). The confusion matrices are presented in Appendix E. A summary of the results is presented in table 4.9.

Ensemble method	Accuracy	Correctly predicted "very good" class (out of 400)	Correctly predicted "good" class (out of 400)	Correctly predicted "poor" class (out of 400)
Hard voting	86%	281	362	388
Soft voting	87%	281	369	392
Stacking: XGBoost meta	90%	310	377	395
Stacking: AdaBoost meta	89%	288	379	395
Stacking: Random forest meta	86%	281	361	388
Stacking: Gradient boost meta	86%	280	360	391

Table 4.9: Summary of results for stacking of models that were trained on extracted and selected features

Discussion of results: interpretation and analysis

From the results it is clear that ensembling the models that were trained on extracted features yielded better results than the ensembling of CNN models, however this is primarily due to the greater accuracies of the first type of models. The best results come from stacking with an XGBoost meta-learner (90% accuracy and general good results for all classes). However, compared to the previous ensembling experiment, in this one the accuracies of the models are generally not beaten. Probably, the models that were trained on the extracted features have very different strategies and results for this classification task, so combining them makes matters worse than relying on just a single model. This does not come as a surprise, as it is common for single well-performing models to be better than an ensemble of them, because by combining different outputs sometimes the complexity is too high and might not be fit for general purposes.

4.6 Testing on real data

I tried to use my best-performing model (YOLOv8) on real data. I tested only on a few images, but the results are promising. In figures 4.15 and 4.16 I showed two samples from the real world that I tested on.

Figure 4.15: ”Very good” real image: taken near Bacau, Romania

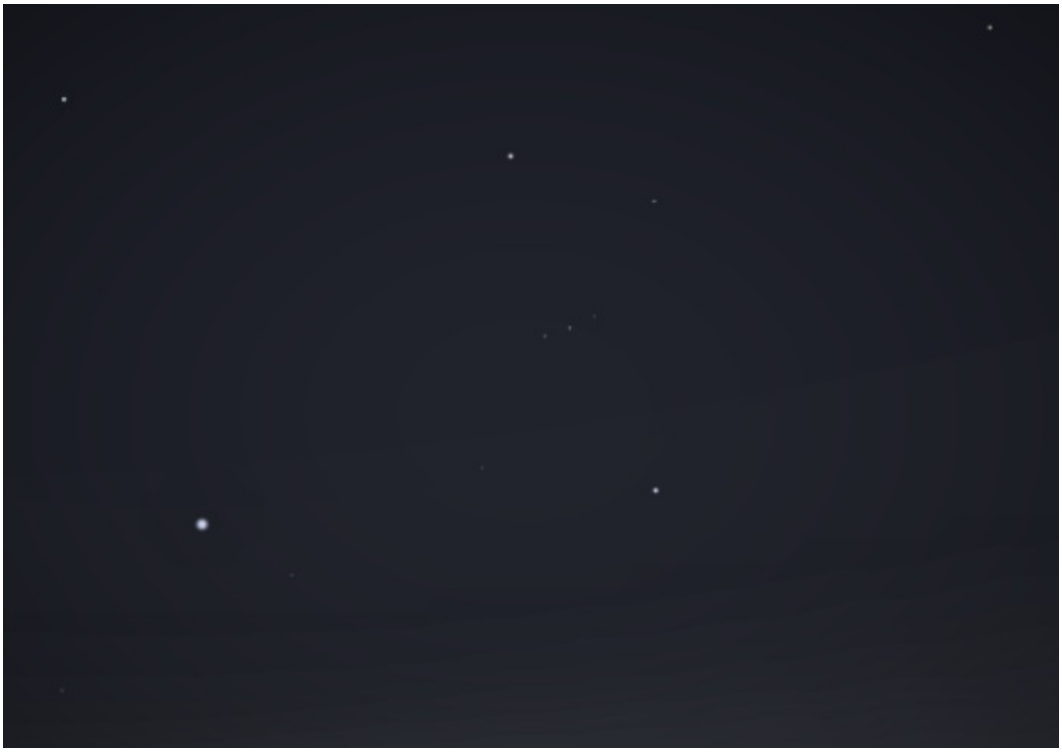


Figure 4.16: ”Poor” real image
source: https://explainingscience.org/

Chapter 5

Methodology: Constellation detection

5.1 Dataset

5.1.1 Image generation

For the task of constellation detection in night sky images, I also used the Stellarium planetarium software, version 1.1. I used the same view settings (Section 4.1.1), observer movement algorithm (Section 4.1.1) and surface brightness values (Section 4.1.1) as presented in chapter 7. An important thing to note here is that the observer was placed in both hemispheres, so images contain both southern and northern constellations. I generated a total of 24000 images, 2000 for each surface brightness value from table 4.1.

5.1.2 Automatic bounding boxes generation

I changed some things in the Stellarium script, as well as to output bounding box coordinates in YOLOv5 format, which will prove necessary in the object detection task. A good starting point was the code from [Ram20], originally used to calculate the percentage of constellations in an image.

First, I created a list of stars from each of the 88 constellations, along with their equatorial coordinates (Figure 4.3) and magnitude. I decided to only fit a constellation inside a bounding box if the following conditions were satisfied:

- the constellation is fully present on the sky (not necessary all stars from my list for that constellation are visible, but theoretically they are in the image plane)
- at least 4 stars are visible, from the naked-eye limiting magnitude point of view (they can be seen in the image). Otherwise, it would be very hard to discern one constellation from another, especially because there are different levels of pollution in my dataset.

Below, Orion is depicted as seen on a clear sky (figure 5.1) and on a heavily polluted sky (figure 5.2).

Figure 5.1: Orion (naked-eye limiting magnitude 6.5)

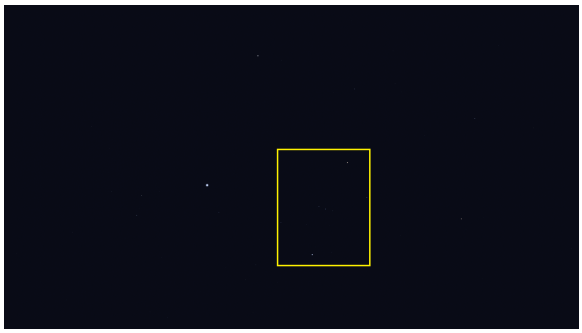


Figure 5.2: Orion (naked-eye limiting magnitude 3.7)

In order to compute bounding boxes for constellations that satisfy the two conditions, I took the following steps:

- First, I projected the position of each star in my list for each constellation onto the image (as seen here [Ram20]). The angular distance is calculated between the observer's point of view projected onto the surface celestial sphere and the star. Let us denote the equatorial coordinates (right ascension, declination) of the observer(R_1) by (α_1,δ_1) and for the star(R_2) by (α_2, δ_2). Then, the angular distance is calculated as the following (using the spherical law of cosines in triangle PR_1R_2): $\cos\Delta = -\cos(90-\delta_1)\cdot\cos(90-\delta_2) + \sin(90-\delta_1)\cdot\sin(90-\delta_2)\cdot\cos(\alpha_1-\alpha_2)$. In figure 5.3 there is an illustration of the celestial sphere.

- Using the spherical law of cosines in the same triangle, we can write the law of cosines for PR_1: $\cos(90-\delta_1) = -\cos(90-\delta_2)\cdot\cos(\Delta)+\sin(90-\delta_2)\cdot\sin(\Delta)\cdot\cos(R_2)$. From here, we can extract angle R_2 as: $\cos(R_2) = \frac{\cos(90-\delta_1)+cos(90-\delta_2)\cdot\cos(\Delta)}{\sin(90-\delta_2)\cdot\sin(\Delta)}$

- Using the value of Δ and R_2, we can determine the star's position relative to the observer on the celestial sphere. To represent the star's position on a 2D plane (in this case, an image), I used the gnomonic projection. The gnomonic projection is particularly useful because it maps great circles (shortest paths between two points on a sphere) to straight lines on a 2D plane, making it a popular choice for navigation purposes [Sny87]. The polar coordinates of the star in the tangent (gnomonic) plane will be the radial distance $x = \tan(\Delta)$ and the angle $y = \Theta - R_2$ (relative to the rotation angle of the observer's view around the zenith: see 4.1.1).

- Now that we have the polar coordinates of the star in the gnomonic plane, we convert them to cartesian coordinates in the image plane:$x_c = x\cdot\sin(y)$,

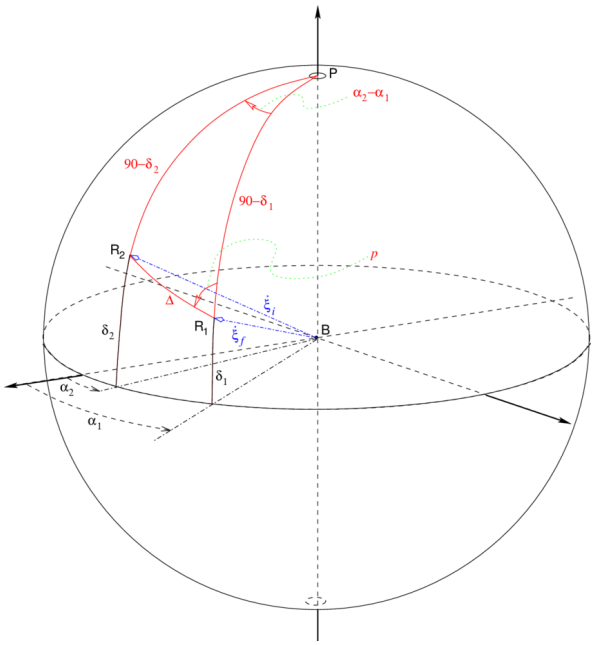


Figure 5.3: Angular distance between observer's projection on the celestial sphere and the star

[FCV+04]

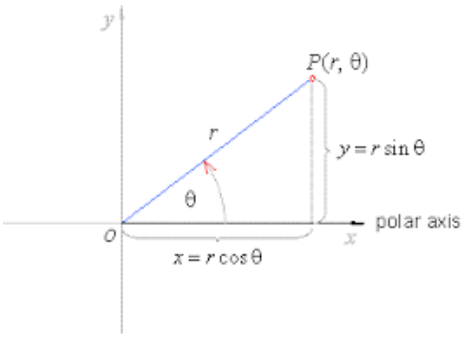


Figure 5.4: Polar to cartesian coordinates

[Ser]

$y_c = x \cdot \cos(y)$ (see figure 5.4). Then, we check to see if (x_c,y_c) coordinates of each star of each constellation are in the visible area (we compare x and y to X and Y, the "dimensions" of the image). By doing this, we will know if a star is present in the image area or not.

- For each star, I also checked to see if the magnitude of the star is lower than the naked eye limiting magnitude imposed by the pollution factor (Table 4.1), in order to know if the star will appear in the generated image or not.
- Next, for each constellation that has all stars in the image plane and at least 4 of them are visible (by order of magnitude), the bounding box coordinates were computed. The steps are described below:
  - First, for each visible constellation (by the two conditions imposed earlier), I initialize 4 variables, corresponding to the minimum and maximum values of x_c and y_c for each star by $-\infty$, ∞ respectively.
  - Then, I iterate through each of the visible stars of that constellation and compute their normalized Cartesian coordinates (x_c, y_c) in the image plane (I normalize them between 0 and 1 because this is required by the YOLO bounding box format.
  - I update x_{min}, x_{max}, y_{min}, y_{max} accordingly. This will, in the end, become the "edge" points of that constellation

  - When finished with all the visible stars of that constellation, I compute the coordinates of the center point of the bounding box for that constellation: $center_x = \frac{x_{min}+x_{max}}{2}$, $center_y = \frac{y_{min}+y_{max}}{2}$.
  - Now, I compute the width and height of the bounding box: $width = x_{max} - x_{min}$, $height = y_{max} - y_{min}$.
  - In the end, for each constellation I return the YOLO formatted bounding box coordinates: [1 - $center_x$, 1 - $center_y$, width, height]

- After each generated image, the script writes to file a line for each detected constellation, containing the class label (numbers from 0 to 87, corresponding to the 88 constellations) and the bounding box coordinates generated above.

In figure 5.5, the generated bounding boxes for an image are depicted. I changed the settings to also show the constellation lines and labels for demonstration purposes. The bounding box file of the computed coordinates is presented in Table 5.1 for reference. I changed the constellation class labels from numbers to names for clarity.

Table 5.1: Table caption goes here.

Constellation	$1 - center_x$	$1 - center_y$	**width**	**height**
ORION	0.489	0.636	0.116	0.169
CANIS MAJOR	0.683	0.732	0.135	0.180
LEPUS	0.591	0.573	0.051	0.147
COLUMBA	0.727	0.568	0.080	0.124
TAURUS	0.334	0.541	0.086	0.380
AURIGA	0.211	0.808	0.185	0.263

Figure 5.5: Generated bounding boxes for visible constellations

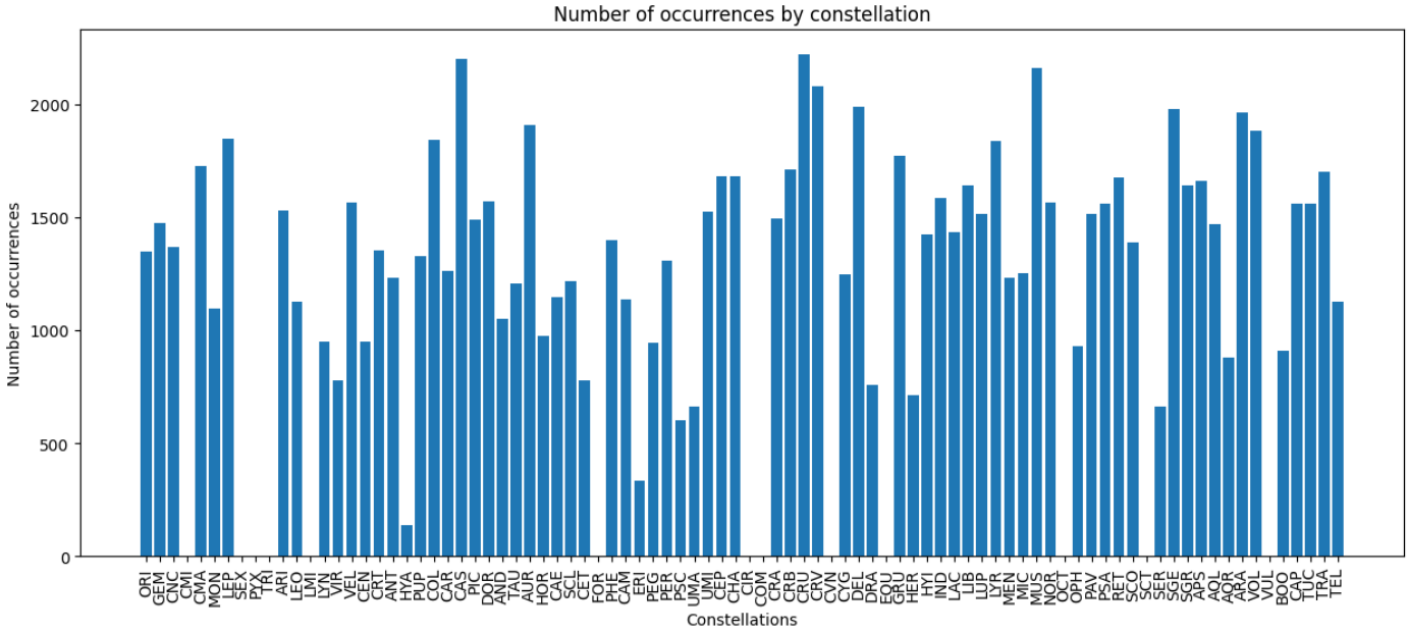


Figure 5.6: Number of instances per constellation class

5.1.3 Data exploration

As mentioned before, the set of 24000 images contains 88 classes corresponding to all existing constellations. The number of instances per each constellation class is depicted in Figure 5.6. Constellations' 3-letter abbreviations were taken from this source, which conforms to IAU (International Astronomical Union) format: [Bak95]. Out of the 24000 images, 3302 don't contain any constellations (by the two conditions mentioned in subsection 5.1.2).

