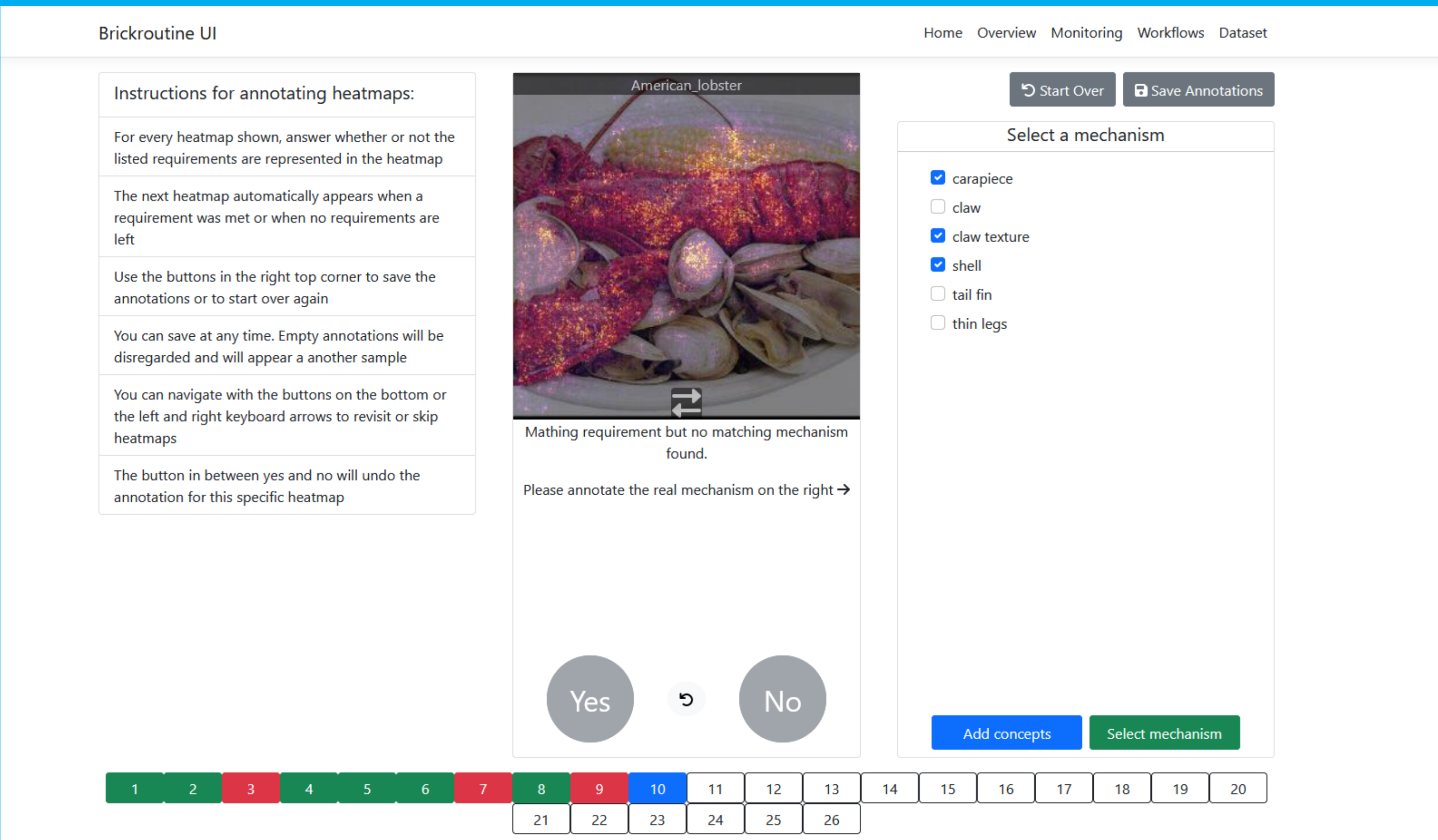
# Brickroutine

## A Human-in-the-Loop System for Interpreting Image Recognition Models

A.H. Ziengs

### System Design and Implementation



# Brickroutine

## A Human-in-the-Loop System for Interpreting Image Recognition Models

by

A.H. Ziengs

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday June 1, 2022 at 1:00 PM.

| | | |
|---|---|---|
| Student number: | 4391799 | |
| Project duration: | March 17, 2021 – June 1, 2022 | |
| Thesis committee: | Prof. dr. ir. G.J.P.M. Houben | TU Delft, chair |
| | Dr. Y. Yang | TU Delft, supervisor |
| | Dr. L. Miranda da Cruz | TU Delft, supervisor |
| | Dr. N. Yorke-Smith | TU Delft |



TUDelft

# Preface

I vividly remember the day when my parents got their first computer. A colossal second-hand machine that ran Windows 3.11. As an 8-year-old boy, I was given the task of explaining to them how to use it for basic tasks. My affinity with and interest in these machines kept evolving at a pace that was proportional to the development of these machines themselves. Years later, I found myself disassembling them, unscrewing the mainboard and copying shell commands from the internet, something that only was available at my former high school.

In hindsight, it is easy to say that computers were my calling and that I should have done something with them professionally as soon as an educational path allowed me to do so. Yet, driven by ignorance and stereotypes, I chose to study technology management after I finished high school. Years later, when I completed my first bachelor's at age 22 and when I got my first of many existential crises, I did something which was perceived as odd and risky by many. I enrolled for a second bachelor's degree, Computer Science at Delft University of Technology. It took me only three weeks to realize two things. First, it was not gonna be a walk in the park and secondly, it indeed was my calling and never felt so intrinsically motivated for something to put the effort in.

Fast forward to 2022, countless lectures, exams, nervous presentations, houses, and a pandemic later. While already working a full-time job, this thesis marks the completion of my master's degree. I am grateful to have done it in a way that taught me invaluable skills that I will benefit from for many years to come. Therefore, I would like to use this preface to thank some people that enabled me to do so. First, *Jie Yang*, who as a supervisor has always challenged and motivated me throughout the entire time that my thesis lasted. By encouraging me to step up when I needed to and slowing me down when I was drowning in the details, you have been an enthusiastic supervisor that I would recommend to anyone. Next, *Luís Cruz*, who has taught me a lot about academics. I am incredibly grateful for your critical and honest questions that always got me thinking about the narrative of my thesis and constantly kept reminding me of the importance of the bigger picture. The last one that was part of my recurring meetings is *Agathe Balayn*, for who I am thankful to have learned a lot from. With me working on something that intersects with your PhD, you always have been committed to providing me with extensive feedback (and numerous moments when I thought *"ah yes, she has a good point"*) on short notice. From the moment I started my master's, I haven't been too thrilled about doing my thesis but the three of you made it a nice experience after all.

With computer systems, a solid foundation is of fundamental importance. This can be metaphorically said about life in the broader sense. After a quarter of my thesis, I moved from a studio apartment to a house with friends. I don't know how I would have survived without this adventure, which initially would be just for a brief term. I want to give a big shout out to my housemates who always cheered me up when I had enough, made dinner so that I could do marathons of coding and accompanied me in doing sports and other activities that relieved some stress and pressure. This often resulted in me working even harder the next day. *Lars, Nico, Abe, and Jesse*, your contributions to my positive thesis experiences are greater than you probably imagine and I will forever cherish the great memories of my final year as a student. Speaking of a good foundation, I can hardly find the appropriate words to describe my gratitude towards *Isabel*, my girlfriend. Sacrificing loads of quality time to allow me to compensate for my lack of organisational skills, asking the right questions and always providing a listening ear can be seen as an accomplishment by itself.

I would like to conclude this in dutch with some words for my parents. *Papa en Mama*, ik ben dankbaar voor alle ondersteuning die jullie mij in mijn Delftse jaren hebben gegeven, waardoor ik mezelf op vele fronten heb kunnen ontwikkelen. Ondanks jullie beperkte affiniteit met wetenschap en studeren, heb ik me atlijd enorm gesteund gevoeld en weet ik dat ik atlijd op jullie kan rekenen en daar ben ik trots op.

*Bart*
*Delft, May 22, 2022*

# Contents


**1 Introduction** **1**
1.1 Use Cases . . . 1
1.2 Contributions . . . 2

**2 Background** **3**
2.1 Interpretability . . . 3
2.2 General Debugging . . . 4
2.3 Debugging ML systems . . . 4
2.4 Fairness . . . 5
2.5 System Architectures . . . 6

**3 The Brickroutine System** **7**
3.1 General Objective . . . 7
3.2 User Stories . . . 9
3.3 Starting Point . . . 10
3.4 Heatmap Extraction . . . 12
3.5 Workflows Specifications . . . 12
3.5.1 Workflow 0: Requirement Elicitation . . . 12
3.5.2 Workflow 1: Requirement Validation Annotation . . . 13
3.5.3 Workflow 2: Mechanism Validation Annotation . . . 15
3.5.4 Workflow 3: Adding new requirements . . . 18
3.5.5 Workflow 4: Requirements Correction . . . 19
3.5.6 Workflow 5: The Overview . . . 21
3.6 Process Overview . . . 26

**4 System Design** **29**
4.1 Architectural Requirements . . . 29
4.2 Backbone: Docker . . . 29
4.3 User Interface: React . . . 31
4.4 API: .NET . . . 31
4.5 Storage: MongoDB . . . 32
4.6 Data Monitoring: Mongo Express . . . 33
4.7 Communication: RabbitMQ . . . 34
4.8 Heatmap Extraction: Python . . . 34
4.9 Comparison with Existing Solutions . . . 35

**5 Evaluation: Informativeness** **37**
5.1 Goals . . . 37
5.2 Experimental Setup . . . 37
5.2.1 Approach . . . 38
5.2.2 Metrics . . . 38
5.3 Experimental Results . . . 39
5.3.1 First round . . . 39
5.3.2 Second round . . . 41
5.3.3 Third round . . . 42
5.3.4 Fourth round . . . 43
5.4 Overview . . . 43
5.5 Discussion . . . 44

**6 Evaluation: Validity** **47**
6.1 Goals . . . 47
6.2 Experimental Setup . . . 47
6.2.1 Approach . . . 50
6.2.2 Metrics . . . 50
6.3 Experimental Results . . . 51
6.3.1 First round . . . 51
6.3.2 Second round . . . 54
6.3.3 Third round . . . 55
6.3.4 Fourth round . . . 56
6.4 Overview . . . 56
6.5 Discussion . . . 58

**7 Discussion** **61**
7.1 Requirement-first vs Concept-first . . . 61
7.2 Multiple Requirements Annotation . . . 62
7.3 Differences Between the Experiments . . . 62
7.4 Iterative Approach . . . 63
7.5 Usability . . . 64
7.6 Future work . . . 64

**8 Conclusion** **67**

**A Results for sea creatures** **69**
A.1 Results for sea creatures round 1 . . . 69
A.2 Results for sea creatures round 2 . . . 70
A.3 Results for sea creatures round 3 . . . 71
A.4 Results for sea creatures round 4 . . . 73

**B Results for Birds** **75**
B.1 Results for birds round 1 . . . 75
B.2 Results for birds round 2 . . . 78
B.3 Results for birds round 3 . . . 82
B.4 Results for birds round 4 . . . 87

**C Proposed Improvements for Brickroutine** **95**

**D Readme from Code Repository** **97**

# 1
# Introduction

Nowadays, the term Artificial Intelligence (AI) is commonly seen in newspapers, television, universities, and businesses. This rather broad term refers to the ability of computers and machines to perform some tasks in such a way, that it resembles human intelligence. As a subset of this area, Machine Learning (ML) is characterized by using computational methods that use past information available to make accurate predictions. It consists of designing efficient and accurate algorithms to facilitate these predictions [22]. These predictions vary in complexity and can be anything ranging from forecasting future events to an application for self-driving cars.