From the plot, one can see that there are ≈ 10 constellations with no instances at all. These constellations have dim or few stars (less than 4 that could be visible from the observer), so no generated images contained them. I found this to be better for the general noise in the dataset, as there really isn't a way in which YOLOv5 could recognize a constellation from just 2 or 3 stars, especially because the constellations appear in different sizes (depending on the position on the celestial sphere and the distance to horizon), with different pollution levels, and also during augmentation multiple zooming steps are applied to the images, which I believe is a greater benefit to the general task rather than inconsistently trying to predict such constellations.

5.2 Image preprocessing

Being only interested in the detection of constellations, I wanted to remove as much background noise as possible, while enhancing the stars and eliminating several effects, such as star twinkle. I wanted to try at best to eliminate any information

about light pollution from images, other than the number of visible stars. The prerequisite steps consisted in normalizing pixel values and turning the image into grayscale.

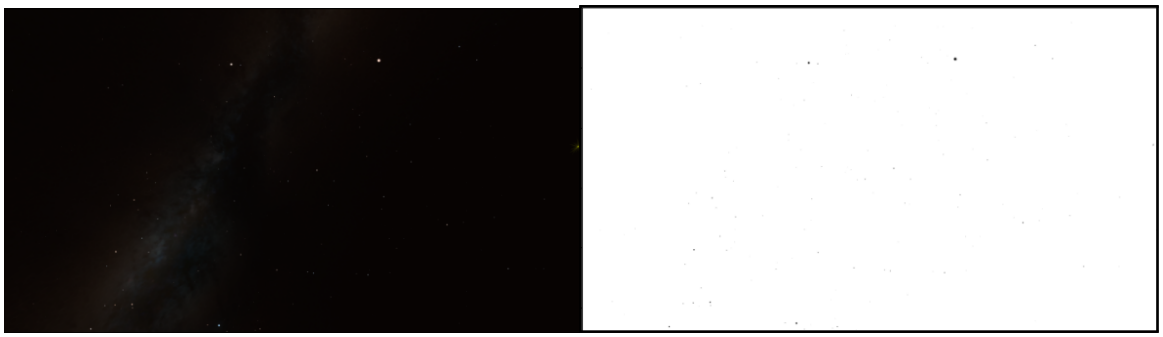
5.2.1 Contrast stretching

I wanted to be able to separate stars from the background as cleanly as possible. Luckily enough, stars are always brighter than the background sky, regardless of the pollution level.

Contrast stretching is a technique by which pixel intensity values are spread in a range from 0 to 255, in order to better illustrate the difference in brightness values. This OpenCV implementation that I followed uses two adjustable parameters [Ope23]:

- α: this represents the value by which pixel intensities are multiplied (in order for their values to be dispersed). A value greater than 1 signifies an increase in contrast in the image.
- β: after multiplying the pixel intensities by alpha, the beta term is added to each pixel, in order to shift the values, and thus spread them even more.

I illustrated the contrast stretching transform with $\alpha = 2$, $\beta = 30$ on an image: Figure 5.7. Unfortunately, after applying contrast stretching, the halo effect surrounding some stars remains (from the simulated atmospheric effect, pollution level, contrast between the star and its surroundings, etc.). Thus, an additional transform was needed to eliminate this sort of background noise.



Original "very good" class image Contrast stretched image

Figure 5.7: Contrast stretching

5.2.2 Image filtering

Box filter

The first image filtering technique that I tried was a simple averaging filter. This works by replacing the value of each pixel with the average value of its neighboring pixels, within a kernel. The kernel contains identical values that sum up to 1 (they are normalized to the size of the kernel). This filter is used to reduce noise and smooth out details in an image. A visual representation of how kernel averaging works is illustrated in Figure 5.8. The kernel that I used is represented in figure 5.9. From the resulting images it is clear that the haze effect of stars was a little bit

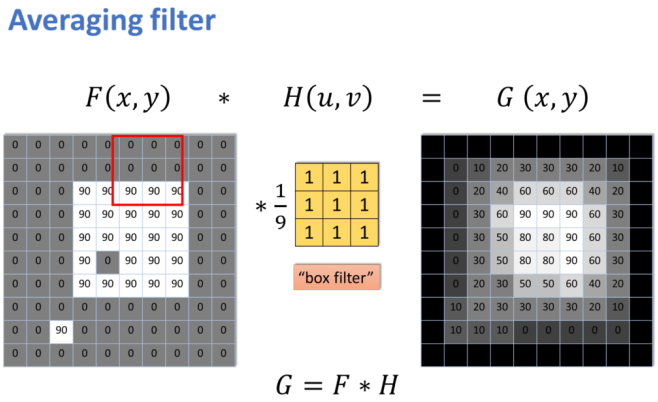


Figure 5.8: Averaging filter [Tea23]

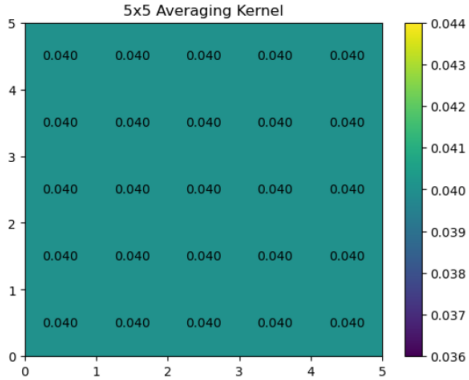


Figure 5.9: Averaging kernel

ameliorated, however I wanted to also try out and see how other filtering techniques behave on this dataset.

Gaussian blur

This is a type of convolution and frequency domain filtering that aims to smooth content in images, by eliminating high-frequency values. It applies convolution on the image with a Gaussian kernel computed from the Gaussian function. Basically, each cell of the kernel defined by the coordinates (x, y) gets the following value, normalized so that the sum of all cells inside the kernel is 1: $f(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$ [GW02]. σ represents the standard deviation of the distribution. A larger value of σ will result in more aggressive smoothing, while a smaller sigma value will result in less aggressive smoothing. This parameter is important in determining the trade-off between preserving image details and reducing noise.

The kernel utilized in this example is shown in Figure 5.10. As the distance from the central pixel increases, the influence of surrounding pixels in the kernel decreases. The Gaussian kernel is slid across the image, with a different pixel at the

center of interest each time. Subsequently, the value of the central pixel is replaced with the result of the kernel's convolution with neighboring pixels, thereby smoothing edges and reducing noise. Figure 5.11 shows the Gaussian function with varying values of σ and μ. In this case, σ was set to 1.1. While the images are less affected by

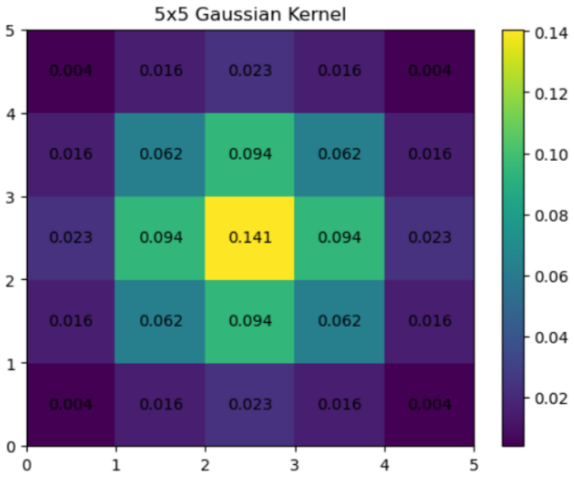


Figure 5.10: Gaussian kernel

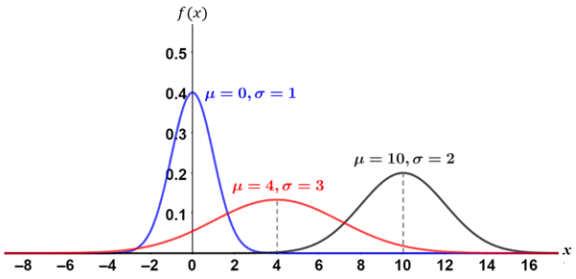


Figure 5.11: Gaussian function with different σ values [mat23]

the haze, further improvements are necessary. I observed that optimal parameters for contrast stretching and Gaussian/average blur depend heavily on the level of pollution in the image. Furthermore, the kernel size that produces optimal results varies even among "very good" class images based on whether they show the Milky Way or not. Therefore, to optimize the constellation detection task, it was necessary to find a way to automatically set these parameters.

5.3 Object detection algorithm

5.3.1 Custom preprocessing layer

In order to preprocess the image for constellation detection, I added a custom layer at the beginning of the YOLOv5 architecture to calculate the best α, β (from contrast stretching - section 5.2.1), and kernel size for image filtering, in order to be able to get the best noise reduction and star enhancement, regardless of the pollution level in the images. In the object detection process I tested with both Gaussian blur and average filtering kernels in the custom preprocessing layer, to see how the model behaves, and if there is any performance increase when using one instead of the other.

I used the Pytorch nn.Parameter class, which enables the parameter to be learnable (updated during the training process through backpropagation and optimization) [PyT23]. I initialized the learnable parameters with the following values: $\alpha =$

2, $\beta = 30$ and kernel size = 5.

I created a custom layer class, which inherits from nn.Module (base class for all neural network modules in Pytorch [PyT21b]). In the forward function, I implemented the following preprocessing steps:

- converted the image to grayscale (result denoted by "gray")
- computed the mean intensity of the pixels in the image, for the parameters to learn from this value (result denoted by "mean_intensity")
- adjusted the brightness of the image using contrast stretching, by the following formula: adjusted = gray · self.alpha + self.beta · mean_intensity. I multiplied β by the mean intensity of the image, as I wanted the change in the pixel values (brightness control) to be relative to the intensity of the image (in order to better adapt to the level of pollution in the image).
- computed the kernel (if the learnable kernel size was an even value, I added one to make it odd-valued, in order to always have a "center" pixel in the kernel)
  - for averaging filtering: I created an identity matrix on the same device where the x tensor is on, and then divided each value by the squared kernel size learnable parameter. I reshaped it to a 4D tensor [1, 1, kernel_size, kernel_size], to be compatible with the nn.functional.conv2d function input kernel [PyT21a]. I also used a padding of kernel_size / 2, to ensure that the output image has the same dimensions as the input image.
  - for Gaussian blur: the steps are similar, except for the computation of the kernel values.
- I transformed the converted image back to 3-channel format, in order to be compatible with the first layer of YOLOv5

5.4 Training

After concatenating the custom preprocessing layer at the beginning of YOLOv5s model architecture, I first trained the data for 200 epochs using the box filter as presented in subsection 5.2.2, and then for another 100 epochs using the Gaussian filter (subsection 5.2.2). The training evolution of several metrics for the last 100 epochs is presented in figure 5.12, while the metrics for the first 200 epochs are in Appendix F. I trained the data for 100 epochs at a time due to computational power reasons. During YOLOv5 training, simple mosaic augmentation techniques are applied by

default (zooms, flips). When an image is loaded for training, another 3 images are augmented from that image. This is especially beneficial for the constellation detection task, as it helps generalize my data while keeping the true nature of the objects. YOLOv5s was trained on 640-pixel images, so this is also the size that I used for training, on the recommendation of the online documentation of the software [JCS+22] that this is the best starting size. I used a batch size equal to 16 while training, so in each epoch, from the total of 19200 train images, 1200 were picked. At the end of each epoch, precision is tested using a batch of 75 images from the validation set. In the graph 5.13 I showed the strong correlation between the number of instances and mAP50 for each of the constellations from the validation set at the end of the training process.

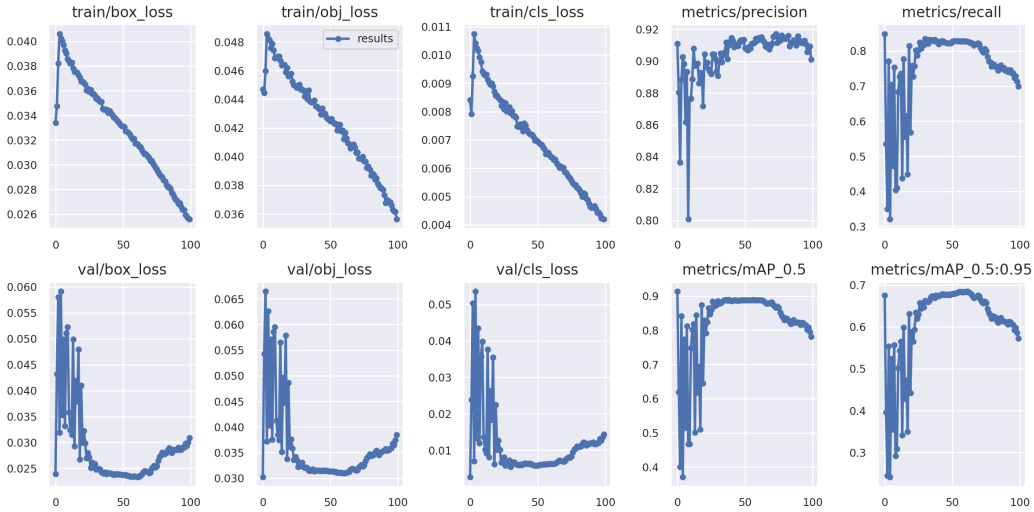


Figure 5.12: Epochs 200-300 (with best weights from 100-200)

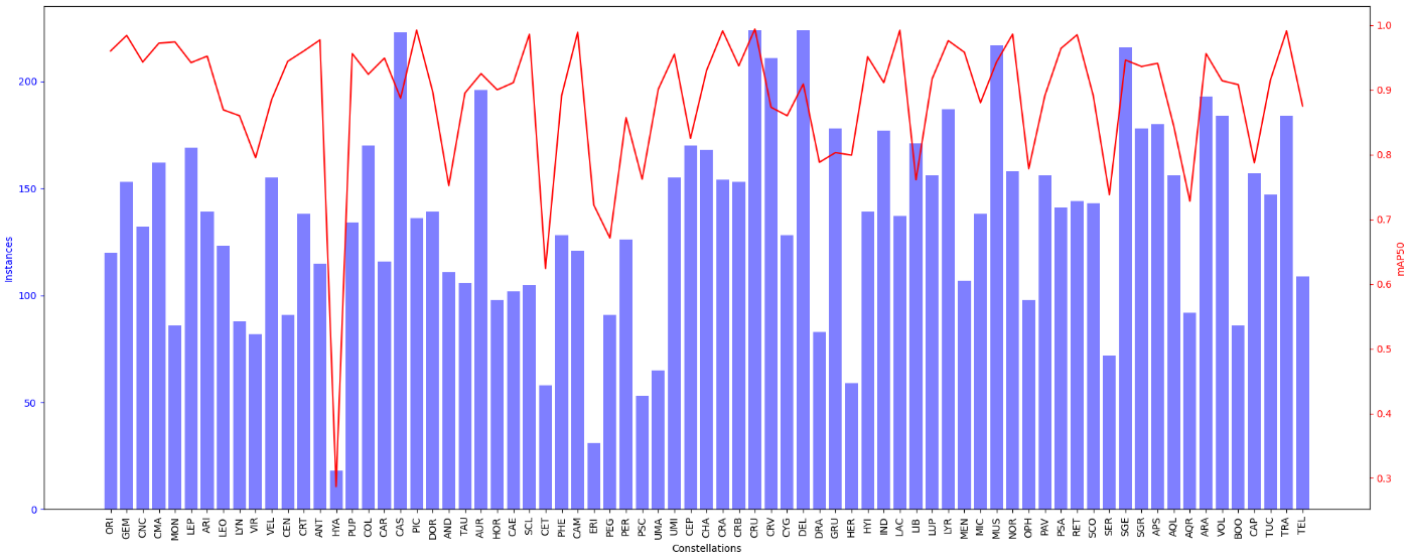


Figure 5.13: Validation set - number of instances vs mAP50

5.5 Results & discussion

In table 5.2 I depicted the result metrics from the test set, and during the training process, on the validation set. The mAP50 value is very high taking into account the uneven class distribution and the variety of pollution levels in the dataset, but it is an indicator of the fact that the algorithm correctly identifies constellations, in general. In addition, the precision and recall values are very good. The mAP50-95 value is lower, but this is also the case on the train set. It indicates that it is hard for the model to output a perfect bounding box each time, mainly due to the differences in pollution levels that lead to different appearances of the constellations (their number of stars decreases with pollution). Examples of predictions made on the test set are illustrated in figures 5.14 (true constellations), and 5.15 (predicted constellations). The confusion matrices for the validation and test sets, as well as more examples of predictions from the test set, are presented in Appendix F.

	Precision	Recall	mAP50	mAP50-95
Validation	91.7%	84.9%	91.3%	68.4%
Test	91.7%	82.7%	88.6%	68.1%

Table 5.2: Results

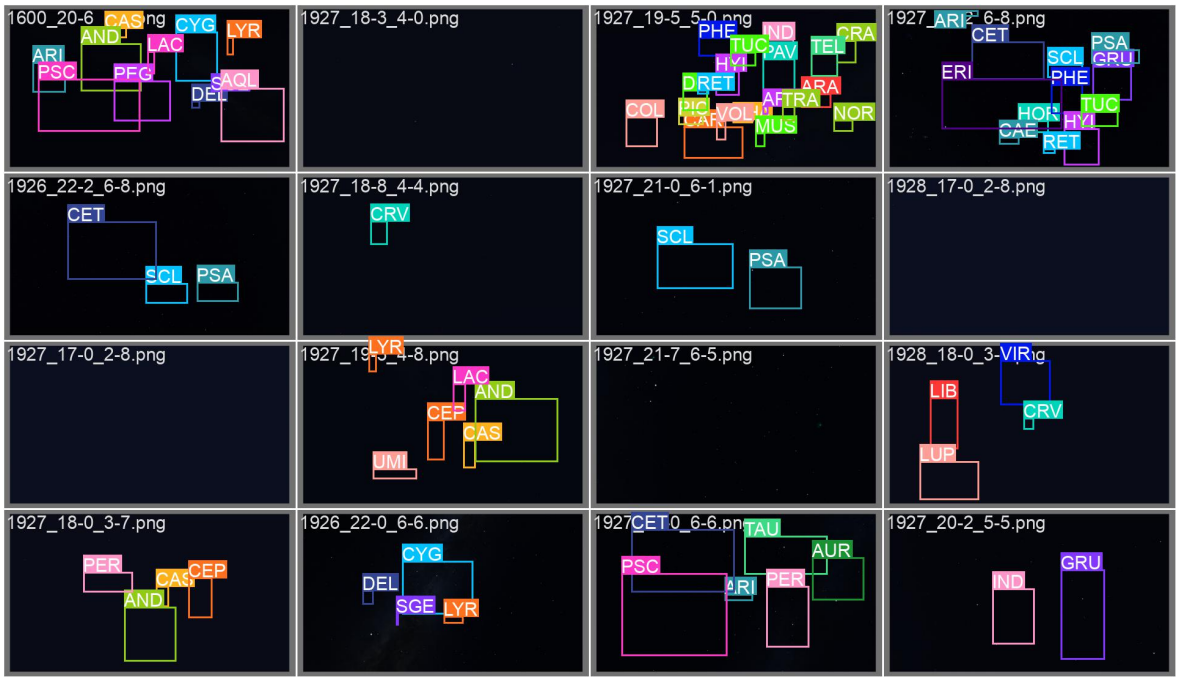


Figure 5.14: Test set - true constellations

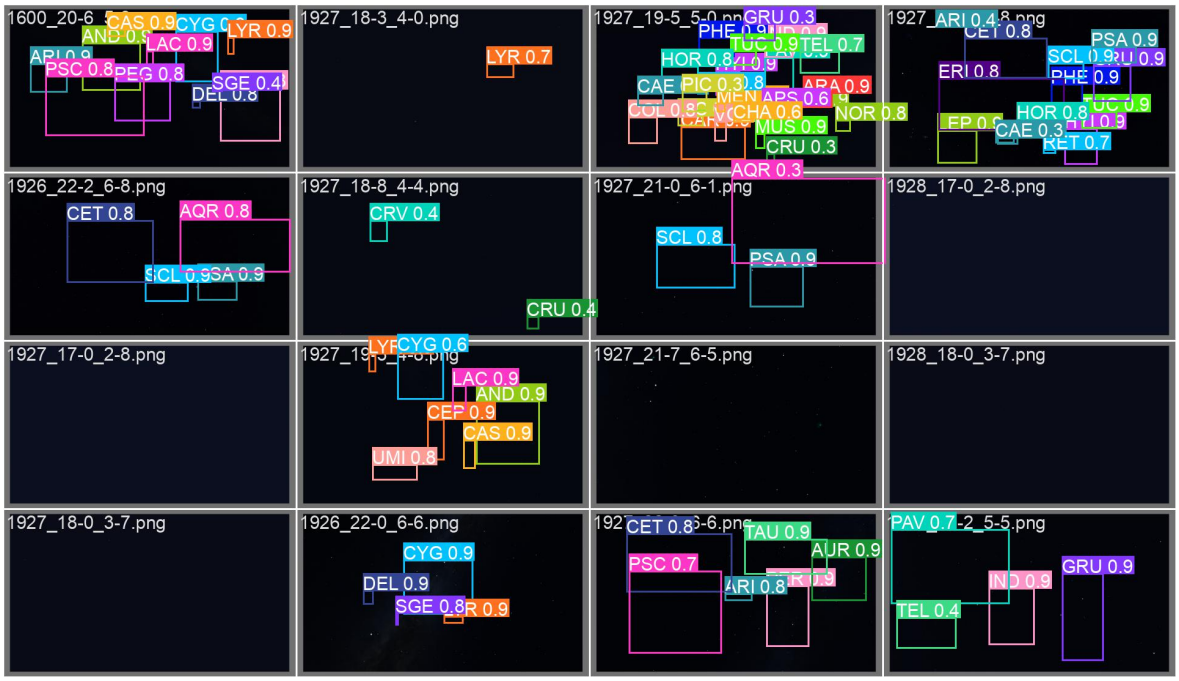


Figure 5.15: Test set - predicted constellations (0)

Chapter 6

Methodology: Mobile application

I developed a mobile application in order to incorporate the previously discussed models into a concrete product. As a disclaimer, an open-source Android application project that detected objects from images was used as the basis for my app (source: Github project). I added the classification server and feature, multiple functionalities such as the red filter, layout changes, information about the detected constellations available on click, and also many changes in order to make my model show the bounding boxes for my specific format.

6.1 Functionalities

The mobile application has several functionalities:

- The user can select a photo from the gallery or take one on the spot from the ”Select” button.
- The user can also select a photo from the test images by pressing the ”Test image” button.
- The user can receive light pollution levels from the selected image. A message with the light pollution level appears on screen when the ”Classify” button is being pressed (Figure 6.1).
- The user can see detected constellations by pressing the ”Detect” button from the selected image (Figure 6.2).
- The user can use the ”Live” button in order to detect constellations in real time.
- The user can press the ”Red filter” button in order to have a red hue on the screen above all elements, as to not have his night vision accommodation affected while stargazing (Figure 6.3).

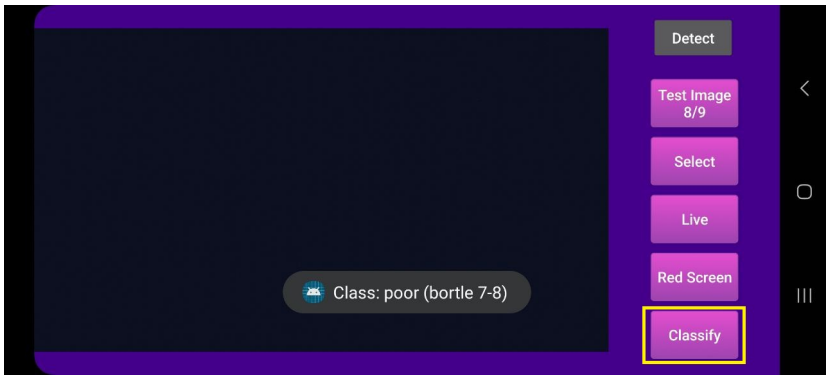


Figure 6.1: Light pollution detection

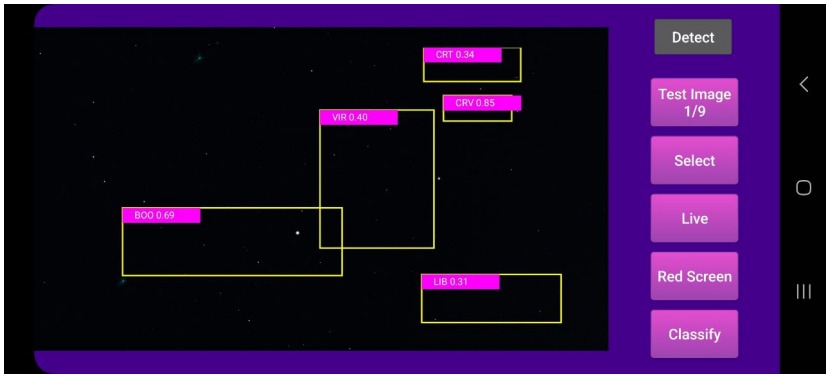


Figure 6.2: Constellation detection from photos

Figure 6.3: Red filter

- The user can press on the bounding boxes of the detected constellations in order to see information about their stars and mythology (Figure 6.4).

6.2 Architecture

6.2.1 Diagrams

I created the following app diagrams in order to illustrate a high-level view of my application: use case diagram (Figure 6.5), sequence diagram (Figure 6.6) and

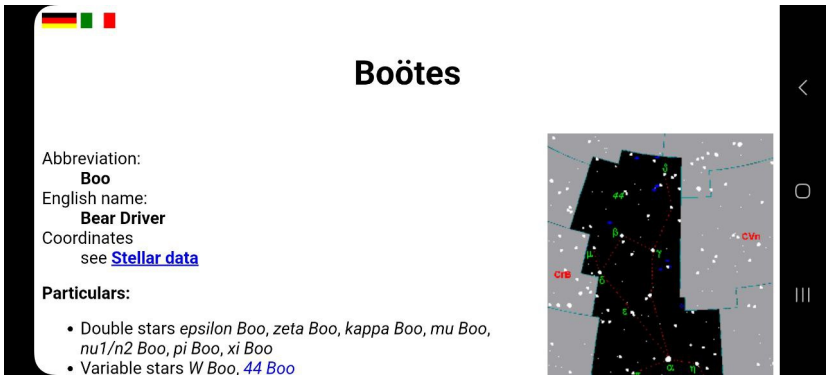


Figure 6.4: Information about the constellation Bootes, displayed when pressing on its bounding box

class diagram (Figure 6.7).

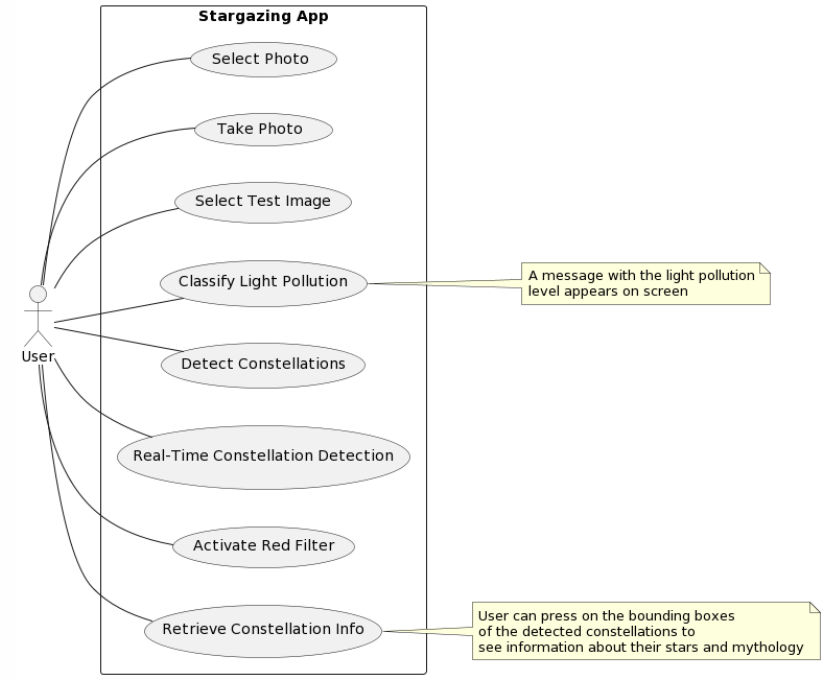


Figure 6.5: Use case diagram

6.2.2 Backend

I developed an Android application using Java in Android Studio, a widely-used integrated development environment (IDE) for Android app development. Within the app, I incorporated a trained YOLO detection model in TFLite format, which is specifically optimized for mobile and embedded devices. The adoption of the TFLite format ensured lightweight and efficient inference, enabling real-time object detection with optimal performance on Android devices. To integrate the YOLO model into the app, I utilized the LiteModuleLoader from PyTorch, a powerful deep learn-

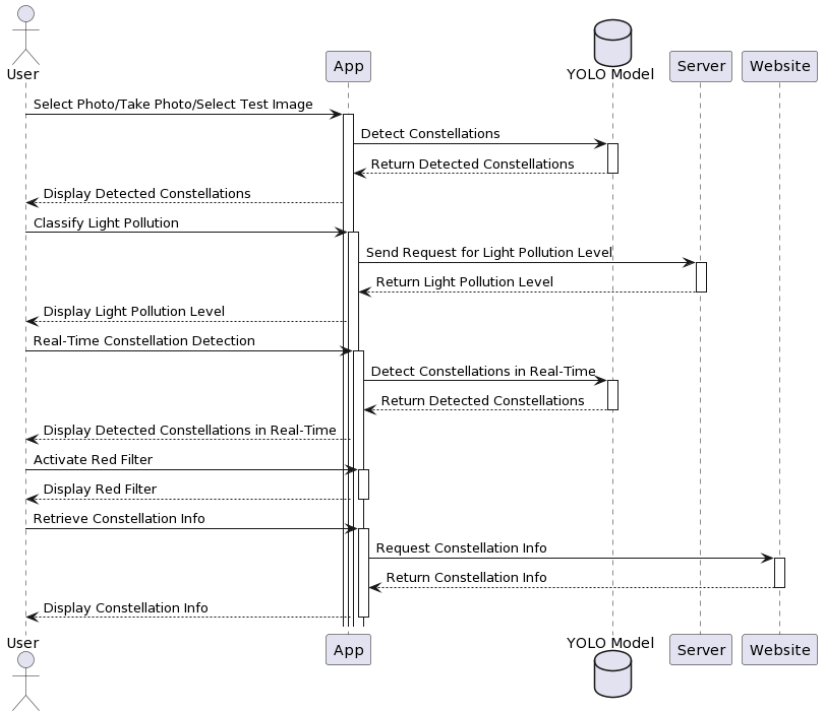


Figure 6.6: Sequence diagram

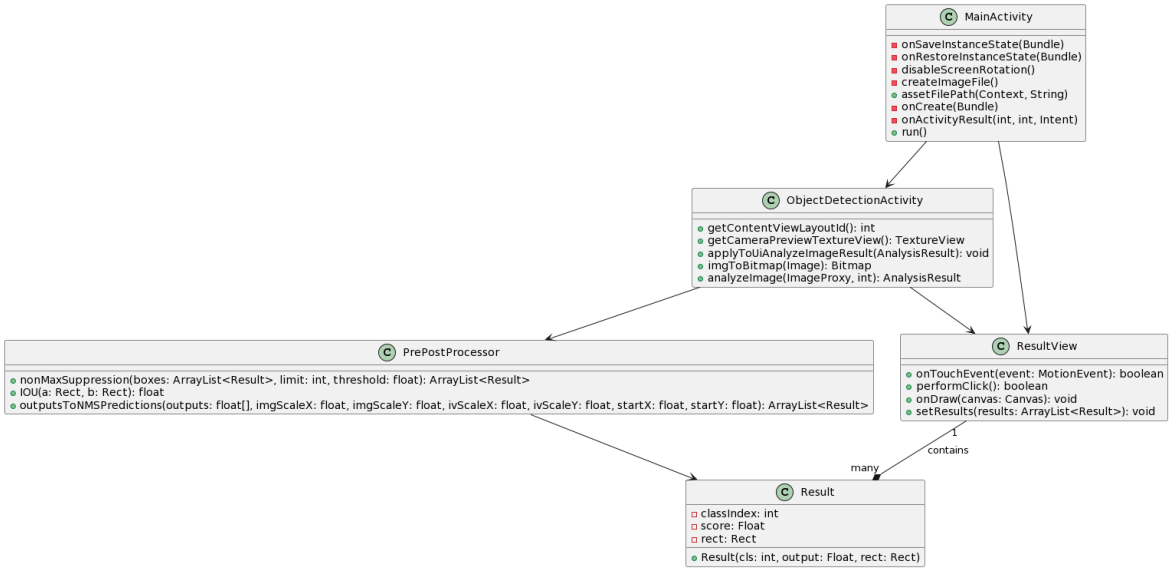


Figure 6.7: Class diagram

ing framework. Additionally, I included a file containing the classes of constellations, facilitating the identification of different celestial objects during the detection process. Leveraging various methods provided by the PyTorch library, I performed model inference on input images.