However, humans face barriers in trusting the models and verifying whether or not they behave in reasonable ways when deployed [25]. This becomes increasingly problematic when signs of unfairness or biasedness arise and model decisions affect humans. The consequence of this phenomenon is that certain groups face discrimination, either implicit or explicit [12]. To assess the success of machine learning systems, other aspects besides performance matter. Criteria such as safety, nondiscrimination, avoiding technical debt or, providing the right to explanation are also seen as important and a condition for using these ML systems in a real-life fashion [12].

As a part of modern Machine Learning, computer vision algorithms have become more ubiquitous in the last decade. Machine learning systems have an increased performance edge over humans for some tasks [12]. This achievement can be mainly contributed to the developments in Deep Learning, which makes use of multiple-layer neural networks to make predictions.

The variation in the complexity of ML models has consequences for the extent to which humans can explain resulting predictions. For the simpler models such as K-Nearest Neighbors and Decision Trees, we can have an intuitive feeling about the predictions [22]. For a deep neural network, however, the specific features at each layer make explaining the output hard to impossible for humans. This situation is oftentimes called the black box problem in AI [13]. Developers can track the intermediate states of the mathematical calculations but are blind to the inner workings of the model.

In many use-cases, during the development or execution of these predictions, artificial intelligence applications are combined with human intelligence. In the case of computer vision applications, crowd workers are commonly used to annotate images as input for supervised learning algorithms. Additionally, the term human-in-the-loop (HITL) machine enjoys an increasing number of publications over the last decade [30]. HITL refers to a strategic approach that combines human and machine intelligence. [23] The goals of applying HITL can vary. Next to providing input to increase efficiency or maximize accuracy, developers of AI systems have a fundamental responsibility to facilitate the privacy and fairness of these systems. To assess the fairness, we need ways that allow us to debug and explain these systems.

## 1.1. Use Cases

Consequently, the problem that arises is that both end-users and developers of a Machine Learning system have difficulties defining the concept of *interpretability* and therefore experience problems tracking down model decisions. This is especially problematic when the system displays unexpected behaviour and the root cause of the bug is unknown. To define this concept in the scope of this project, a few use-cases are illustrated below.

- Given a neural image classifier that is trained on a bird data set, a developer of AI models wants to see why a certain erroneous prediction of a bird species was made. He wants to explain this in terms of concepts that a human also would use to describe such a problem (e.g. Which part of the birds makes the algorithm confuse bird $y$ and $z$ with each other?).
- Given a requirement in human language for an entity to classify as something we all understand, and that is along the line of reasoning of humans (e.g. a kitchen contains a sink and an oven), end-users of AI models want to verify that this requirement (amongst others) is in fact used to classify a certain image and that the correct predictions are not merely coincidental.
- Given a neural image classifier that is used in the medical domain to detect anomalies on an x-ray, doctors using this application would like to know why the model predicted a medical condition that they cannot verify with their domain knowledge.

In short, we can say that we want to interpret the model and that it is becoming increasingly harder to debug an ML model when the complexity increases and this matters more when ethical factors come into play. Such systems are supposed to be fair so that they can be trusted by people that are subject to their predictions. Moreover, since the input to these systems is ever-increasing, we want the systems to be robust so that they always keep these fair and trustworthy characteristics. In their Standard Glossary of Software Engineering Terminology [10], IEEE defined robustness as *"The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions"*. Following up, the problem can be described as:

> *"Humans have limited possibilities to disassemble a machine learning model and verify if the algorithm followed a line of reasoning that is comprehensible so that decisions can be assessed in terms of fairness, robustness, and trustworthiness"*

## 1.2. Contributions

A partial answer to the problem statement is given by the work of Balayn et al [2]. In their work, a human-in-the-loop machine learning method is presented that explains the inner mechanism of a trained model in terms of human-comprehensible concepts. This thesis is an extension of that research and will be conducted in close collaboration with the authors. In their work, it is shown how to understand the model. The goal and main contributions of this work are to subsequently compare it to what a human would expect and have a complete ready-to-use system. The eventual contributions include:

**Contribution a** Functional and Non-Functional requirements elicitation for such a system (section 3.2).

**Contribution b** A mapping of the contextual requirements into the required workflows (section 3.5).

**Contribution c** A mapping of the activities of such a system into an architecture that fits a known architectural paradigm (chapter 4).

**Contribution d** An implementation of this software.

**Contribution e** An extensive evaluation of this implementation (chapter 6 and chapter 5).

# 2
# Background

The problem defined in our problem statement revolves around wrong outputs and wrong features for Machine Learning (ML) systems. This chapter reviews the existing work on fairness and interpretability and the fundamental difference between debugging traditional software systems and Machine Learning ML systems. Additionally, relevant work on (ML) software architectures is covered. Since the goal and contributions of this thesis are relatively new, we list resources that are inspirational but do not necessarily compare to the envisioned outcome of this thesis. Interpretability and fairness are hot research topics nowadays [32]. Because we want to interpret the behaviour of our model with the eventual goal to assess, amongst other things, fairness, we look for existing work there that serves as background knowledge. Compared to general software testing, ML testing encompasses more complexity because it may not only occur in the code but in the data as well. In order to grasp the problem in its entirety, we assess how debugging machine learning systems differs from traditional software debugging.

## 2.1. Interpretability

Numerous methods for interpretability exist. Traditionally, existing tools are organized in such a way that they can be interpreted by an explanator that is easy to comprehend. These methods are local, meaning they concern individual samples. Common techniques are decision trees, feature importance and heatmaps [33].

Moving on to global (explaining the network as a whole [33]), in [14], ACE, an algorithm to extract visual concepts (after training) from images is presented. In this work, simple to more complex concepts in the form of image patches are extracted that eventually are part of the classification task of the algorithm. This work does not require human supervision to be executed but does rely on individual perception with respect to interpretation of the results. Additionally, in the case users want to test a predefined hypothesis about the inner workings of the model, the input images require that these concepts are explicitly present.

In [2], which is the foundation for this thesis, a human-in-the-loop machine learning approach that used crowdsourcing is presented. Crowd workers are used for annotating saliency maps so that eventually, model behaviour can be explained with concepts that are understandable to humans. The outcomes are evaluated against ACE [14] and with a relatively little amount of annotated images, human-comprehensible interpretations could be done by the model. As in earlier attempts at interpretability [25], statistical testing is used to quantify the performance.

From both the academic and corporate world a few comparable systems have originated:

- Snorkel[1], to programmatically build training data. This project has been integrated with the recently emerged snorkel.ai[2] project into an entire suite that allows for labelling, deploying, monitoring and more. Compared to our proposed system, however, snorkel is more to accelerate workflows, whereas our system is explicitly aimed at interpretability.

[1] https://www.snorkel.org/
[2] https://snorkel.ai/

- AI Explainability 360[3]. To comprehend models' predictions in multiple stages of an AI pipeline. This system has similar goals compared to ours but is aimed at more mathematical approaches such as boolean decision rules and linear models applied to various types of data. Our system aims at the interpretability of image recognition models with semantic concepts.
- Error Analysis [4], an API set for python that allows for a visual explanation of errors for both identification and diagnosis. After testing the models, errors are analysed. The difference between our problem and system is that we do not have a concise definition of an error. In our case, we want to interpret the results with semantic concepts, something that is not limited to incorrect predictions of the model.

## 2.2. General Debugging

In general software development, programs display a deterministic character. In an arbitrary programming language, given a fixed input, results are identical when executing the code multiple times. This has consequences for the scope of the debugging workflow. In their standard glossary of software engineering terminology [10], IEEE describes the definition of debugging is as follows:

**Definition.** ***Debugging*** *is the process of locating and correcting errors in a program in which errors have been detected*

Subsequently, the authors present a debugging process model:

1. Initial source code with a hypothesis about the outcome that is not verified by executing the current code.
2. Narrowing down the region that is likely to cause the problem
3. Modification of the source code so that the expected outcome could be reached
4. Verify the hypothesis, if not, go back to step 1.
5. The bug is located

A more systemic approach is presented in [20] where the relationship between using assertions and faults in software programs is investigated. In this work, a slight statistical relationship between the assertion density and the fault density is shown. In an IEEE paper, assertions have been defined as *Formal constraints of software system behaviour that are commonly written as annotations of a source text. The primary goal in writing assertions is to specify what a system is supposed to do rather than how it is to do it* [27]. This is mostly implemented by having a Boolean predicate in the source code which should evaluate to true in order to ensure a correctly working software and raises an error otherwise. The purpose of this mechanism is to locate bugs, both during compile-time and run-time.

## 2.3. Debugging ML systems

Traditional software is deterministic in nature whereas most ML algorithms possess stochasticity. This stochastic element makes debugging more difficult because tests and assertions cannot test for fixed values. Additionally, the inner workings of most ML models are abstracted and invisible to developers since these techniques are mostly black boxes. The working of algorithms is implicitly learned from the data rather than explicitly specified in the source code. The applicability of an ML algorithm is measured with statistical tools such as accuracy and area under the curve (AUC). For less complex algorithms such as Bayesian networks and support vector machines, visual tools exist to explain the models' behaviour to some extent [19].

In recent years, many approaches have been taken to debug ML systems and make them more interpretable. In [18], it is shown that assertions in ML systems can be used for run-time monitoring by logging unexpected behaviour or triggering corrective actions. Moreover, model assertions can find errors with a high degree of confidence. Due to the aforementioned stochasticity, which is not a certain

[3] https://aix360.mybluemix.net/
[4] https://erroranalysis.ai/

error but a statistically high sign of unexpected behaviour. The implementation of these assertions is done in a python library and allows developers to write assertions with a high level of abstraction. Example use cases are the analysis of TV news, autonomous vehicles, video analytics and medical classification.

The need for testing the entire ML pipeline is justified in [5] where 28 needs that can be modelled as an assertion are presented. These are placed into four separate categories; *data, model, infrastructure* and *monitoring*. Each category lists seven tests and the total score is the minimum score that is obtained for each category. The minimum is chosen because all aspects are regarded as equally important for a solid system. The problem statement from section 1.1 relates to the concept of debugging for the sake of interpretability and [2] distinguishes use cases for debugging.

- *Exploratory:* developers of a model want to comprehend the model for fine-tuning and evaluation purposes without searching for a specific answer (e.g. "How does a model `A` make its decisions given a certain input and hyperparameters" as opposed to "Why predicted model `A` outcome `Y` given input `X`?")
- *Explanatory:* developers or end-users want to verify why a model did a certain prediction on a given input (e.g. "Why predicted model `A` outcome `Y` given input `X`?"). In this case, there's a distinction between *global interpretability* (rule construction based on a set of input entries) and *local interpretability* (individual input entries).