Furthermore, I implemented a Flask server, a lightweight web framework for Python, which I hosted within a Docker container. This server served as the backend for the application's classification task. Employing Retrofit, a prominent HTTP client library for Android, I established communication between the Android app

and the Flask server. Retrofit simplified the process of making network requests, enabling seamless transmission of photos from the app to the server for light pollution level classification. The integration of Flask as the server-side framework and Retrofit as the client-side library facilitated efficient data exchange between the Android app and the server. Upon receiving an image from the app, the Flask server processed the image using the classification model, which determined the light pollution level.

6.2.3 Frontend

For the frontend of my Android application, I used XML and Java in order to create an engaging user interface and handle the app's functionality. XML (Extensible Markup Language) helped me define the structure and appearance, by specifying sizes, positions and overall aspect of components.

By using Java, I managed to handle the logic and behavior of the components. For example, I defined listeners and handlers of certain events (user interactions wth the app), such as button clicks or touch events. To display information about the constellations, I implemented a web scraping technique. When the user clicks on a bounding box, the onTouchEvent method is triggered, and I retrieve the associated constellation name. Using this name, I dynamically generate a URL that points to an external web resource containing information about the specific constellation. Then, I launch a WebViewActivity with this URL, allowing the user to explore the details of the constellation within the application.

6.2.4 Design patterns

Model-View-Controller (MVC)

In the code, I used the MVC design pattern. The model is represented by the Result class, which contains information about one bounding box (coordinates, class, confidence score). The view is represented by the MainActivity, which displays all buttons and the image to be tested, but also the ObjectDetectionActivity, which represents another screen that deals with the UI elements and interaction events for live detection. The controller is represented by PrePostProcessor class, which processes the data from the view, interacts with the model and updates the view. It also contains a non-max-suppression algorithm which chooses between overlapping bounding boxes, by taking into account the confidence score of the prediction.

Observer pattern

I also used the Observer design pattern, when configuring event listeners (i.e. ”setOnClickListener”). The observer (event listener) waits until the click of the button notifies it of the change. This design pattern is especially used in order to handle UI events.

6.3 Testing

I developed a suite of instrumented tests (Figure 6.8), which fall under the category of integration tests, to assess the app's behavior and functionality in a realistic environment. These tests focus on interactions with UI components, allowing me to simulate user actions and verify the outcomes. To accomplish this, I leveraged Espresso, an Android testing framework that facilitates the simulation of button clicks and other gestures, enabling me to observe their impact on the application. Additionally, I employed JUnit, a widely adopted unit testing framework in the Java ecosystem, utilizing its assertions to compare expected and actual outcomes.

The instrumented tests encompassed several aspects, beginning with the examination of button visibility and state within the MainActivity. Subsequently, I addressed the core features of the application, specifically classification and object detection. For the classification test, I submitted the initial test image and verified that the server responded with the correct class for the given image. Concerning object detection, I assessed the accuracy of the top-ranked class in the model's output.

By employing these instrumented tests, I was able to gain confidence in the app's user interface, the accuracy of its classification capabilities, and the effectiveness of its object detection functionality.

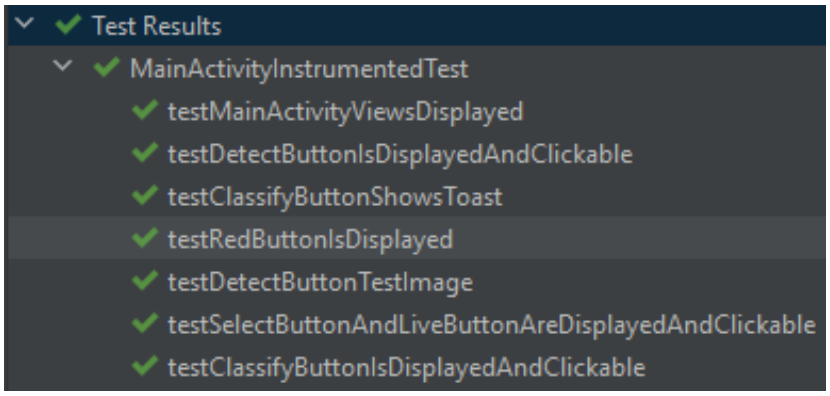


Figure 6.8: Instrumented tests

Chapter 7

Conclusion

7.1 Comparison with SOTA

Although I was unable to find directly comparable literature in terms of datasets, methodology, or even specific objectives for the problems studied, I will attempt to establish connections and draw parallels between my work and existing research.

For the light pollution classification task, the results are presented in table 7.1. My best results were classification using YOLOv8, where I achieved $\approx 99\%$ accuracy on a 1200-image test set, with a training set of 3600 generated images. The next best result was obtained using a mean-value threshold feature selection with random forest or extremely randomized trees, with a linear SVM classifier, with which I obtained 97% accuracy. With my best results surpassing 99% accuracy using YOLOv8 and other methods achieving high accuracy as well, I am confident in the effectiveness of my approach and the fact that light pollution can be evaluated even from images of the night sky, a problem that was never addressed before.

Paper	**Year**	**Dataset**	**Method**	**Accuracy**
[SYT+20]	2020	tabular (from IDA website)	random forest	100%
[SYT+20]	2020	tabular (from IDA website)	decision tree	99.7%
[YLT+20]	2020	2 satellite images	SVM	89.6%
[YLT+20]	2020	2 satellite images	neural network	90.7%

Table 7.1: SOTA results for light pollution classification

For the constellation detection task, there are even fewer papers that I can draw a parallel with, as the majority of related papers focused on star identification, not

constellations as a whole. However, comparable results are presented in table 7.2. I obtained 88.6% mAP50 on the test set, and used a 24000 generated images dataset. Compared to the existing literature on constellation detection, my work stands out as I not only tackled the challenge of detecting constellations, but also considered the influence of varying levels of light pollution in the image data.

Paper	Year	Dataset	Method	Accuracy
[GM21]	2021	1 284 780 generated images	ResNet-like architecture	92.7% (F_1 score)
[JWL15]	2015	14 test images (used templates of constellations for the algorithm)	template matching	71.4%

Table 7.2: SOTA results for constellation detection

7.2 Answers to research questions

RQ1: Yes, light pollution can be effectively classified with machine learning techniques from night sky images, as the obtained results are very good and are also applicable to real data.

RQ2: In general, feature engineering yielded better results than when using deep neural networks on raw data, with the exception of YOLOv8 which gave the best results.

RQ3: In identifying discrete patterns such as constellations, object detection algorithms are sensitive to noise in the data (i.e., light pollution), as seen from my approach with YOLOv5 but also in related studies.

RQ4: Yes, the same model can be trained to detect constellations from both clear and polluted skies, as the high accuracy achieved by YOLOv5 was on a dataset in which the distribution of light pollution in the data was even, ranging from very low to very high levels.

RQ5: The main limitations consist of computational power. I used Kaggle and Google Colab and only had limited processing units with which I worked. Also, the limitations in constellation detection can come from the uneven class distribution, or the automatic preprocessing based on the mean intensity of the images.

RQ6: Testing on real data, I realized that the light pollution classifiers are also applicable there. They could be used for local monitoring of light pollution by anyone, providing valuable information for urban planning, environmental conservation,

and astronomy research. In the case of constellation detection, the developed models offer potential applications in astronomy education, planetariums, and stargazing applications, by enhancing the learning experience.

7.3 SWOT analysis

Strengths

- I managed to classify light pollution from night sky images, a novel approach in this domain. Also, I classified constellations taking into consideration the light pollution level, enhancing the discrete object detection task.
- I created large and balanced datasets with regards to light pollution levels. These datasets can be utilized for various tasks and approaches, and can be easily expanded without requiring any additional effort.
- Classification and detection tasks have a very fast inference time.
- I tried various models and also ensembled them. The various experiments can provide insight for future research on these topics.
- I achieved high performance metrics for both of the studied tasks.

Weaknesses

- I didn't train the models or test on real images. For the light pollution classification task, I tested the algorithm on a few real images with different levels of light pollution, and saw that it can also classify real images. However, I couldn't find real images on which the constellation detection task can work. Images that I found were extremely edited, or didn't contain full constellations inside of them (with all stars visible or not).
- The developed application is only for Android devices.
- Different objects that appear in the night sky images (trees, buildings or the moon) will interfere with the results.

Opportunities

- The application has the potential to be utilized in schools and educational clubs as a means to advocate for astronomy as an essential scientific discipline that should be incorporated into educational curricula.

- The light pollution feature, when perfectioned, can be used by governments for evaluating light pollution levels without the need of expensive equipment, in order to mitigate laws. My pioneer classification from night-sky images approach can serve as a base for a bigger project, that can raise awareness about light pollution concerns.
- The diverse range of tested approaches explored in this study presents an array of possibilities for improvement, paving the way for the initiation of various research avenues and opening up new opportunities for advancement.

Threats

- Switching from the presented synthetic data approach to a real data one is hard. There are no databases with unedited night-sky photos on which the models could be trained, and even if there were, it is hard to manually label them, especially for constellation detection.
- Encouraging widespread adoption and continuous engagement with the application may pose a challenge. Raising awareness about the importance of combating light pollution and promoting the use of the application among the target audience could require effective marketing and educational efforts.

7.4 Future improvements

Several areas for future improvements and research can be explored. For example, the datasets can be enlarged and enhanced with more pollution levels. Also, a dataset with real data can be composed in order to help the model become more applicable in real scenarios. Moreover, models can be trained with more epochs. A hybrid approach can also be explored, in which, for light pollution classification, information could be taken from both night-sky images and local SQM data in order to make a decision. To ensure that trees, buildings or the moon that may appear in images do not interfere with the accuracy of light pollution classification or the detection of constellations, there is an opportunity to develop an algorithm that effectively removes these objects from night sky images. By eliminating their presence, the algorithm can enhance the reliability and precision of the light pollution classification results and improve the detection of constellations within the image.

In conclusion, by opening new horizons in the field of light pollution classification from night sky images and skyglow-informed constellation detection, this research has laid the foundation for further advancements and can spark motivation for future investigations in this domain.

Appendix A

Mathematical models that describe light pollution

Walker's model [Wal73]

Formula: $I = C \cdot P \cdot d^{-2.5}$, where I represents the sky brightness, d the distance, P - population and C depends on the flux per inhabitant. These are detailed in table A.1.

Term	Explanation	Unit of measurement
Sky Brightness (I)	amount of light received per unit area of the sky	$mag/arcsec^2$
Population (P)	number of people in that location	$10^6/unit$
Distance (d)	distance from a city	miles

Table A.1: Walker's model - parameters

Treanor's model [Tre73]

The model representation is as follows: $P = \frac{L(r)}{L(N)} = (\frac{A}{r} + \frac{B}{r^2})e^{-\frac{k}{r}}$, where L(r) is the sky brightness, L(N) the natural sky brightness, P - the population and A, B - constants.

Berry's model [Ber76]

The model formula is $I = a\sqrt{P}(bD^{-2} + cD^{-1})e^{-kD}, D = \sqrt{D^2 + h^2}$, where I is the sky brightness, P - the population, D - distance, h - height of scattering layer, a, b, c - constants, k - aerosol extinction coefficient.

Aubé's model [AFFRSH05]

The formula is $I_{no} \approx I_1 + I_{r1} + I_2 + I_{r2}$, where I_{no} represents the light spectral intensity, while the other terms represents different orders scattered intensities, after or before reflection on the ground.

Appendix B

CNN models' architectures

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure B.1: Architecture of MobileNetV2. It is characterized by a streamlined and efficient structure, featuring depthwise separable convolutions and inverted residual blocks.

[SHZ+18]

Figure B.2: EfficientNetB7 model architecture. It entails a combination of compound scaling, advanced depthwise convolutions, and efficient use of computational resources.

[AA22]

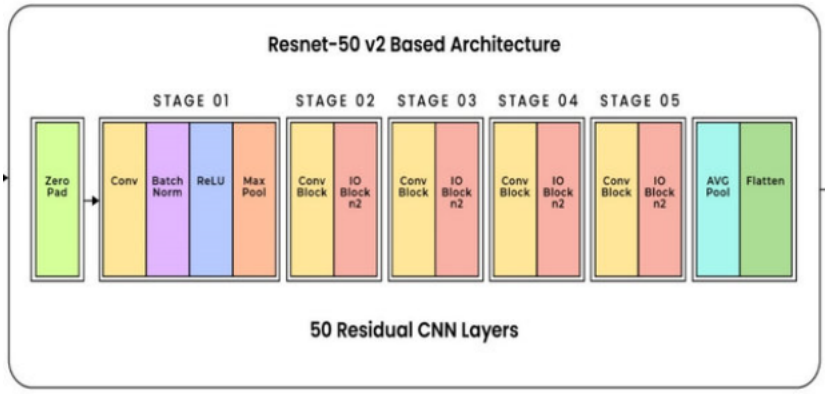


Figure B.3: ResNet50v2 model architecture. It incorporates skip connections and bottleneck layers.