Both scenarios are important because they serve as a use case for which the output of our system could be utilized.

## 2.4. Fairness

As a non-functional characteristic that becomes more important nowadays [32], fairness is characterised as *the degree to which AI systems don't have an unwanted algorithmic bias*. In [9], multiple definitions are addressed that relate to treating every individual equally and not giving a less qualified individual an edge over a qualified one.

Back when still in its infancy, ML was used for seemingly innocent applications such as online ads or filtering emails. Nowadays, it contributes to filtering loan applicants, deploying police officers and diagnosing diseases. In the last years, a vast amount of research is done on the subject of fairness because it is suspected that the usage of these algorithms can introduce discrimination [9]. One study found that classifiers for face detection performed better for people with white skin compared to people that are dark-skinned [6]. Additionally, one study shows that biases in language with respect to gender and race and cultural stereotypes are passed on to artificial intelligence [7]. For cases like loan applications and fraud, the algorithms are trained on fraudulent samples. When the sample set is unbalanced with respect to something that can lead to discriminatory practices, it becomes bias prone.

Previous works lists five reasons that cause unfairness as follows [3]:

- Skewed sample
  When a small initial bias exists, the errors may accumulate over time, hence increasing bias even more.
- Tainted examples
  When data is erroneously labelled by humans as a result of a subjective bias.
- Limited features
  The lack of significant features may cause the model to not find an adequate relationship between the features and the class labels.
- Sample size disparity
  Occurs when the sampled data leans more toward one specific class.
- Proxies
  There are cases where features are an indirect pointer to sensitive features that are subsequently learned by the model (e.g. an address in a certain neighbourhood might eventually cause a bias towards skin colour), even though the sensitive features themselves are omitted.

More specifically, Krause et al. describe how inequality in the real world inevitably can lead to a bias that is encoded in the data [9]. Furthermore, since ML algorithms reduce the average error in their training phase [22], it can cause only majority populations are fitted to the model, leading to a bias towards minority populations. In our work, we aim to contribute to fairness by looking at the inner working of the model and by designing user interfaces, that help understand the models' performance for certain categories of the data.

## 2.5. System Architectures

Since one of our contributions will be a concrete implementation, the current state of system architectures with relevancy for our context will be addressed in this section. A trend of the last decade is that systems are deployed in the cloud. Big cloud providers such as Amazon, Google and Microsoft leverage this trend by offering entire platforms such as AWS[5] or Azure[6]. These infrastructural changes triggered a change in architectural styles. Microservices architecture implies that small loosely coupled services are each responsible for contributing a small part (hence *"micro"*) to an application [11]. The counterpart of microservices is the traditional monolithic architecture where all functionalities are combined and bundled into one application that is executed during run-time. Monoliths still have advantages over micro-services such as rapid development, better testability and ease of deployment. The combination of a microservices architecture on a cloud computing platform results in key selling points such as flexibility with respect to deployment, being language-neutral and modularity.

The microservices architecture is commonly implemented using Docker [7] that uses a virtualization platform to run software in so-called *containers*. A docker container is an isolated environment that runs most programming languages. A software program is executed in a container that ensures all the necessary requirements with respect to containers. The host system does not have to support the programming languages. In [17], the usage of docker within a microservices architecture is justified. Numerous reasons are given, docker is suitable for microservices because it accelerates; *automation*, *independencies*, *portability* and *resource utilization*.

[5]https://aws.amazon.com/
[6]https://azure.microsoft.com/
[7]https://www.docker.com/

# 3
# The Brickroutine System

Following chapter 2, a requirements elicitation for the system, which from this point onward we will call *“Brickroutine”* is conducted in this chapter. This name is chosen because this system allows the end-users to see the building bricks of an AI model by using several routines. First, the general objective with respect to debugging and explainability is revisited and subsequently, the necessary actions and procedures will be explained. The use cases for this system stem from existing research on the explainability of AI systems from Balayn et al. (2021) [2]. The nature of the typical user of this system can vary. We envision it to be one of the following:

- *Developers* represent the people involved with the development of the AI model that is used to classify images. They want to use this system to know which concepts are learnt by the model. This provides them with the knowledge that they could use to modify the model.
- *Domain experts* represent the people involved with the provision of the initial requirements for classification. When images stem from a category that requires specific knowledge with respect to the semantic concepts, these people could use the system.

First, we will specify the general objective of Brickroutine and introduce terminology in section 3.1, then the sets of actions needed to achieve this objective will be discussed in section 3.5.

## 3.1. General Objective

Consider the following situation: Given a neural image classifier that is trained on a birds data set, a developer of AI models wants to see why a certain erroneous prediction of a birds species was made. He wants to explain this in terms of concepts that a human also would use to describe such a problem (e.g. Which part of the birds makes the algorithm confuse bird $y$ and $z$ with each other?).

Suppose bird $y$ has yellow wings and a short pointy beak and bird $z$ has black wings and a red dot on its head. If an ornithologist would see a bird with a red dot, the concept of the red dot is used to classify that bird as bird $z$ because there is a relationship between that concept and the class the bird belongs to.

**Definition.** *A **concept** is a semantic interpretation of a visual characteristic of an object that is used to classify that object. Concepts may be composed of other concepts and can be used in combination with other concepts. The ML model internally establishes the relationship between concepts and classes. With the right tools and techniques, the parts of an image that an ML model used to classify could be observed in a heatmap.*

**Definition.** *A **heatmap** or **saliency map** is an image that highlights the pixels that were used in each layer of a neural network to come up with the prediction of that specific class. It features the original image with an overlay that represents the relatedness to the prediction of each pixel. It is used so that annotators can connect those regions with semantic concepts.*

The eventual goal and contribution of the system is to *test a Machine Learning model on the relationships between concepts that it has learned*. We want to verify if it has established the same relationship as humans would. Therefore, the correct relationship between the concepts and classes is a requirement.

**Definition.** *A **requirement** is a set of concepts that ultimately can be verified by our system. These requirements are initially submitted to Brickroutine and can change over time. For example, when humans see a sink and an oven, they would commonly identify the scene as a kitchen. Consequently,* $\mathtt{oven} \wedge \mathtt{sink}$ *is a requirement for* $\mathtt{kitchen}$*.*

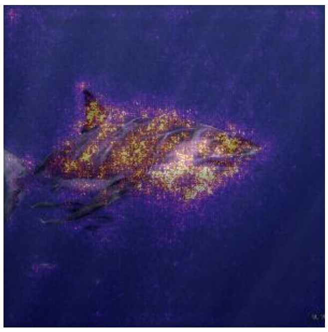
Ideally, we want to verify a complete list of requirements. For instance, what differentiates a kitchen from a bathroom in human-understandable language? A human would argue that both feature a sink but the combination sink and oven would only be present in a kitchen and would rather make an odd bathroom. These requirements typically originate from a person that is skilful on the topic that data set is about, a so-called expert in a specific domain.

Following section 2.3, ML models have no cognition of concepts when classifying due to their black-box nature. It recognizes patterns and classifies an image with a certain likelihood when previously learned patterns are present in that image. In [2], the pixels of an image that the model uses to classify objects are highlighted with saliency maps. When subsequently the concepts belonging to these parts of the images are annotated with semantics, concepts and relationships between them can be explained. As a result, the inner workings of the model can be described by what we call mechanisms.

**Definition.** *A **Mechanism** is an approximation of the relation between concepts that results from a trained model. After training, the model has implicitly learned a definition. For instance, when Brickroutine helps us conclude that for some images that belong to the class kitchen, the model learned patterns that humans would conceptualize as an oven, a sink and a chair respectively. A mechanism would be* $\mathtt{oven} \wedge \mathtt{sink}$ *for the class* $\mathtt{kitchen}$*.*

Figure Figure 3.1 shows visual examples of the afore-mentioned terminology. Figure 3.1a shows the input image of a shark, the requirement for this image is what concepts initially are thought to be defining a shark, for instance, a dorsal (top) fin, a pointy snout and a caudal (back) fin. Figure 3.1b shows the heatmap that can be used to approximate the real mechanism, this mechanism is retrieved by annotation in our system and in this case represent the concepts dorsal fin, snout, mouth, grey skin and a small part of the ocean, although this is depending on the perception of the annotator. We can see in the images that the smaller fishes surrounding the shark are not picked up by the model.

(a) input image of a shark to test requirements against



(b) heatmap of a shark to annotate the mechanism

Figure 3.1: pictures of a shark to demonstrate the terminology

Additionally, concepts can be (and often are) made from other concepts. When a step is taken towards finer-grained concepts, other concepts appear. The lowest granularity of concepts that can be distinguished in this context will be referred to as seed concepts.

**Definition.** *A **seed concept** is a concept that is at the lowest granularity. A table for instance is made up of multiple legs and a tabletop, the legs and tabletop, in this case, identify as seed concepts. Taking a step towards concepts of a higher granularity, seed concepts can form intermediate concepts that should eventually lead to a class for a requirement or mechanism.*

We see that the definitions of requirements and mechanisms closely resemble each other. The subtle difference lies in the fact that requirements are what we define as "how a human would look at an object and determine which class it belongs to". Mechanisms, on the other hand, are how an ML-model looks at it. Revisiting the general objective of Brickroutine, *test a Machine Learning model on the relationships between concepts that it has learned*, we want to compare the initial requirements with the learned mechanisms to infer the inner workings of the model. For instance: Is the mechanism that the model uses along the line of reasoning of humans, i.e. do the requirements and the mechanisms overlap? Moreover, if they differ, which mechanism is used instead and is this valid?

Let us imagine a simple world where one procedure answers the problem described above. Roughly we could split up our procedure into three steps:

- Get the requirements from the domain experts
- Let Brickroutine extract the mechanisms from the heatmaps.
- Compare and conclude

However, there are a couple of matters that make this more complex. First, a cold start problem arises because the requirements have to be entered manually by the user and there is no guarantee that the concepts in these requirements match concepts learned by the model. A result could be that mechanisms could in the best case only be partially overlapping with requirements or even have no concepts in common at all . Lastly, we cannot know the actual level of granularity of the mechanisms in advance. Going back to our example, does the model see an actual table as a table or as a combination of four legs and a tabletop? To cope with these phenomena, the routine of identifying, verifying or modifying appropriate requirements and mechanisms is split up into a different sequence of actions, which we will call a workflow.

**Definition.** *A **workflow** is a sequence of actions with the goal of identifying, verifying or modifying the requirements and mechanisms. Each workflow has a separate goal as a part of the general objective. The execution of the actions in the workflow is either initialized by the user or other workflows. For most workflows, conditions exists that should be met before the workflow can be executed.*

**Definition.** *A **condition** indicates that the combination of images, heatmaps and annotation should be in a certain state before the user can start this workflow. An example of this is that there should be images available in the system when the users wants to annotate them. These conditions have been identified in the descriptions of our workflows in section 3.5.*

## 3.2. User Stories

To account for the functional requirements of the system, it is helpful that the requirements are described from the perspective of the end-user [4]. As initially stated in this chapter, the end-users of this system will fall into two categories: developers and domain experts. Developers can also function as both, under the condition that they are sufficiently informed about the semantic characteristics of the classes that belong to the images of the submitted data set.