[QZA22]

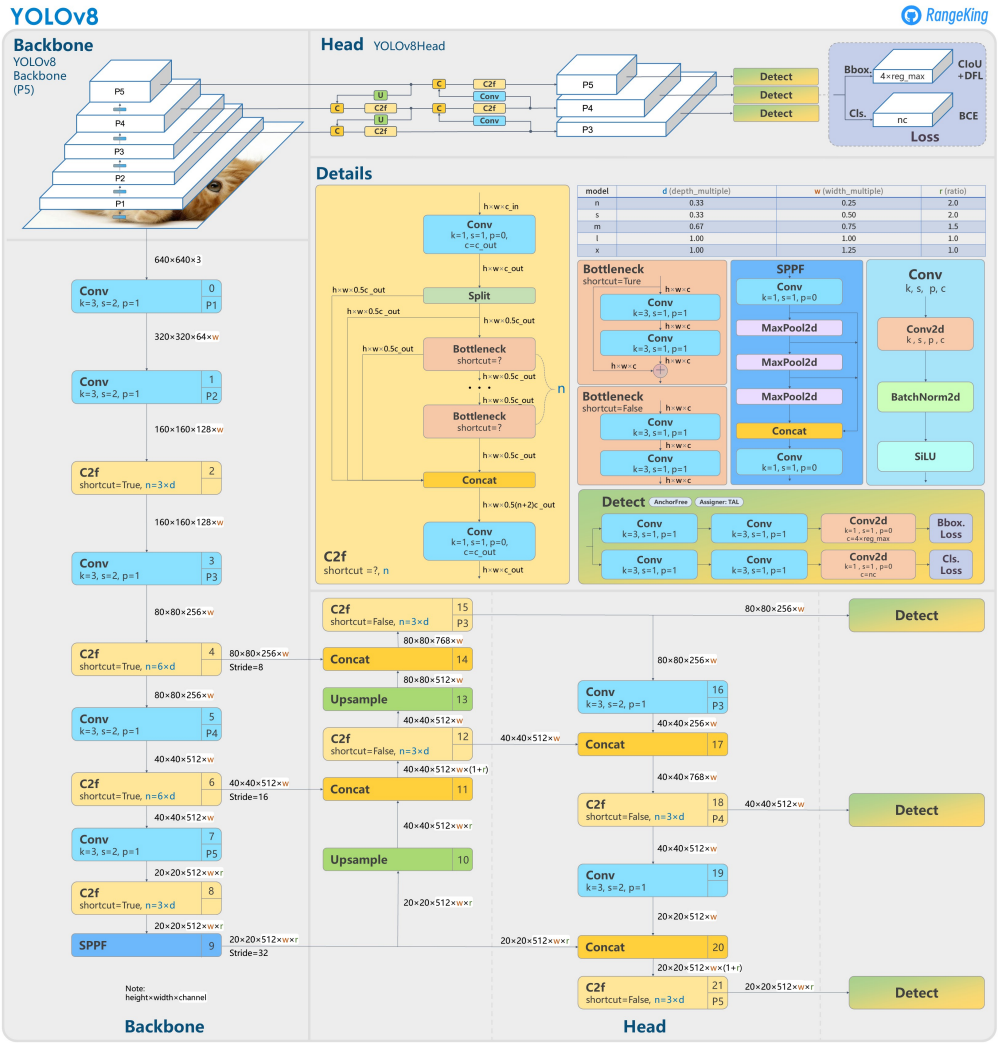


Figure B.4: YOLOv8 Model architecture
[Ran23]

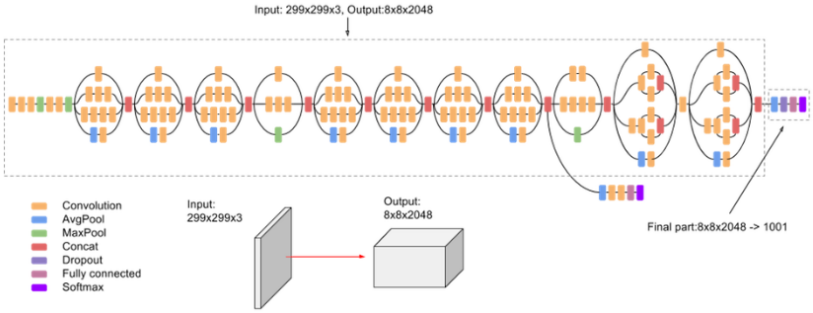


Figure B.5: InceptionV3 model architecture. It incorporates an efficient use of multi-scale feature extraction through inception modules, enabling robust and accurate image classification and recognition tasks.
`https://iq.opengenus.org/inception-v3-model-architecture/`

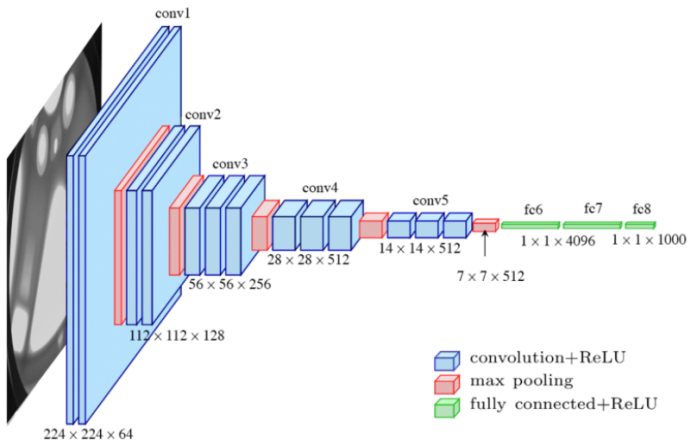


Figure B.6: VGG16 model architecture. It is renowned for its simplicity and effectiveness, consisting of multiple stacked convolutional layers and fully connected layers that enable accurate image classification and feature representation.
[FaLL17]

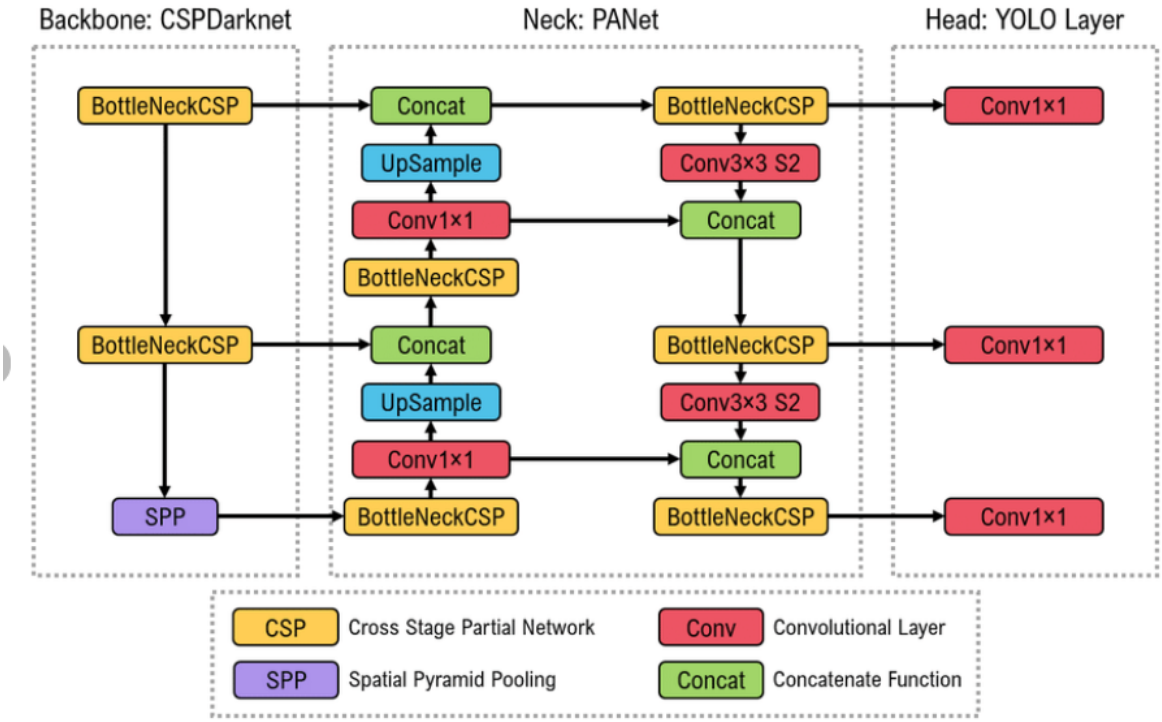


Figure B.7: YOLOv5 model architecture
[KKD+22]

Appendix C

History of training processes and confusion matrices of CNN models trained on raw data

Figure C.1: MobileNetV2: History of training process

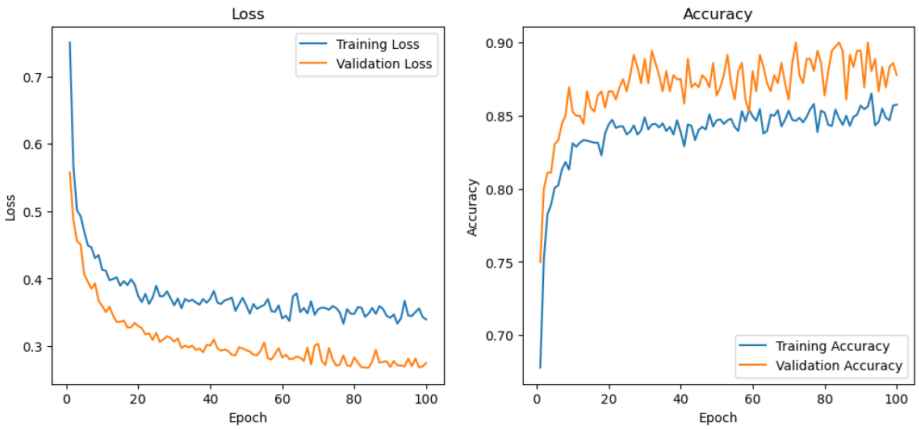


Figure C.2: EfficientNetB7: History of training process

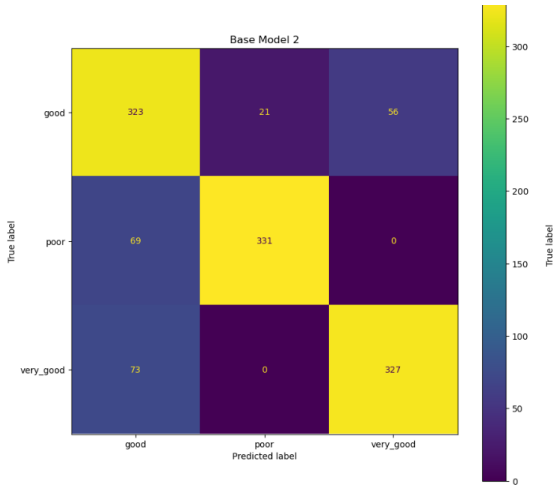


Figure C.3: EfficientNetB7: Confusion matrix

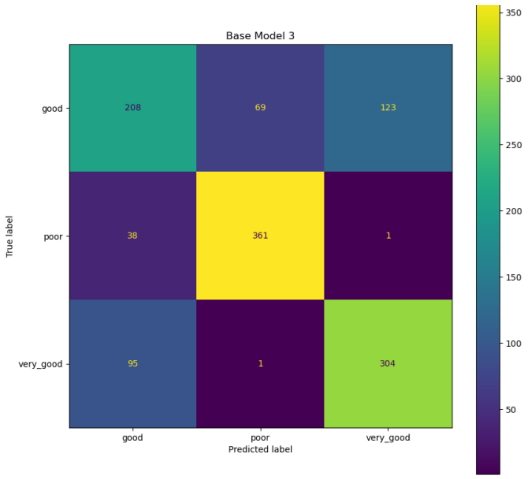


Figure C.4: MobileNetV2: Confusion matrix

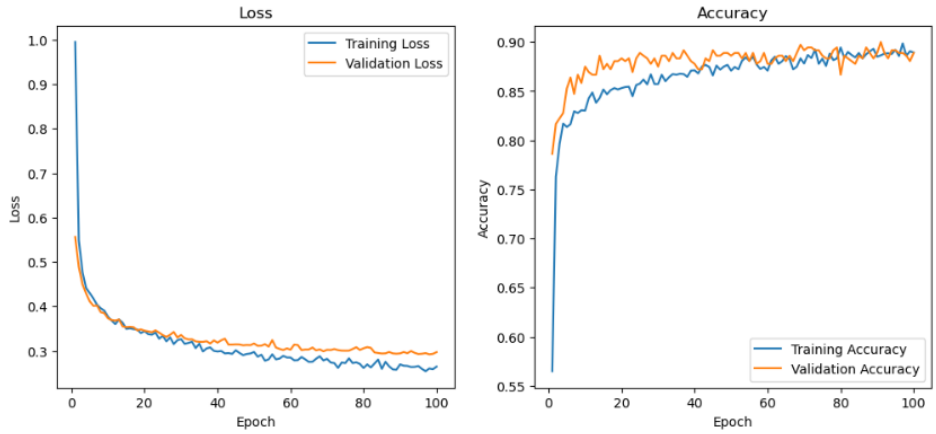


Figure C.5: ResNet50V2: History of training process

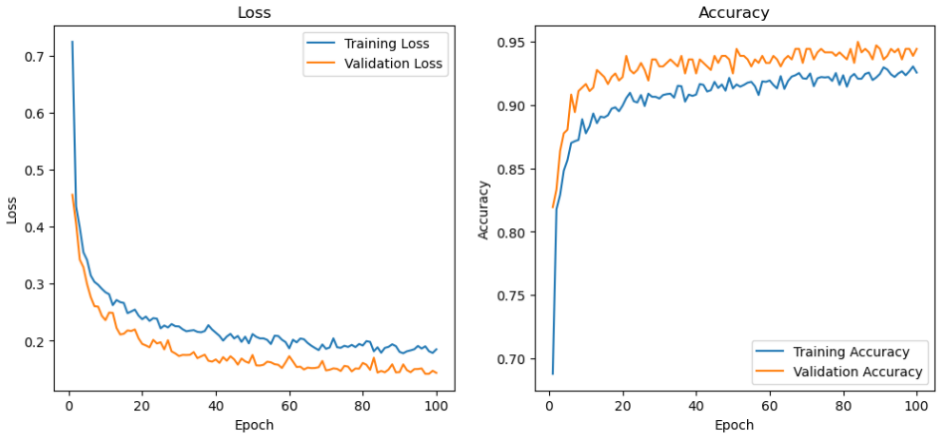


Figure C.6: InceptionV3: History of training process

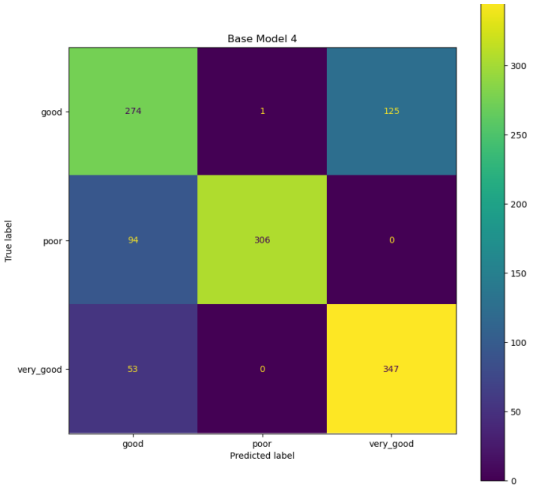


Figure C.7: InceptionV3: Confusion matrix

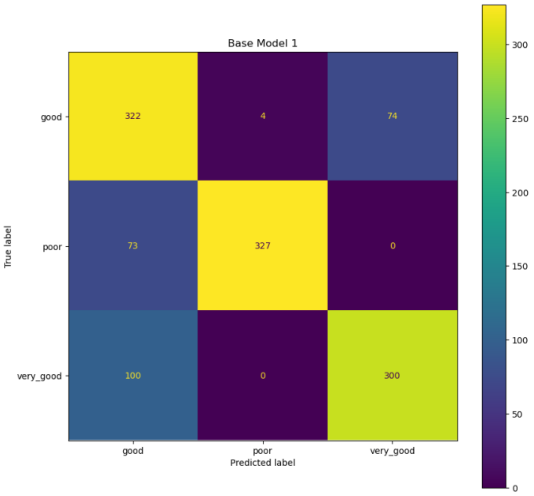


Figure C.8: ResNet50V2: Confusion matrix

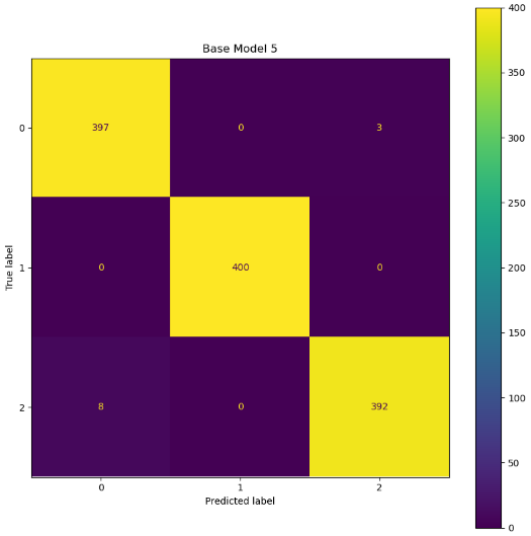


Figure C.9: YOLOv8: Confusion matrix

Appendix D

Confusion matrices: stacking of CNN models

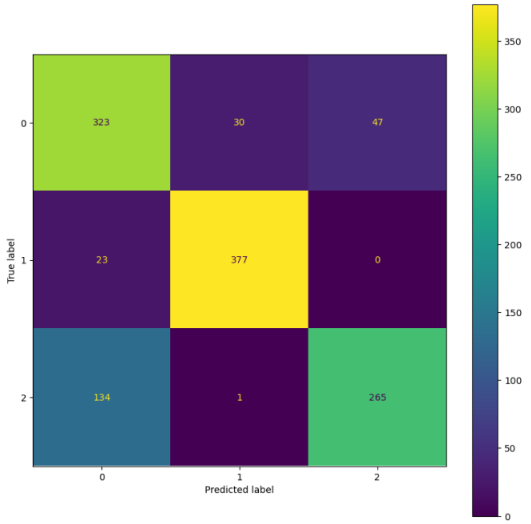


Figure D.1: AdaBoost meta-learner on base probabilities of each class: confusion matrix

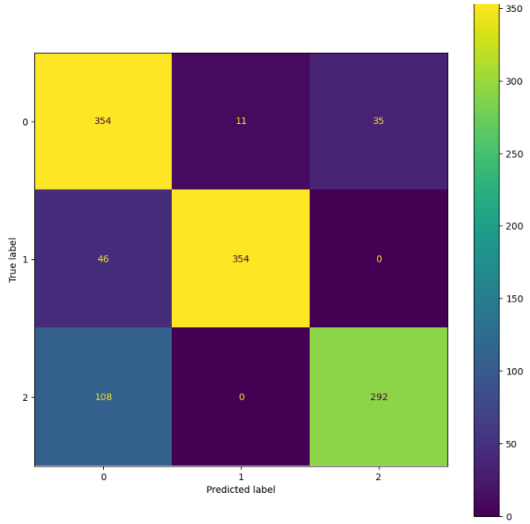


Figure D.2: Stacking ensemble with Gradient boosting classifier meta-learner based on base learners' predicted classes: confusion matrix

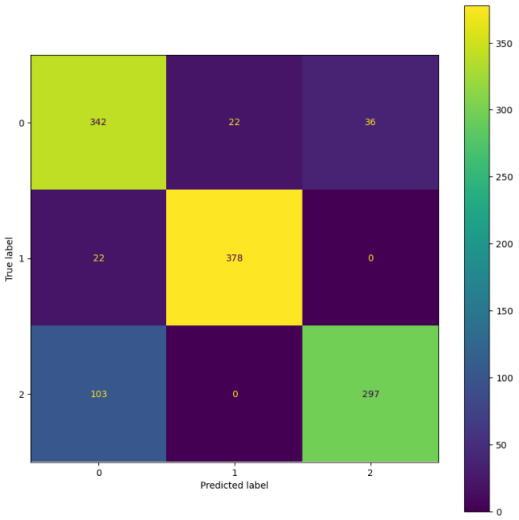


Figure D.3: Stacking ensemble with Gradient boosting classifier meta-learner based on base learners' probabilities of each class: confusion matrix

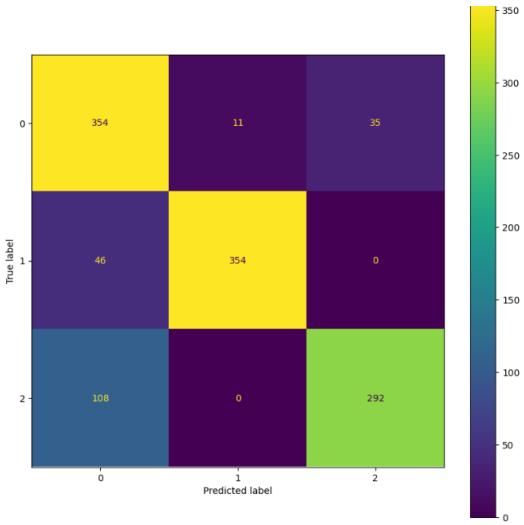


Figure D.4: Stacking ensemble with XGBoost meta-learner based on base learners' predicted classes: confusion matrix

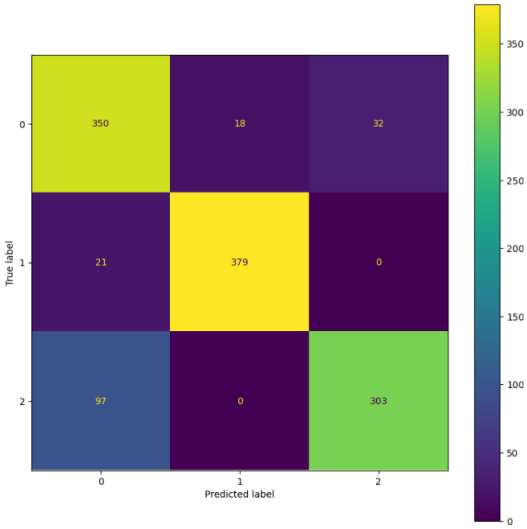


Figure D.5: Stacking ensemble with XGBoost meta-learner based on base learners' probabilities of each class: confusion matrix

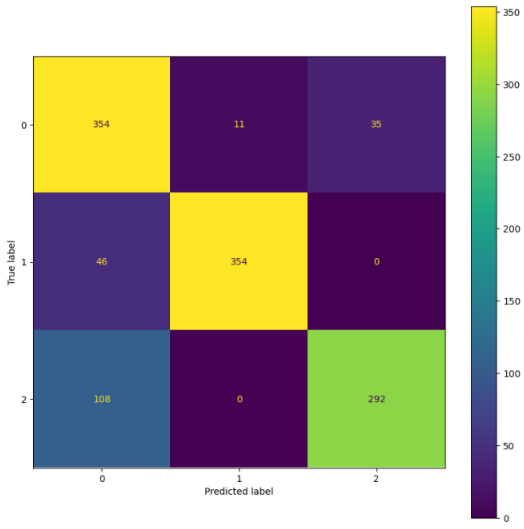


Figure D.6: Stacking ensemble with Random forest meta-learner based on base learners' predicted classes: confusion matrix

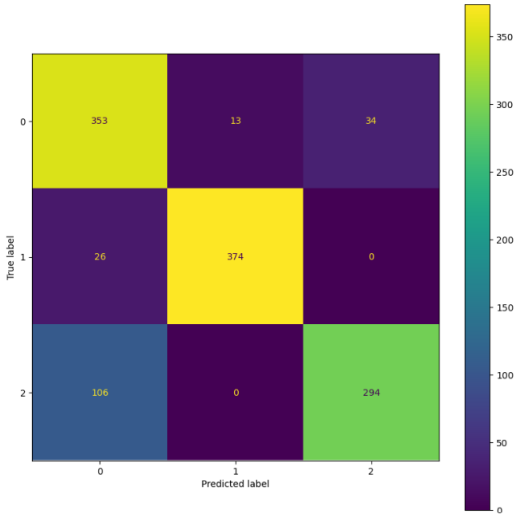


Figure D.7: Stacking ensemble with Random forest meta-learner based on base learners' probabilities of each class: confusion matrix

Appendix E

Confusion matrices: ensembling of models trained on extracted features

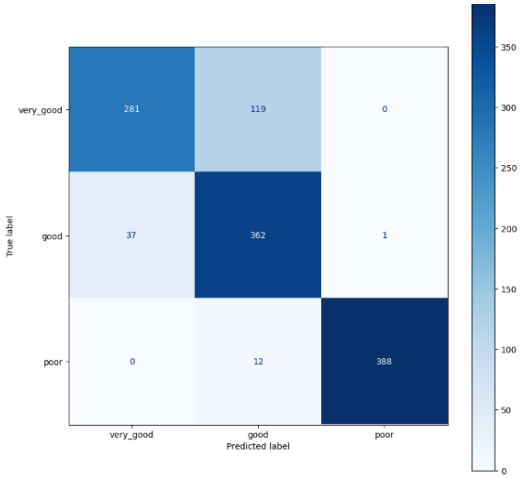


Figure E.1: Hard voting: confusion matrix

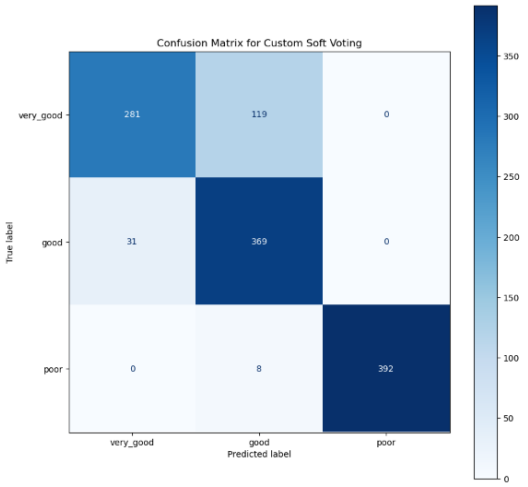


Figure E.2: Soft voting: confusion matrix

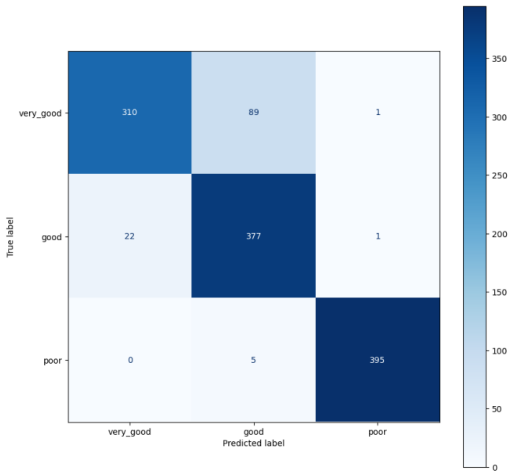


Figure E.3: XGBoost meta-learner: confusion matrix

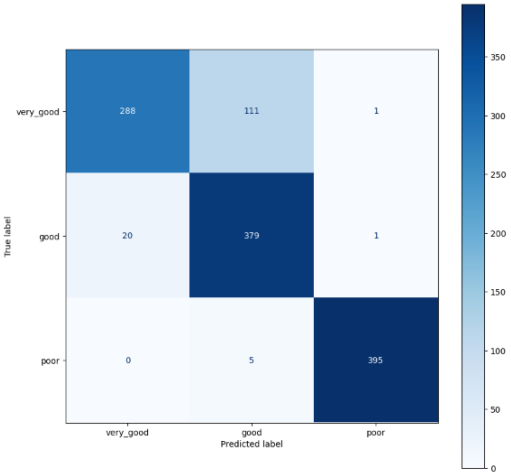


Figure E.4: AdaBoost meta-learner: confusion matrix

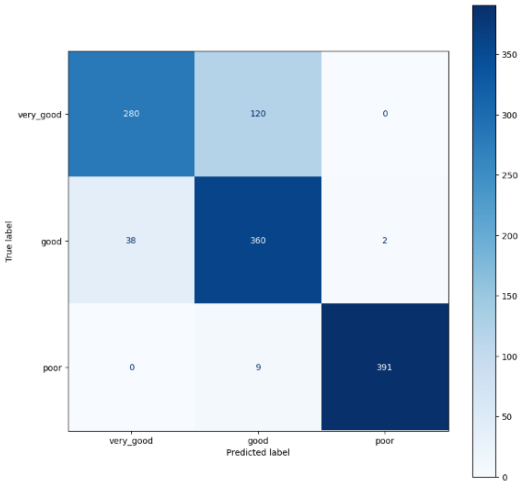


Figure E.5: Gradient boosting meta-learner: confusion matrix

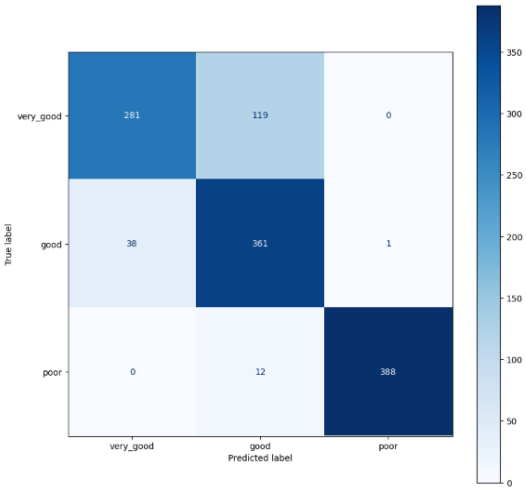


Figure E.6: Random forest meta-learner: confusion matrix

Appendix F

Constellation detection metrics

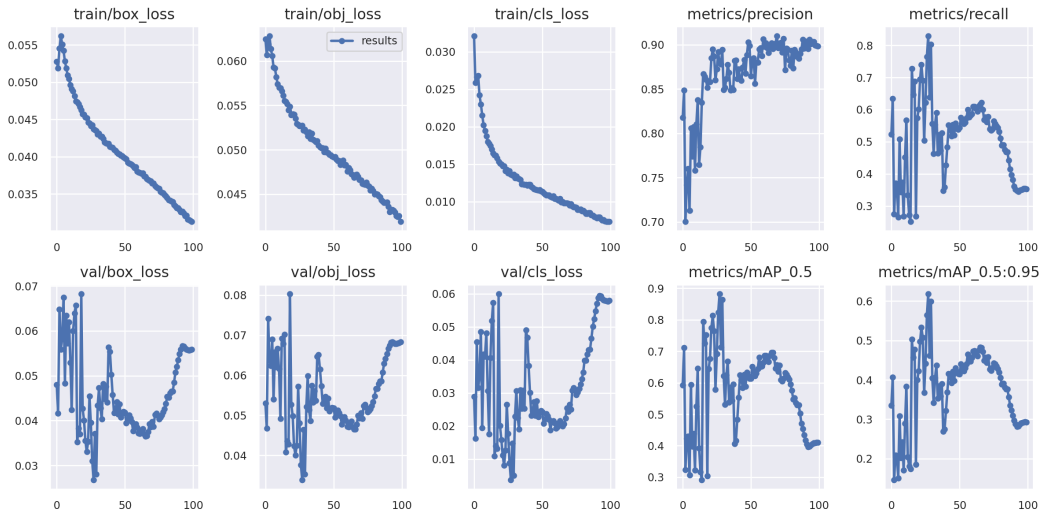


Figure F.1: First 100 epochs (with pretrained weights) - box filter in preprocessing