First, we define the user stories from the perspective of a *developer*:

- As a user I want to interpret my computer vision models so that I can know which semantic concepts of my images the model reasoned on for correctly classified images and I can verify if this is valid.
- As a user I want to interpret my computer vision models so that I can know on which parts of my images the model reasoned for incorrectly classified images and I can take appropriate actions.
- As a user I want to upload a file with predictions that an ML model made so that I can see the predictions in my system.
- As a user I want to upload a file with predictions that an ML model made so that I can compare my semantic concepts analyses to the predictions.
- As a user I want to upload the original images that served as input for the ML model so that requirements can be annotated.

- As a user I want heatmaps to be automatically extracted when I upload the images so that I can see which parts of my original images my model used to predict.
- As a user I want to be able to upload heatmaps when they are generated elsewhere so that I can see which parts of my original images my model used to predict.
- As a user I want to upload multiple data sets and predictions and easily switch between them so that I can use the system for multiple combinations of data sets and model results while maintaining the same storage mechanisms.
- As a user I want to have an overview of all the classes and requirements with a table that shows metrics for respective images in my data set so that I can interpret my model in terms of human-understandable semantic concepts.
- As a user I want to have an overview of all the classes and requirements with a table that shows metrics for respective images in my data set so that I can interpret my model in terms of human-understandable semantic concepts.
- As a user I want to have a user interface so that I can use the system, execute actions and see the output of my annotation work

Subsequently, we define user stories from the perspective of a person doing the annotations, the so-called *domain expert*:

- As an annotator I want to submit requirements for each class so that I can verify if these requirements are present in the images that I upload
- As an annotator I want to be able to verify these requirements in the heatmaps so that I can assess if my model used these requirements to make predictions and understand my model.
- As an annotator I want to enter custom mechanisms for each image when my requirements are not verified so that I can understand my model.
- As an annotator I want to enter custom mechanisms for each image when my requirements are not verified so that I can understand my model.
- As an annotator I want to select how many images for each class I want to annotate so that I can work efficiently towards my goals and do not have to annotate unnecessary images.

## 3.3. Starting Point

Now that we have established a general objective and functional requirements, we can describe our workflows step by step. Our approach is to present the workflows in conjunction with the interface design because of the coupled nature of those two parts. The initial feature that we describe and implement is enabling the user to upload a data set. A data set suitable for our system consists of the following parts:

- A set of images that the model user to make predictions.
- A file with file names predictions. The format we ask the user to use for their csv files is: *image name*, *true label*, *predicted label*.
- A set of heatmaps for every image that is being uploaded (optional).

The interface for uploading a data set is shown in figure Figure 3.2. Here we see input fields with explanations about the headers and heatmaps. The form is equipped with client-side validation to sanitize the inputs at this point.

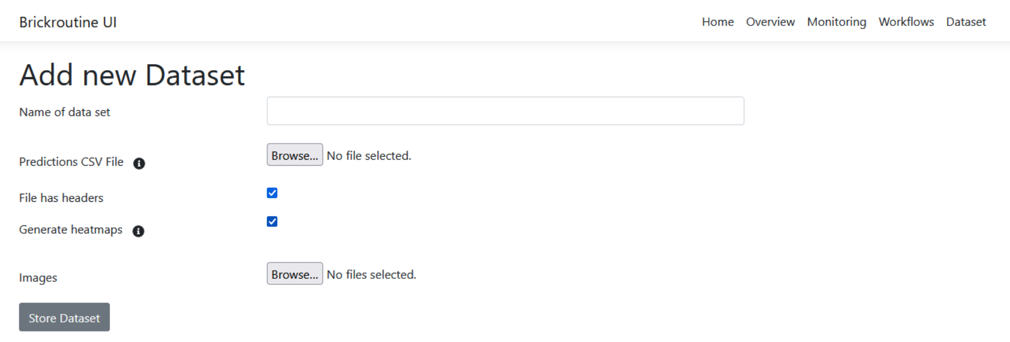



Figure 3.2: The user interface for uploading a data set

If we recall our functional requirements from the user stories, we do not simply design Brickroutine for just one data set. After uploading at least one data set, the user can select their designated data set by using the controls and the top and mark the data set as *active*. From this point, all statistics and actions will be designed for that specific data set. Since the user uploaded the model result, we present an overview per class of the accuracy and a counted list of the predictions to have the statistics in the system and aid users in coming up with scenarios that they might want to verify in subsequent steps. Figure 3.3 shows the interface design of the above-mentioned parts.

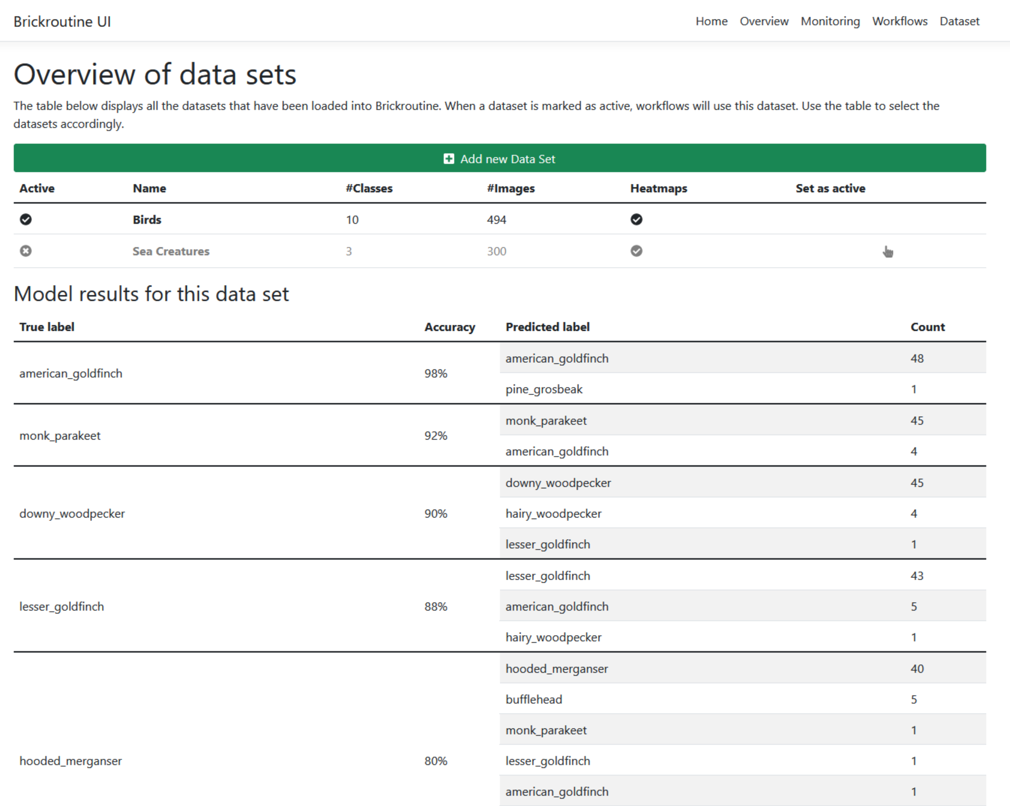



Figure 3.3: The overview of a data set

After uploading the images and while the heatmap generation process is triggered, the users can see the state of their data set by navigating to the home screen by clicking on the *home* tab at the top of the page. The interface is displayed in Figure 3.4. The expandable table at the top of the page shows the users the requirements per class and the per image state the numbers. In subsection 3.5.6, we will elaborate on this table after we have described Brickroutine in more detail. Below the table is a list (*User actions for this data set*) that shows the user which steps are done and still need to be done. Not in the picture is a detailed visual description of a typical flow in Brickroutine, which is covered in section 3.6.

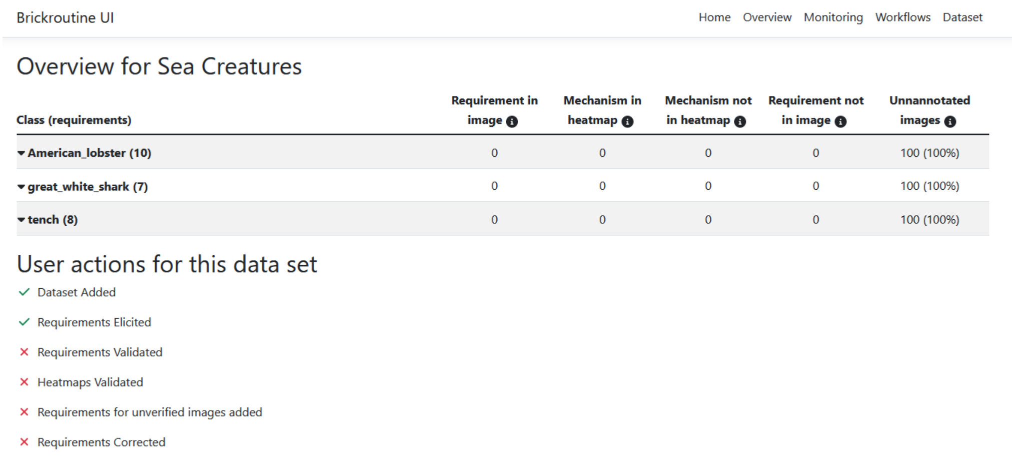



Figure 3.4: Brickroutine home screen

## 3.4. Heatmap Extraction

As mentioned in section 3.1, the heatmaps will help the user understand which specific parts of the image the model used to make classifications. The concept of heatmaps or saliency maps was first described in [28] and can be applied to convolutional networks. The output in the heatmaps should be interpreted as the relative importance of the pixels from the input image with respect to the predicted classs.

Currently, the model at the back-end of our system is fixed to Inception V3[29]. By first finding the derivative (using back-propagation) and using the predicted class as input for this function, we get the input values of our last layer (softmax for Inception V3) represented in the vectorized (one-dimensional) form. By transforming this back to the original image size, we get an approximation of the pixels that are used by the model. Finally, if we use these values as an overlay on the original image, the user can annotate which concepts are featured by the image at the places where the heatmap values are significant.

If the user selected the checkbox to generate the heatmaps within Brickroutine (Figure 3.2), the heatmaps will be extracted in the background. A detailed technical explanation of this procedure will follow in section 3.4. Having the heatmaps in Brickroutine is a condition for the mechanism validation annotation procedure to start. When the heatmaps are finished, the user sees this in the data set overview screen (Figure 3.3).

## 3.5. Workflows Specifications

In this section, detailed outlines and pseudocode of each workflow are given. For each workflow the condition, input and result are given.

### 3.5.1. Workflow 0: Requirement Elicitation

This workflow is concerned with obtaining the requirements for each classification that is possible in the data set it concerns. This is done by asking the domain experts to specify a list of requirements for each class. For each requirement, a weight is required. The weight is a decimal number between $0$ and $1$ and can be interpreted as *the likelihood of which an object of the respective class features the concepts that are part of this requirement*. A higher weight means a higher likelihood. Figure Figure 3.5 depicts the overview and Figure 3.6 shows how we implemented this weighted element in the user interface.

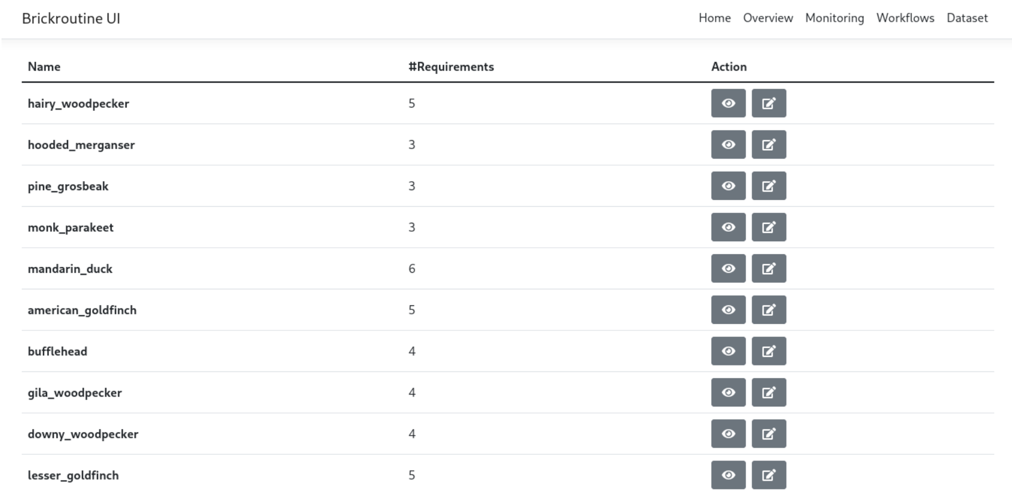



Figure 3.5: Workflow 0: Overview of classes

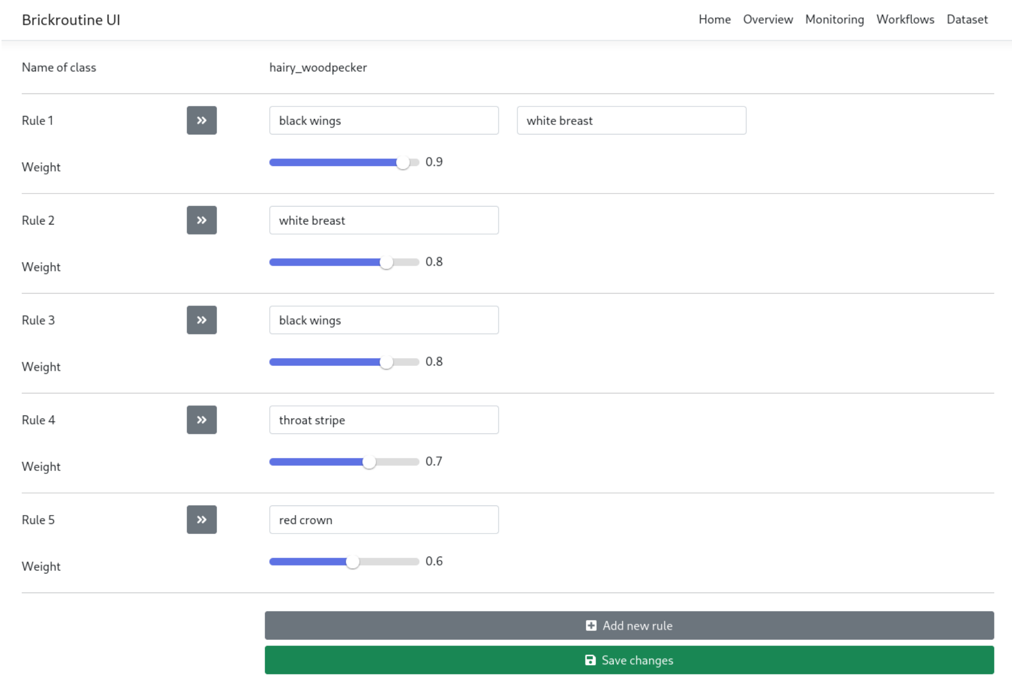



Figure 3.6: Workflow 0: Define and modify requirements for a class

### 3.5.2. Workflow 1: Requirement Validation Annotation

The goal of this workflow is simply to do an initial verification on an image level. A requirement is marked as verified when all the concepts of that requirement appear in the input image. The outcome of this workflow is used in subsequent steps. An outline in pseudocode is given in algorithm 1.

**Data:** Original images and requirements for each class
**Condition:** Unannotated images
**Result:** Requirements annotated in the images that belong to the sample

```
for All images from sample do
    for All requirements for this class sorted by weight in ascending order do
        if Requirement appears in image then
            Set this requirement as verified
            break
        end
    end
end
```

**Algorithm 1:** Pseudocode for Workflow 1

For this workflow, we can imagine that the user does not want to annotate all images at once. In fact, we think users should use Brickroutine in small iterations (more on that topic in section 3.6). Additionally, following the user stories, we want the users to select the desired amount of images to annotate for each class specifically. As a result, the user interface depicted in Figure 3.7 allows the user to control this.

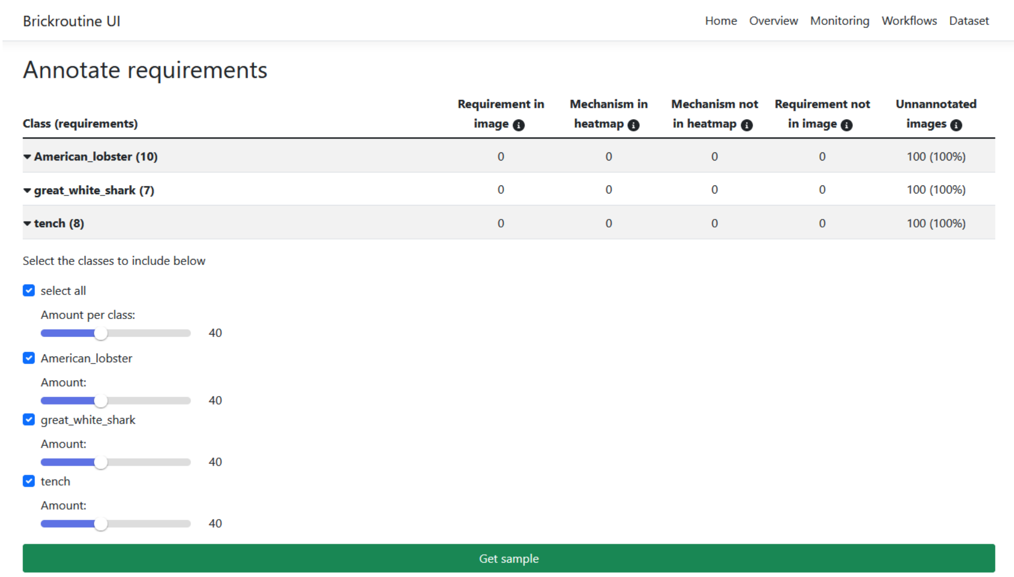



Figure 3.7: The sample selector for workflow 1

On the top of the image, we see the same expandable table as is on the home screen. Below are the controls that allow the user to either select all images with the same amount of images per class or a (de)select individual classes with specific numbers. When all numbers are appropriately defined, the users can hit Get Sample and the system shows the requirement annotation interface, which is shown in figure Figure 3.8.

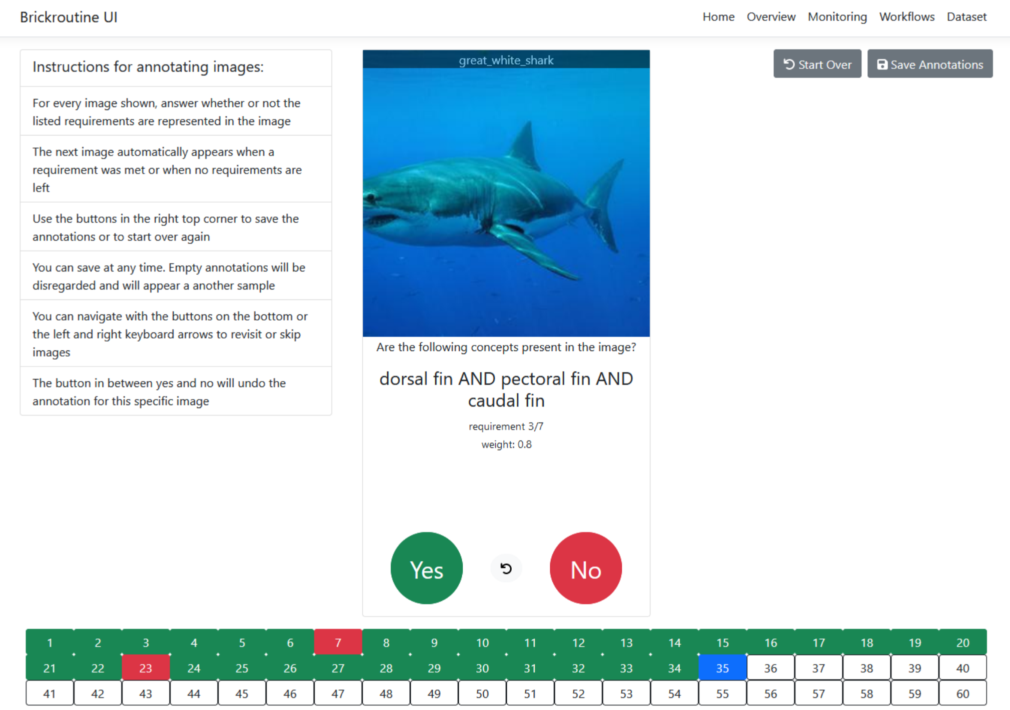



Figure 3.8: The requirement annotation interface for workflow 1

At the bottom, we implemented a control panel that turns green if a matching requirement (the shown concepts appear in the image) was present in the image, red when none was found, blue for the image the users are currently annotating and white for images that still have to be annotated. The user can jump between images by clicking on these buttons or using the arrow keys on the keyboard. In the middle, right to the instructions, we see the actual annotation component. It shows the concepts that are part of the current requirement, the sequence number of the requirement and the weight. The three buttons on the bottom are to give input with respect to the presence of a requirement and to undo the current image in case the user makes a mistake. To submit their annotations (this can be done before all images are annotated) the users use the button on the top right. The button to the left of the submit button can be used to set all the images back to the initial state.

Although it is possible that multiple requirements are present in an image, the pseudocode in algorithm 1 shows that we limit the user to annotating one requirement per image and proceeding to the next image when that requirement is found. Because all the requirements have weights attributed to them and the requirements are presented to the user in ascending order, the requirement with the highest weight will appear first. In this way, annotation time is reduced. The weight of the requirements and therefore the order in which they are presented in this workflow can be adjusted in a subsequent workflow (subsection 3.5.4). Eventually, the mechanisms that are found by the system could be equivalent to these requirements.

### 3.5.3. Workflow 2: Mechanism Validation Annotation

The goal of this workflow is simply to verify the presence of the requirements on a concept level by leveraging the heatmaps. A condition is that every image that is addressed in this step has been through the previous workflow and that the heatmaps for this data set are extracted. An outline in pseudocode is given in algorithm 2. We ask the user if the requirement is entirely covered because the model might have learned concepts of a finer granularity when the concepts from the requirement are only partially covered. This information is used in the following workflows.