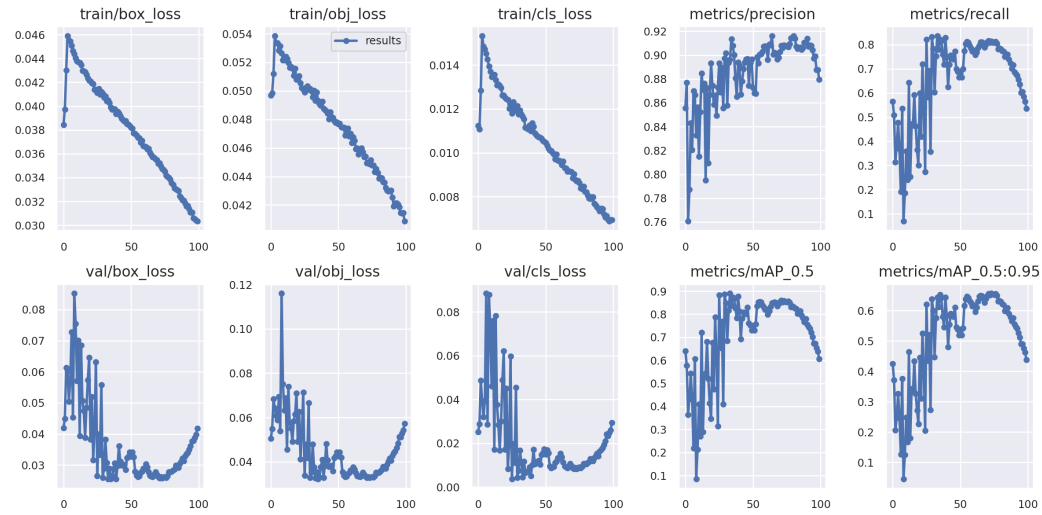


Figure F.2: Epochs 100-200 (with best weights from 0-100) - box filter in preprocessing

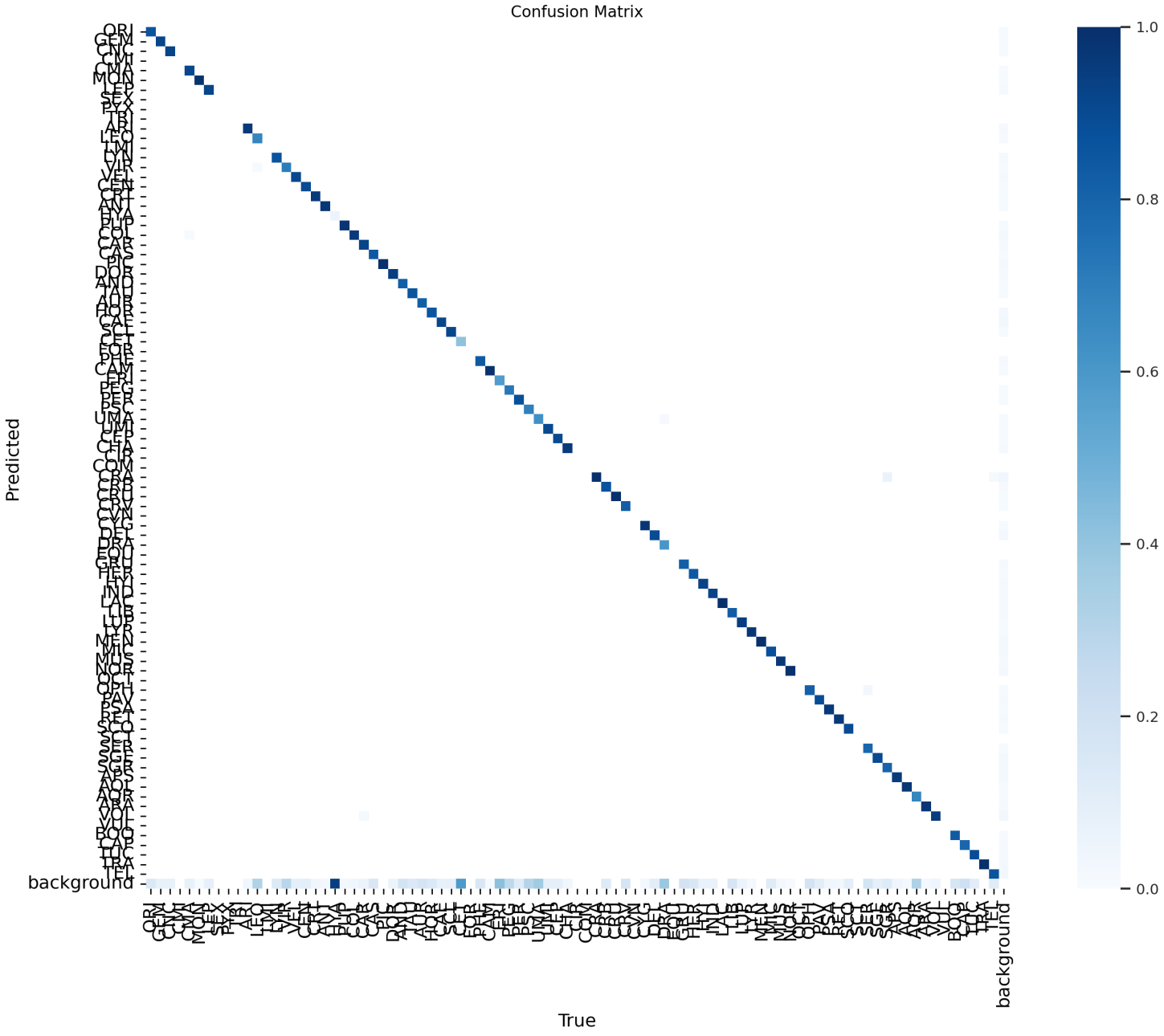


Figure F.3: Confusion matrix for constellation detection - validation set

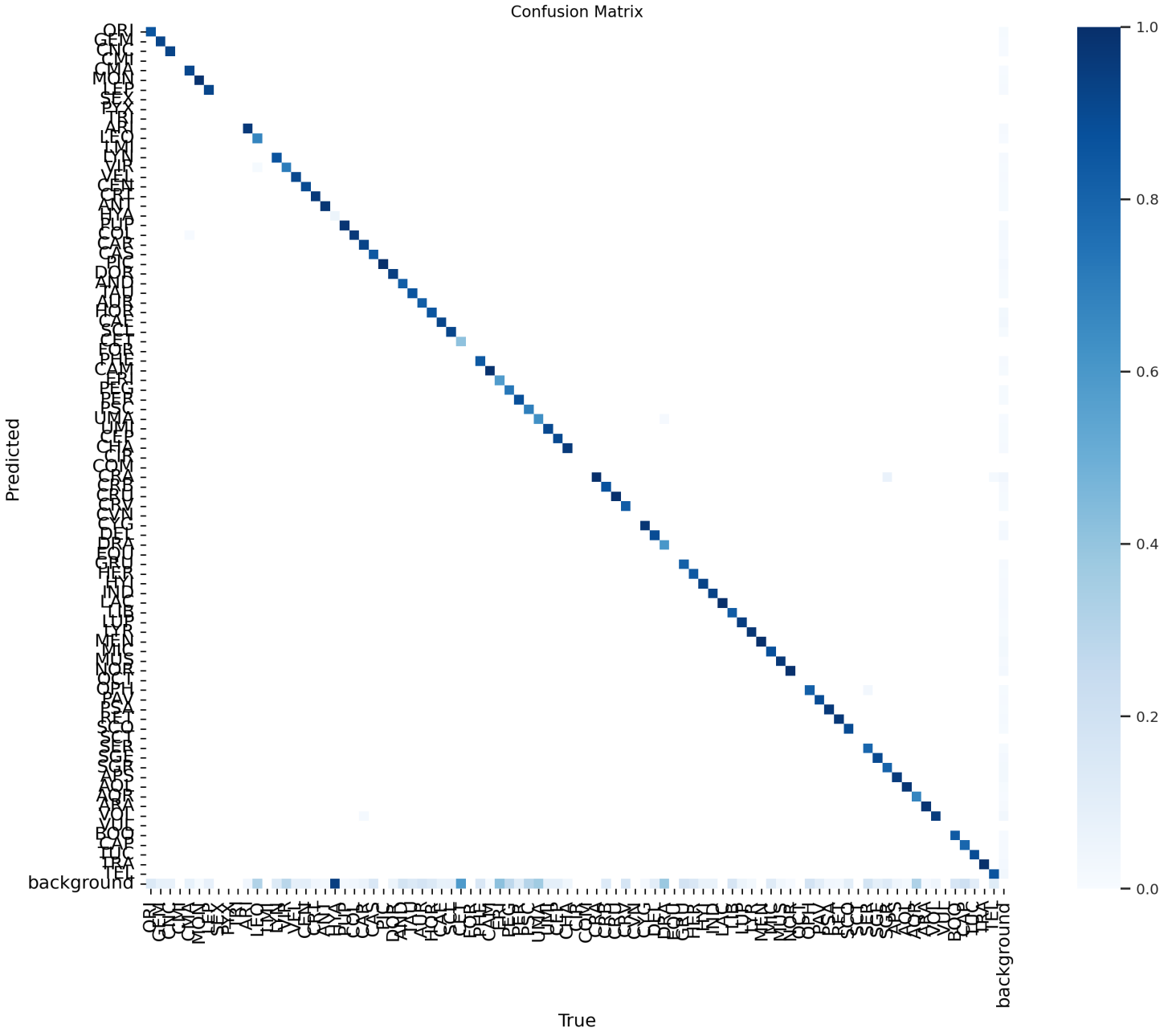


Figure F.4: Confusion matrix - test set

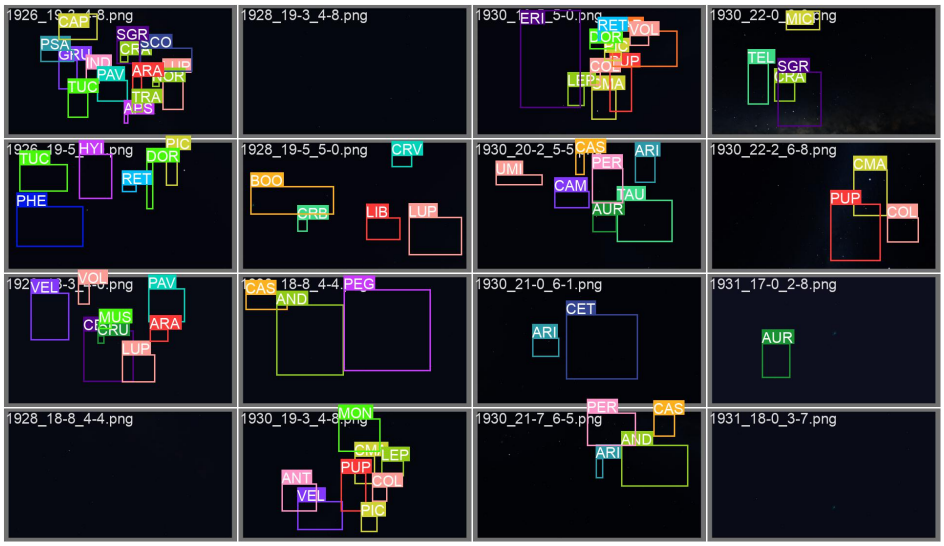


Figure F.5: Test set - true constellations (1)

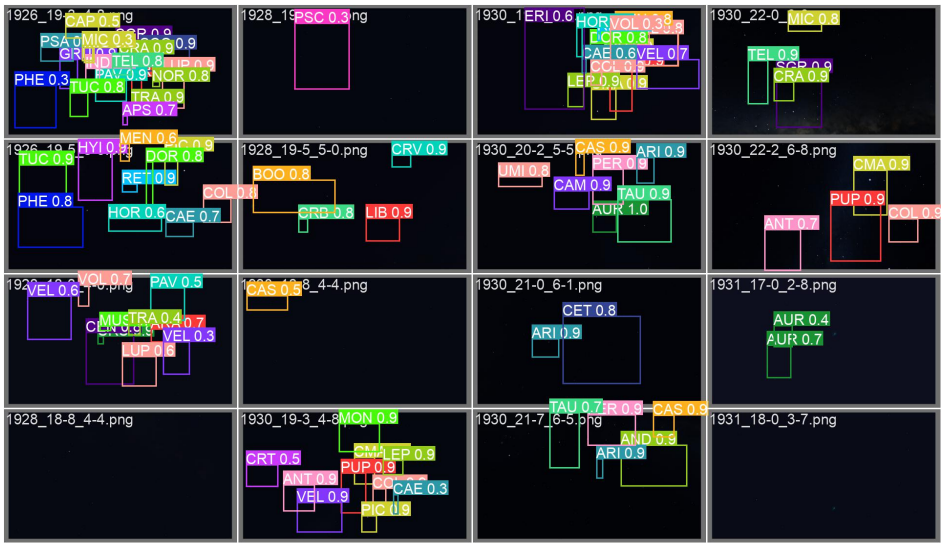


Figure F.6: Test set - predicted constellations (1)

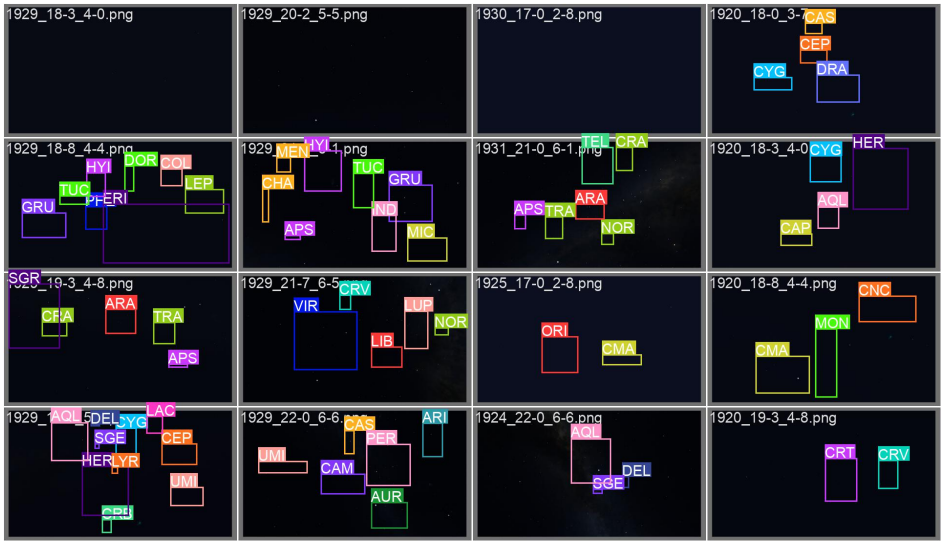


Figure F.7: Test set - true constellations (2)

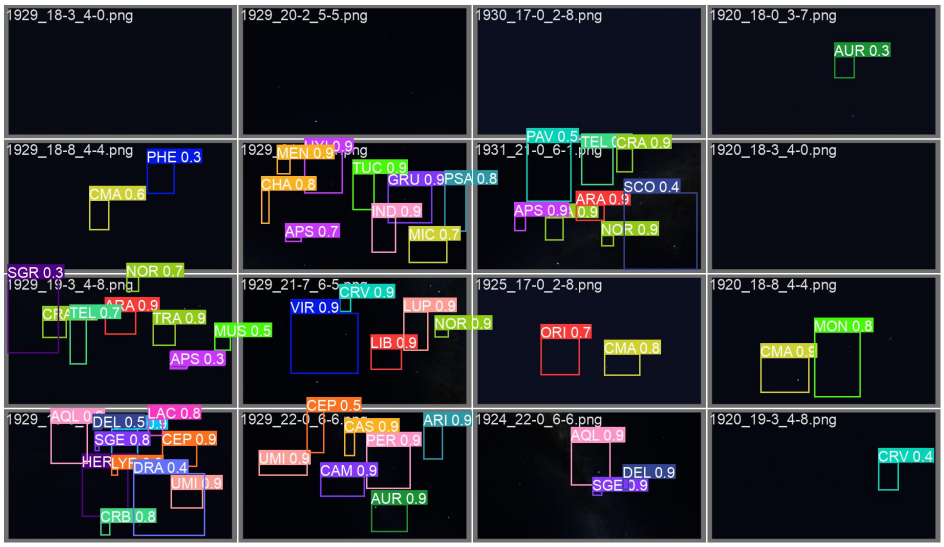


Figure F.8: Test set - predicted constellations (2)

Bibliography

[AA22] Cihan Akyel and Nursal Arıcı. Linknet-b7: Noise removal and lesion segmentation in images of skin cancer, 2022.