Contrary to algorithm 1, in algorithm 2, we take an extra step to check if the mechanism matches any similar requirement with lower weight. If in this stage, the mechanism proved to be similar to a requirement with a lower weight than the requirement that was annotated, we set that specific requirement as

**Condition:** Heatmaps present in system and requirements extracted
**Data:** Heatmaps and requirements for images
**Result:** Images with annotated mechanisms

**for** *All images from sample* **do**
  **if** *If all concepts of annotated requirement are highlighted by heatmap* **then**
    Mark this image as verified mechanism
    **if** *concepts are entirely covered* **then**
      mark this mechanism as entirely covered
    **else**
      mark this mechanism as not entirely covered
    **end**
  **else**
    Mark the requirement for this image as not verified in heatmap
    Select all relevant concepts as a custom mechanism
  **end**
  Execute algorithm 3 `// Check if the mechanism matches another requirement`
**end**

**Algorithm 2:** Pseudocode for Workflow 2

verified, both in the image itself and in the generated heatmap. We do this by the procedure highlighted in algorithm 3 where the `key` function represents a generated string based on alphabetically ordered concepts so that the requirement will be returned if all the concepts of the annotated mechanism are matching.

**Data:** Images with annotated mechanisms
**Result:** Images with optionally matched requirements

$C \leftarrow$ empty dictionary
**for** *All classes $c_i$ from the submitted images* **do**
  $R \leftarrow$ empty dictionary
  **for** *All requirements $r_i$ in $c_i$* **do**
    $R \leftarrow (\mathtt{key}(r_i), r_i)$
  **end**
  $C \leftarrow R$
**end**
**for** *All images $I_i$ from sample* **do**
  **if** $\mathtt{key}(\mathtt{mechanism}(I_i)) \in C$ **then**
    Set $C[\mathtt{key}(\mathtt{mechanism}(I_i))]$ as the verified requirement and mechanism for $I_i$
  **end**
**end**

**Algorithm 3:** Retroactively matching the mechanisms with requirements

For the images that the user annotates in this step, the requirements should be present in the system. Analogous to the previous workflow, we have the sample selector again. However, we allow selecting the user only images from classes that have one or more annotated requirements. Likewise, the maximum number is also bounded by the number of verified requirements. The user interface for this step is shown in image Figure 3.9. Here we can see the range of the sliders match the numbers in the table at the top and that the class *tench* is stricken through because the amount of verified requirements is 0 in the example.

The user interface for this annotation step is shown inFigure 3.10a. At the bottom of the page the users see the navigation pane again that can be controlled with either keyboard or mouse. On the bottom of the image in the centre, we implemented a button as an overlay (the double arrows) that allows the user to toggle between the heatmap and the original image in case the heatmap is limiting the user in determining which semantic concepts are behind the highlighted parts.

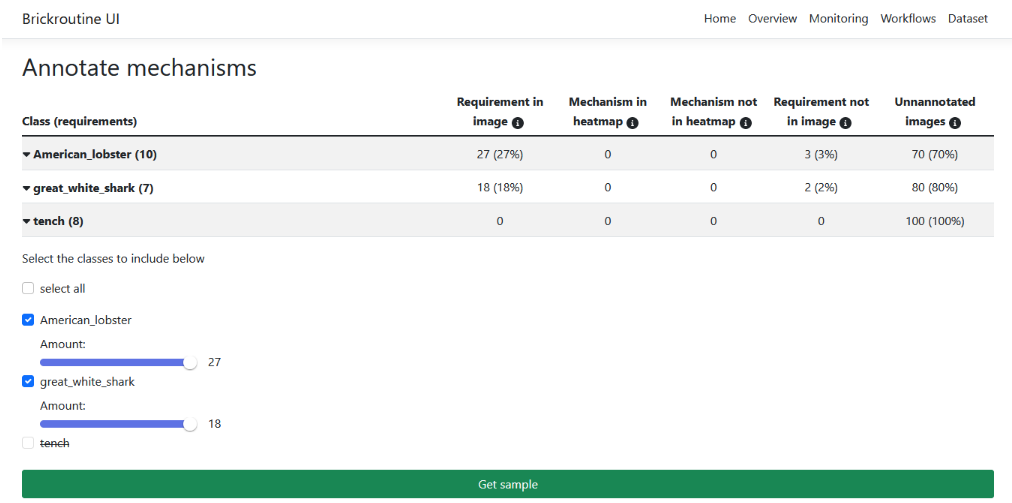
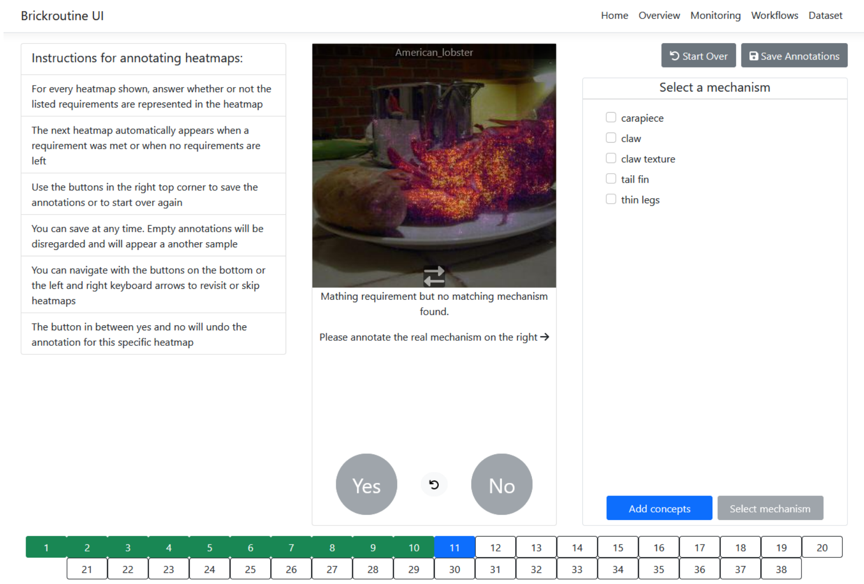
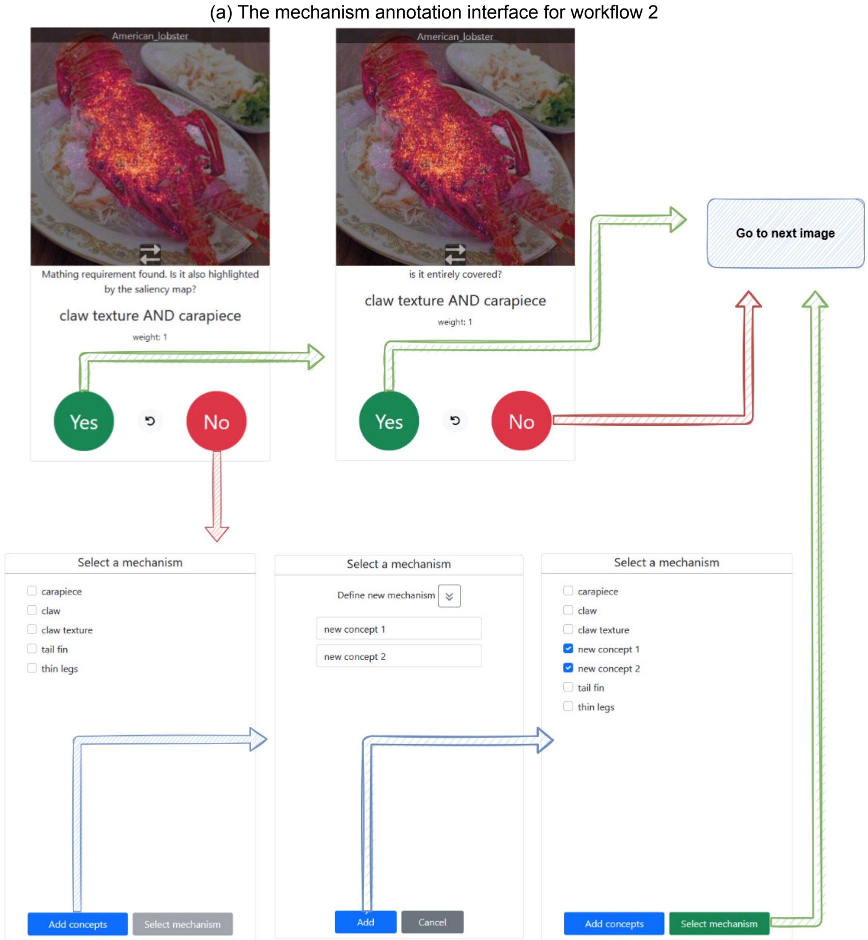



Figure 3.9: The sample selector for workflow 1

In addition to the previous annotation step, a single annotation during this workflow consists of multiple actions with a minimum of two. The minimal case is when the heatmap is highlighting the exact concepts of the requirement. After clicking yes, the user is asked if the requirement is entirely covered. When the mechanism is not matching the requirement, we ask the user to annotate a custom mechanism (algorithm 2, line 11). For annotating this custom mechanism, we present the user with all the distinct concepts of the requirements for the concerning class. Additionally, the users can add new concepts in this stage. When selecting a mechanism for another image of the same class, these newly added concepts will also be part of the input list to speed up annotating time and prevent the definition of ambiguous concepts. The list is ordered alphabetically to help the user pick the correct concept. A flow of all the possibilities with the user interfaces attached is shown in Figure 3.10b.




(a) The mechanism annotation interface for workflow 2




(b) The flow of annotating a custom mechanism for workflow 2

Figure 3.10: The user interface for workflow 2

### 3.5.4. Workflow 3: Adding new requirements

So far, we defined requirements, annotated them and subsequently annotated the mechanisms. Because the requirements originate from the input of the users, there could be images that have no verified requirements at all. The goal of this workflow is to help the user define the requirements for these images. In a Utopian world, we would be able to define a brief set of requirements for each class and all images of that class should feature at least one of those requirements. Unfortunately, the reality is a bit more complex. As an example, think of an arbitrary simple object. Different shapes, colours, compositions and camera angles exist which all influence which concepts are (not) visible. Additionally, the context in the image around an object might influence how a concept in an image should be classified. Additionally, it might also be the case that the user finds the image inappropriate and wants to exclude it from further analysis. Reasons for this can be that the object is very unclear and hence useless for the model, or that the image does not feature the supposed object of the class at all. For these cases, we let the user annotate a *remark*. The outline of workflow 3 is shown in algorithm 4 and the user interface is depicted in Figure 3.11. When a user adds a remark, the requirement is disabled because doing so marks the image as invalid for further analysis.