[AAMA17] Saad Albawi, Tareq Abed Mohammed, and Saad ALZAWI. Understanding of a convolutional neural network. 08 2017.

[AFFRSH05] M Aubé, L Franchomme-Fossé, P Robert-Staehler, and V Houle. Light pollution modelling and detection in a heterogeneous environment: Toward a night-time aerosol optical depth retreival method. In *Atmospheric and Environmental Remote Sensing Data Processing and Utilization: Numerical Atmospheric Prediction and Environmental Monitoring*, volume 5890, pages 248–256. SPIE, 2005.

[And12] David G Andrews. *An Introduction to Atmospheric Physics*. Cambridge University Press, 2 edition, 2012.

[AZH+21] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8:1–74, 2021.

[Bak95] Michael E. Bakich. *The Cambridge Guide to the Constellations*. Cambridge University Press, 1995.

[Ber76] Richard L Berry. Light pollution in southern ontario. *Journal of the Royal Astronomical Society of Canada, Vol. 70, p. 97*, 70:97, 1976.

[Bey17] Michael Beyeler. *Machine Learning for OpenCV*. Packt Publishing Ltd, 2017.

[BK08] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[Bor01] John E. Bortle. *The Bortle Dark-Sky Scale*. Sky & Telescope, Cambridge, MA, 2001.

[Bru81] Frans Bruin. Atmospheric refraction and extinction near the horizon. *Archive for History of Exact Sciences*, 25(1):1–17, 1981.

[CFE00] Pierantonio Cinzano, Fabio Falchi, and Christopher D Elvidge. The world atlas of the artificial night sky brightness. *Monthly Notices of the Royal Astronomical Society*, 318(3):641–657, 2000.

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[dC16] Manfredo P. do Carmo. *Differential Geometry of Curves and Surfaces*. Dover Publications, New York, 1st edition, 2016.

[DCT+22] Florin Dumitrescu, Bogdan Ceachi, Ciprian-Octavian Truică, Mihai Trăscău, and Adina Magda Florea. A novel deep learning-based relabeling architecture for space objects detection from partially annotated astronomical images. *Aerospace*, 9(9):520, 2022.

[DDS+09a] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[DDS+09b] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[Dic16] Terence Dickinson. *NightWatch: A Practical Guide to Viewing the Universe*. Firefly Books, 2016.

[Dur13] Dan M Duriscoe. Measuring anthropogenic sky glow using a natural sky brightness model. *Publications of the Astronomical Society of the Pacific*, 125(933):1370–1382, 2013.

[FaLL17] Max Ferguson, Ronay ak, Yung-Tsun Lee, and Kincho Law. Automatic localization of casting defects with convolutional neural networks. pages 1726–1735, 12 2017.

[FCD+16] Fabio Falchi, Pierantonio Cinzano, Dan Duriscoe, Christopher CM Kyba, Christopher D Elvidge, Kimberly Baugh, Boris A Portnov, Na-

taliya A Rybnikova, and Riccardo Furgoni. The new world atlas of artificial night sky brightness. *Science advances*, 2(6):e1600377, 2016.

[FCV+04] F. Felice, Mariateresa Crosta, Alberto Vecchiato, M. Lattanzi, and Bianca Bucciarelli. A general relativistic model of light propagation in the gravitational field of the solar system: The static case. *The Astrophysical Journal*, 607, 01 2004.

[FH23] Hangrui Fan and Shiwen He. Control light pollution by genetic algorithms. *Highlights in Science, Engineering and Technology*, 35:233–241, 2023.

[FvDFH13] James D Foley, Andries van Dam, Steven K Feiner, and John F Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Professional, 2013.

[Gar86] R. H. Garstang. Model for artificial night-sky illumination. *Publications of the Astronomical Society of the Pacific*, 98(601):364, mar 1986.

[GM21] VA Galkin and AV Makarenko. Neural network approach to recognition of visible constellations by sky photo image. In *Journal of Physics: Conference Series*, volume 1864, page 012014. IOP Publishing, 2021.

[GW02] Rafael C Gonzalez and Richard E Woods. *Digital image processing*. Pearson, 3 edition, 2002.

[GW18] R.C. Gonzalez and R.E. Woods. *Digital Image Processing, Global Edition*. Pearson Education, 2018.

[Gé19] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly, 2019.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[JCQ23] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. YOLO by Ultralytics, January 2023.

[JCS+22] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, (Zeng Yifu), Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe,

Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, November 2022.

[JWL15] S. Ji, J. Wang, and X. Liu. Constellation detection. Final Project for Spring 2014-2015, Stanford, 2015. Stanford Press.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KJ20] Alok Kumar and Mayank Jain. *Ensemble Learning for AI Developers: Learn Bagging, Stacking, and Boosting Methods with Use Cases*. Apress, Berkeley, CA, 2020.

[KKD+22] Iason Katsamenis, Eleni Karolou, Agapi Davradou, Eftychios Protopapadakis, Anastasios Doulamis, Nikolaos Doulamis, and Dimitris Kalogeras. Tracon: A novel dataset for real-time traffic cones detection using deep learning. 05 2022.

[KL16] Yali Katz and Noam Levin. Quantifying urban light pollution—a comparison between field measurements and eros-b imagery. *Remote Sensing of Environment*, 177:65–77, 2016.

[Koc07] Miroslav Kocifaj. Light-pollution model for cloudy and cloudless night skies with ground-based light sources. *Applied optics*, 46(15):3013–3022, 2007.

[KTB+15] Christopher Kyba, Kai Tong, Jonathan Bennie, Jennifer J Birriel, Dirk Blank, Dmitry Chechin, Pierantonio Cinzano, Mark Coleman, Dan Duriscoe, Fabio Falchi, et al. Worldwide variations in artificial skyglow. *Sci Rep*, 5:8409, 2015.

[Lak19] Valliappa Lakshmanan. *Data Science on the Google Cloud Platform, 2nd Edition*. O'Reilly Media, Inc., 2019.

[ld] Scikit learn developers. scikit-learn documentation. Accessed: 2023-04-17.

[LHM+10] Dustin Lang, David W Hogg, Keir Mierle, Michael Blanton, and Sam Roweis. Astrometry. net: Blind astrometric calibration of arbitrary astronomical images. *The astronomical journal*, 139(5):1782, 2010.

[Lie92] C. C. Liebe. Pattern recognition of star constellations for spacecraft applications. *IEEE Aerospace and Electronic Systems Magazine*, 7(6):34–41, June 1992.

[LSHJ+22] Frank A La Sorte, Kyle G Horton, Alison Johnston, Daniel Fink, and Tom Auer. Seasonal associations with light pollution trends for nocturnally migrating bird populations. *Ecosphere*, 13(3):e3994, 2022.

[mat23] Gaussian distribution. `https://www.math.net/gaussian-distribution`, 2023.

[MH23] Harun Mehmedinovic and Gavin Heffernan. Skyglow project. `https://skyglowproject.com/`, 2023.

[MRMS23] Debani Prasad Mishra, Kshirod Kumar Rout, Sivkumar Mishra, and Surender Reddy Salkuti. Various object detection algorithms and their comparison. *Indonesian Journal of Electrical Engineering and Computer Science*, 29(1):330–338, 2023.

[MVM+16] Víctor Hugo Masías, Mauricio Valle, Carlo Morselli, Fernando Crespo, Augusto Vargas Schüler, and Sigifredo Laengle. Modeling verdict outcomes using social network measures: The watergate and caviar network cases. *PloS one*, 11:e0147248, 01 2016.

[NB21] Attila Nagy and Ábel Boros. Improving the sample-efficiency of neural architecture search with reinforcement learning. *arXiv preprint arXiv:2110.06751*, 2021.

[Ope23] OpenCV. Basic Linear Transformations. `https://docs.opencv.org/3.4/d3/dc1/tutorial_basic_linear_transform.html`, Accessed 2023.

[PG17] Josh Patterson and Adam Gibson. *Deep Learning*. O'Reilly Media, Inc., August 2017.

[PKD97] Curtis Padgett and Kenneth Kreutz-Delgado. A grid algorithm for autonomous star identification. *IEEE Transactions on Aerospace and Electronic Systems*, 33(1):202–213, 1997.

[PSAWS06] Lalitha Paladugu, Ebenezer Seisie-Amoasi, Brian G. Williams, and Marco P. Schoen. Intelligent star pattern recognition for attitude determination: the "lost in space" problem. *Journal of Aerospace Computing, Information, and Communication*, 3(11):538–549, 2006.

[PVG+] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. scikit-learn: Machine learning in Python, 2011–.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PVS+21] Ana-Maria Adriana Piso, O Voicu, P Sprimont, Bogdan Bija, and ÓA Lasheras. gendared: The generic data reduction framework for space surveillance and its applications. In *Proceedings of the The 8th European Conference on Space Debris, Darmstadt, Germany*, pages 20–23, 2021.

[PyT21a] PyTorch. torch.nn.functional.conv2d. `https://pytorch.org/docs/stable/generated/torch.nn.functional.conv2d.html`, 2021. Accessed on 2023.

[PyT21b] PyTorch. torch.nn.module. `https://pytorch.org/docs/stable/generated/torch.nn.Module.html`, 2021. Accessed: 2023.

[PyT23] PyTorch. torch.nn.parameter.parameter. `https://pytorch.org/docs/stable/generated/torch.nn.parameter.Parameter.html`, Accessed 2023.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[QZA22] Emad Qazi, Tanveer Zia, and Abdulrazaq Almorjan. Deep learning-based digital image forgery detection system. *Applied Sciences*, 12:2851, 03 2022.

[Ram20] Nikhil Ramolla. Star tracker. `https://github.com/nikhilramolla/star-tracker`, 2020. Accessed: April 20, 2023.

[Ran23] RangeKing. Rangeking's homepage. `https://github.com/RangeKing`, accessed 2023.

[RIU+22] Lala Septem Riza, Ahmad Izzuddin, Judhistira Aria Utama, Khyrina Airin Fariza Abu Samah, Dhani Herdiwijaya, Taufiq Hidayat, Rinto Anugraha, and Emanuel Sungging Mumpuni. Data analysis techniques in light pollution: A survey and taxonomy. *New Astronomy Reviews*, page 101663, 2022.

[RYB+20] David Rijlaarsdam, Hamza Yous, Jonathan Byrne, Davide Oddenino, Gianluca Furano, and David Moloney. Efficient star identification using a neural network. *Sensors*, 20(13):3684, 2020.

[SCP15] Iharka Szücs-Csillik and Dora Poputa. The astro-biblio-students program. 33:101–111, 12 2015.

[Ser] Vladimir Serdarushich. Rectangular to polar coordinate system conversion. `http://www.nabla.hr/FU-RectPolarCoSyst3.htm`.

[Ser18] Sefik Ilkin Serengil. A step-by-step gradient boosting example for classification, 2018.

[Sha19] Rajalingappaa Shanmugamani. *Deep Learning for Computer Vision: Expert techniques for training neural networks using TensorFlow and Keras*. Packt Publishing Ltd, 2019.

[SHZ+18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[SLSMP09] Héctor Solano Lamphar and Ramón San Martín Páramo. Mathematical model for the measurement of light pollution. *2nd Bhartatiaya Vigyan Sammelan, Devi Ahilya Vishwavidyalaya. Indore, India*, 2009.

[Sny87] John Parrish Snyder. *Map projections–a working manual*. US Government Printing Office, 1987.

[SS] H STELLING, JR and W SCOTT, III. The defense meteorological satellite program/dmsp. In *10th Annual Meeting and Technical Display*, page 273.

[Ste21] Stereolabs. Performance of yolo v5, v7, and v8. `https://www.stereolabs.com/blog/performance-of-yolo-v5-v7-and-v8/`, 2021. Accessed: April 29, 2023.

[Ste23] Stellarium Developers. Stellarium. `https://stellarium.org/`, 2001–2023.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[SYT+20] Aastha Sainger, Rishikesh Yadav, Pradnya Tipare, Samidha Waghralkar, Vimla Jethani, and Amit Barve. Analysis of light pollution prediction using mathematical model and machine learning techniques. In *Advanced Computing Technologies and Applications: Proceedings of 2nd International Conference on Advanced Computing Technologies and Applications—ICACTA 2020*, pages 31–43. Springer, 2020.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Tea23] Data Hacker Team. Opencv: Average and gaussian filter. `https://datahacker.rs/opencv-average-and-gaussian-filter/`, 2023.

[TL19] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[Tre73] Patrick J Treanor. A simple propagation law for artificial night-sky illumination. *The Observatory*, 93:117–120, 1973.

[Ure82] Vasile Ureche. *Universul (astronomie) - volumul 1*. Editura Dacia, Cluj-Napoca, Romania, 1982.

[Vis18] MJP Vis. History of the mercator projection. B.S. thesis, 2018.

[VVG+16] Antanas Verikas, Evaldas Vaiciukynas, Adas Gelzinis, James Parker, and M. Charlotte Olsson. Electromyographic patterns during golf swing: Activation sequence profiling and prediction of shot effectiveness. *Sensors*, 16:592, 04 2016.

[Wal73] Merle F Walker. Light pollution in california and arizona. *Publications of the Astronomical Society of the Pacific*, 85(507):508, 1973.

[WPZ+19] Yuanchao Wang, Z. Pan, J. Zheng, L. Qian, and Li Mingtao. A hybrid ensemble method for pulsar candidate classification. *Astrophysics and Space Science*, 364, 08 2019.

[XJL19] Likai Xu, Jie Jiang, and Lei Liu. Rpnet: A representation learning-based star identification algorithm. *IEEE Access*, 7:92193–92202, 2019.

[XLL+21] Renjie Xu, Haifeng Lin, Kangjie Lu, Lin Cao, and Yunfei Liu. A forest fire detection system based on ensemble learning. *Forests*, 12:217, 02 2021.

[XLZ+20] Xingyu Xue, Yi Lin, Qiming Zheng, Ke Wang, Jing Zhang, Jinsong Deng, Ghali Abdullahi Abubakar, and Muye Gan. Mapping the fine-scale spatial pattern of artificial light pollution at night in urban environments from the perspective of bird habitats. *Science of the Total Environment*, 702:134725, 2020.

[YLT+20] Zimin Yin, Xi Li, Fei Tong, Zhibiao Li, and Michael Jendryke. Mapping urban expansion using night-time light images from luojia1-01 and international space station. *International Journal of Remote Sensing*, 41(7):2603–2623, 2020.

[ZC18] Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, 2018.

[ZCLX20] Wei Zhang, Xiaohui Chen, Yueqi Liu, and Qian Xi. A distributed storage and computation k-nearest neighbor algorithm based cloud-edge computing for cyber-physical-social systems. *IEEE Access*, 8:50118–50130, 2020.

[ZHW+21] Georg Zotti, Susanne M. Hoffmann, Alexander Wolf, Fabien Chéreau, and Guillaume Chéreau. The simulated sky: Stellarium for cultural astronomy research. *Journal of Skyscape Archaeology*, 6(2):221–258, Mar. 2021.

[ZW19] Georg Zotti and Alexander Wolf. Stellarium 0.19. 0 user guide. Technical report, Technical report. Available online at github. com/Stellarium/stellarium . . . , 2019.

[ZZL+20] Dawei Zhang, Zhonglong Zheng, Minglu Li, Xiaowei He, Tianxiang Wang, Liyuan Chen, Riheng Jia, and Feilong Lin. Reinforced similarity learning: Siamese relation networks for robust object tracking. 10 2020.

[Śc13] T. Ścieżor. A new astronomical method for determining the brightness of the night sky and its application to study long-term changes in the level of light pollution. *Monthly Notices of the Royal Astronomical Society*, 435(1):303–310, 08 2013.