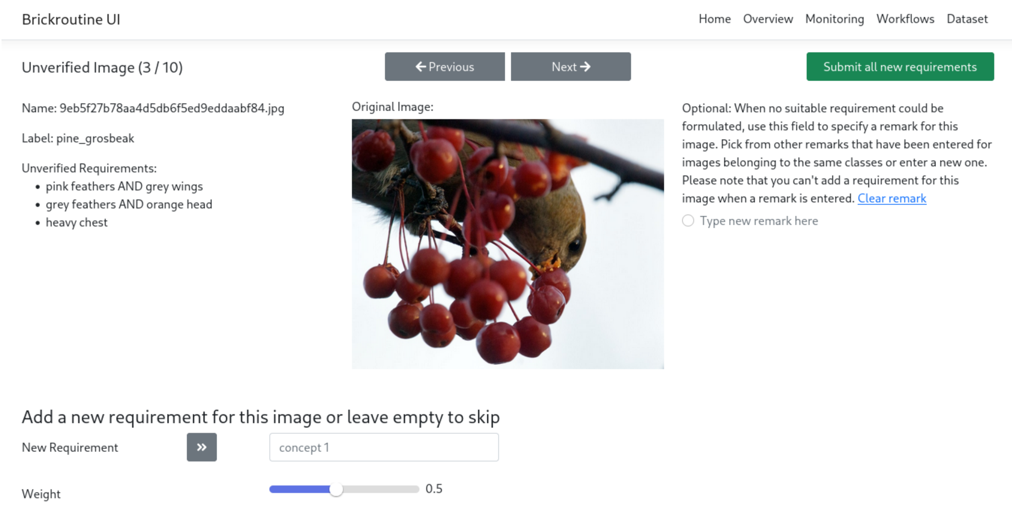



Figure 3.11: Adding new requirements for unverified images

**Condition:** Images without verified requirements $> 0$
**Data:** Images without verified requirements
**Result:** Images with an optional requirement or optional remark

**for** *All images $I_i$ without verified requirements* **do**
  **if** *User can to add a requirement for this image* **then**
    Ask for requirement
    Mark requirement as verified for this image
  **else if** *user wants to add a remark for this image* **then**
    Add remark for this image
**end**

**Algorithm 4:** Adding requirements for unverified images

### 3.5.5. Workflow 4: Requirements Correction

In the process of workflow 1 (subsection 3.5.2). Requirements with weights were elicited from the user. After going through the other workflows, the results in this state of the system could be informative with respect to the requirements. This workflow is concerned with the validity and if necessary, correction of the requirements. For each requirement that is now known by the system the state could be one of the following four:

- *No images have this requirement*
  During workflow 1, no images were annotated to feature the combination of concepts listed in this requirement. This can either mean that for all images a requirement with a higher weight than

this one was annotated or that none of the images matches the concepts of this requirement at all.

- *Not all mechanisms match this requirement*
  This indicates that the requirements were present in images but not for all those images the mechanism was present. This can be an indication that the requirement is not suitable. Therefore, we ask the user to review this requirement and possibly adjust it accordingly.

- *Mechanism found in images, but only for $x$ the mechanism was spanning the entire concept*
  The mechanism was found in the image, but for $x$ of the images, the mechanism was spanning the entire concept. This means that for some images, the mechanism was matching the requirement but only partially. This is an indication that the model might have learned a concept of finer granularity. Therefore, the user is encouraged to annotate concepts of finer granularity than the one(s) currently in the requirement.

- *Mechanism found in images*
  This means that for all the images in which this requirement was present, the mechanisms are matching. The granularity of the requirement is therefore likely to be appropriate. The user is encouraged to review the weight of this requirement. When a requirement has a relatively high number of verified mechanisms, the weight should be proportional so that this requirement is shown early in the process during subsequent runs of workflow 1.

**Condition:** Mechanism validation annotation done for a set of images for each class
**Data:** Requirements for each class
**Result:** Validated or improved requirements

1 **for** *All classes* **do**
2     **for** *All requirements in this class ordered by weight in ascending order* **do**
3         Present requirement to with editable concept classes and weight to user
4         Present state of this requirement to the user
5         **if** *User changes a requirement $R_i$ of class $c$* **then**
6             **for** *Images $I_i$ of class $c$ that previously had this as a verified requirement* **do**
7                 Set this image as unannotated
8             **end**
9         **end**
10     **end**
11 **end**

**Algorithm 5:** Pseudocode for Workflow 4

Logically, some requirements will change when the user uses this workflow. As a result, the old requirement is not existing anymore in the system and the state of the images needs to be modified appropriately. Given an example requirement $r$ with concepts $c_0$ ... $c_{n-1}$ that changes to $r'$ with concepts $c'_0$ ... $c'_{n-1}$, there is no way to know that the images that had $r$ as a verified requirement, also feature $r'$. As a result, we set the state of these images back to unannotated. This is covered in algorithm 5 at lines 5-9. When the user changes the requirements in workflow 0 (Figure 3.6), the same procedure will take effect.

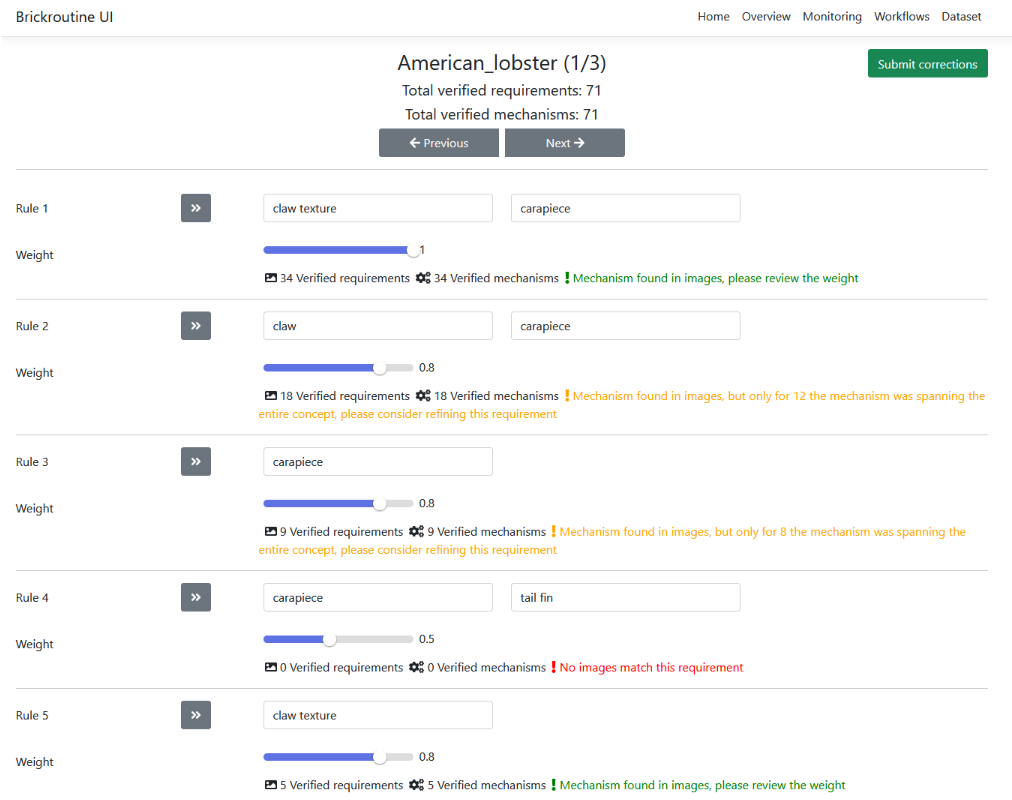



Figure 3.12: Workflow 4 in the UI

### 3.5.6. Workflow 5: The Overview

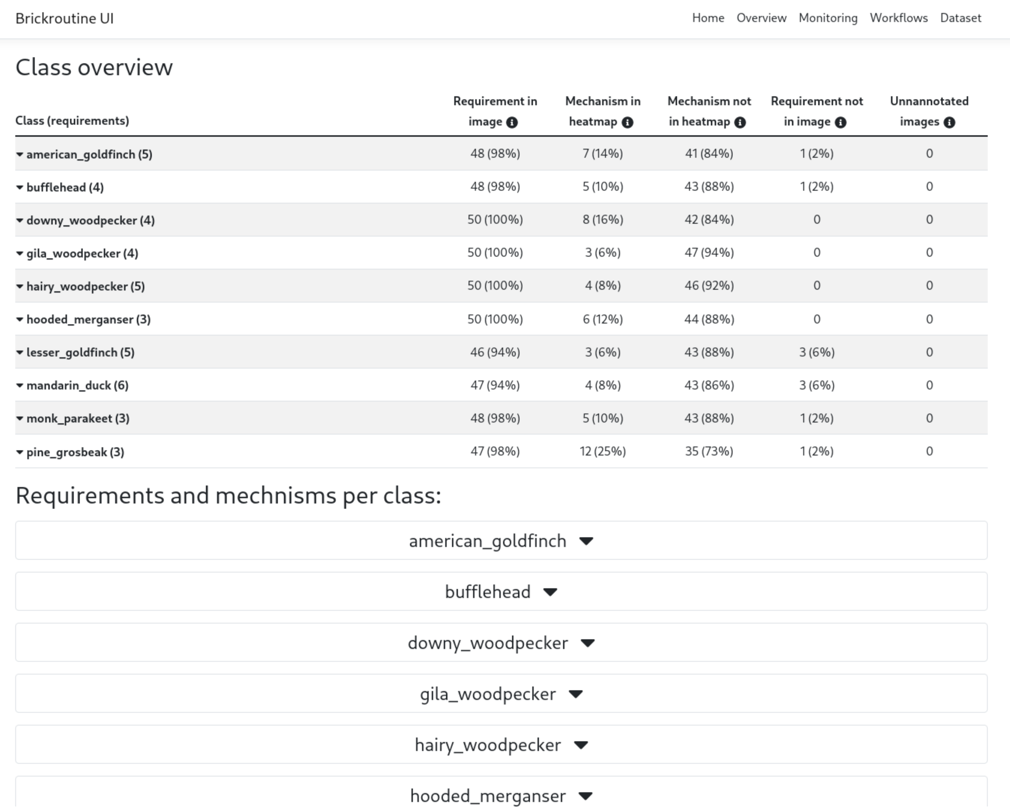
The workflows we defined so far, function as the foundation for our final and most revealing workflow where all dots are connected. In this workflow, it's time for the harvest. As opposed to other workflows, this workflow is about consuming explanatory data.

At the top of the screen, in Figure 3.13a we see our expendable table again with for each class (10 in Figure 3.13a) and an expandable section for each class below. In Figure 3.13b, some rows have been expanded. For each column, left to right, the meaning is as described below. These descriptions will also be shown when users hover over the question mark at each column header.

- *Requirement in image*
  All images for which the presence of this requirement was true. Includes both images that went through the heatmap annotation step and images that still have to.
- *Mechanism in heatmap*
  Images for which the requirement was covered by the heatmap.
- *Mechanism not in heatmap*
  Images for which the requirement was not covered by the heatmap and a custom mechanism was annotated.
- *Requirement not in image*
  All images for which the presence of this requirement was false.
- *Unannotated images*
  Annotate these images in the requirement annotation step (workflow 1).

Please note that the first column (Requirement in image) is always greater than or equal to the second column (Mechanism in heatmap) because we annotate the requirements before the mechanisms. Logically, the third column is the difference between the previous two.

Figure 3.13b shows the expanded table, in which we see per class the requirements that are present in our images (first column) and if the mechanism was following this mechanism (second and third column). The shown percentages behind each number is with respect to the total amount of images known by the system for this class. By using this table, the user can quickly check the results with respect to the defined requirements on a high level.



Brickroutine UI | Home Overview Monitoring Workflows Dataset

Class overview

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| american_goldfinch (5) | 48 (98%) | 7 (14%) | 41 (84%) | 1 (2%) | 0 |
| bufflehead (4) | 48 (98%) | 5 (10%) | 43 (88%) | 1 (2%) | 0 |
| downy_woodpecker (4) | 50 (100%) | 8 (16%) | 42 (84%) | 0 | 0 |
| gila_woodpecker (4) | 50 (100%) | 3 (6%) | 47 (94%) | 0 | 0 |
| hairy_woodpecker (5) | 50 (100%) | 4 (8%) | 46 (92%) | 0 | 0 |
| hooded_merganser (3) | 50 (100%) | 6 (12%) | 44 (88%) | 0 | 0 |
| lesser_goldfinch (5) | 46 (94%) | 3 (6%) | 43 (88%) | 3 (6%) | 0 |
| mandarin_duck (6) | 47 (94%) | 4 (8%) | 43 (86%) | 3 (6%) | 0 |
| monk_parakeet (3) | 48 (98%) | 5 (10%) | 43 (88%) | 1 (2%) | 0 |
| pine_grosbeak (3) | 47 (98%) | 12 (25%) | 35 (73%) | 1 (2%) | 0 |

Requirements and mechnisms per class:

american_goldfinch

bufflehead

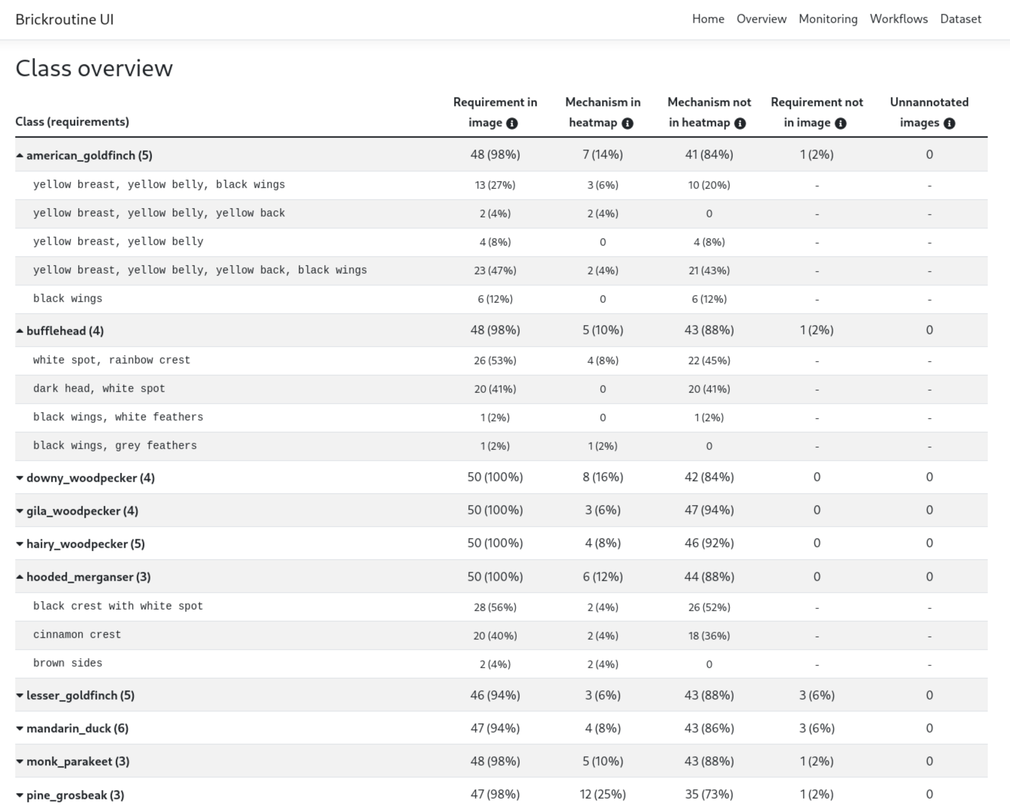
downy_woodpecker

gila_woodpecker

hairy_woodpecker

hooded_merganser

(a) The overview pane: General result



Brickroutine UI | Home Overview Monitoring Workflows Dataset

Class overview

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| american_goldfinch (5) | 48 (98%) | 7 (14%) | 41 (84%) | 1 (2%) | 0 |
| yellow breast, yellow belly, black wings | 13 (27%) | 3 (6%) | 10 (20%) | - | - |
| yellow breast, yellow belly, yellow back | 2 (4%) | 2 (4%) | 0 | - | - |
| yellow breast, yellow belly | 4 (8%) | 0 | 4 (8%) | - | - |
| yellow breast, yellow belly, yellow back, black wings | 23 (47%) | 2 (4%) | 21 (43%) | - | - |
| black wings | 6 (12%) | 0 | 6 (12%) | - | - |
| bufflehead (4) | 48 (98%) | 5 (10%) | 43 (88%) | 1 (2%) | 0 |
| white spot, rainbow crest | 26 (53%) | 4 (8%) | 22 (45%) | - | - |
| dark head, white spot | 20 (41%) | 0 | 20 (41%) | - | - |
| black wings, white feathers | 1 (2%) | 0 | 1 (2%) | - | - |
| black wings, grey feathers | 1 (2%) | 1 (2%) | 0 | - | - |
| downy_woodpecker (4) | 50 (100%) | 8 (16%) | 42 (84%) | 0 | 0 |
| gila_woodpecker (4) | 50 (100%) | 3 (6%) | 47 (94%) | 0 | 0 |
| hairy_woodpecker (5) | 50 (100%) | 4 (8%) | 46 (92%) | 0 | 0 |
| hooded_merganser (3) | 50 (100%) | 6 (12%) | 44 (88%) | 0 | 0 |
| black crest with white spot | 28 (56%) | 2 (4%) | 26 (52%) | - | - |
| cinnamon crest | 20 (40%) | 2 (4%) | 18 (36%) | - | - |
| brown sides | 2 (4%) | 2 (4%) | 0 | - | - |
| lesser_goldfinch (5) | 46 (94%) | 3 (6%) | 43 (88%) | 3 (6%) | 0 |
| mandarin_duck (6) | 47 (94%) | 4 (8%) | 43 (86%) | 3 (6%) | 0 |
| monk_parakeet (3) | 48 (98%) | 5 (10%) | 43 (88%) | 1 (2%) | 0 |
| pine_grosbeak (3) | 47 (98%) | 12 (25%) | 35 (73%) | 1 (2%) | 0 |

(b) The overview pane: The requirements information table

Figure 3.13: The overview pane 1/2

Revisiting two of our earlier-defined user stories below, we will transform these requirements into a user interface:

- As a user I want to interpret my computer vision models so that I can know on which semantic concepts of my images the model reasoned for correctly classified images and I can verify if this is valid.

- As a user I want to interpret my computer vision models so that I can know on which parts of my images the model reasoned for incorrectly classified images and I can take appropriate actions.

This suggests that we should first segment on classes on subsequently on the predicted label. As a result for each class, the interface in Figure 3.14a shows a set of tables. Every row in each table is clickable and will open a popup with the associated images. The different table types are defined below and will only be shown if it has at least one row.

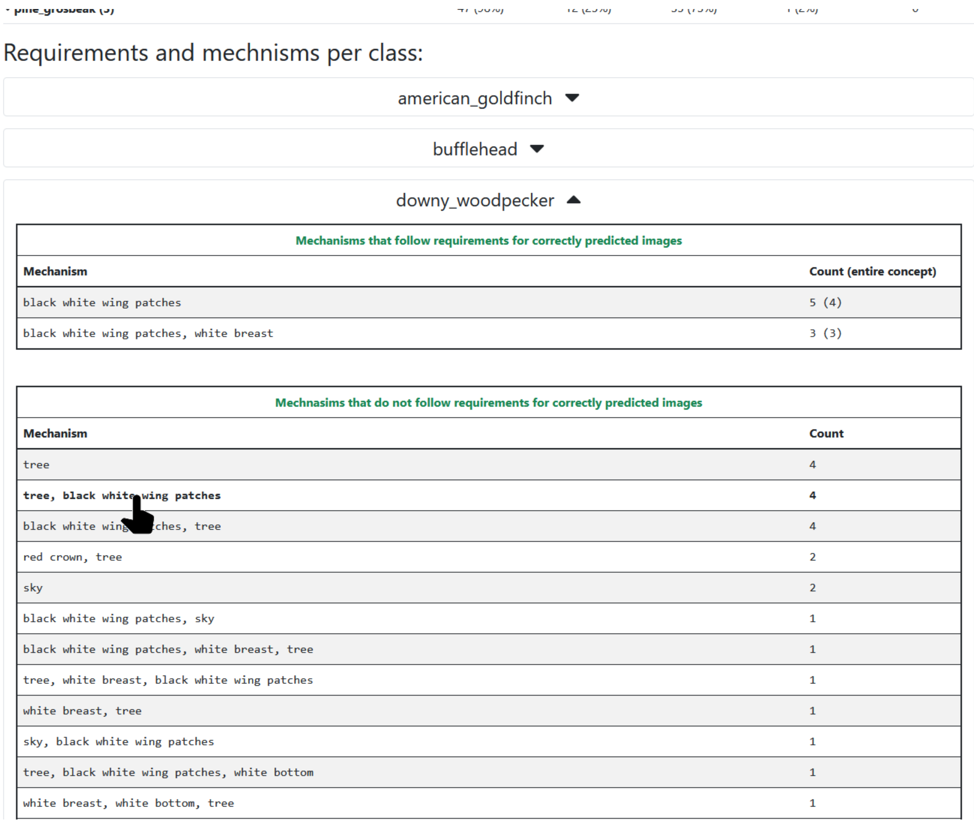
- *Mechanisms that follow requirements for correctly predicted images*
  These are the requirements that also are shown in Figure 3.13b but filtered on correct predictions. By clicking on a row, the user can inspect the heatmaps for correctly predicted images and see if the model followed a valid mechanism and thus verify if the right concepts are used.

- *Mechanisms that do not follow requirements for correctly predicted images*
  These are the custom mechanisms that the user annotated for correctly predicted images. By clicking on a row, the user can inspect the heatmaps for correctly predicted images that did not follow the requirements. In case the model learned a background concept or has another bias, the user can observe the mechanisms here and use this acquired knowledge, either within our outside Brickroutine.

- *Mechanisms that follow requirements for incorrectly predicted images*
  These are the requirements that also are shown in Figure 3.13b but filtered on incorrect predictions. By clicking on a row, the user can inspect the heatmaps for correctly predicted images and inspect which mechanisms are followed to incorrectly predict class. Each row is a unique combination of mechanism, class label and predicted label so that the user can understand incorrect predictions.

- *Mechanisms that do not follow requirements for incorrectly predicted images*
  These are the custom mechanisms that the user annotated for incorrectly predicted images. By clicking on a row, the user can inspect the heatmaps for incorrectly predicted images that did not follow the requirements and track possible causes for this.

- *Remarks for unverified images*
  When a user indicated that a specific image was not useful and thus a remark was entered in workflow 3 (subsection 3.5.4), the heatmaps of remarked image(s) will be shown when the users click on these rows. This will help the user understand the role of these images with respect to the model's behaviour.



Requirements and mechnisms per class:

american_goldfinch

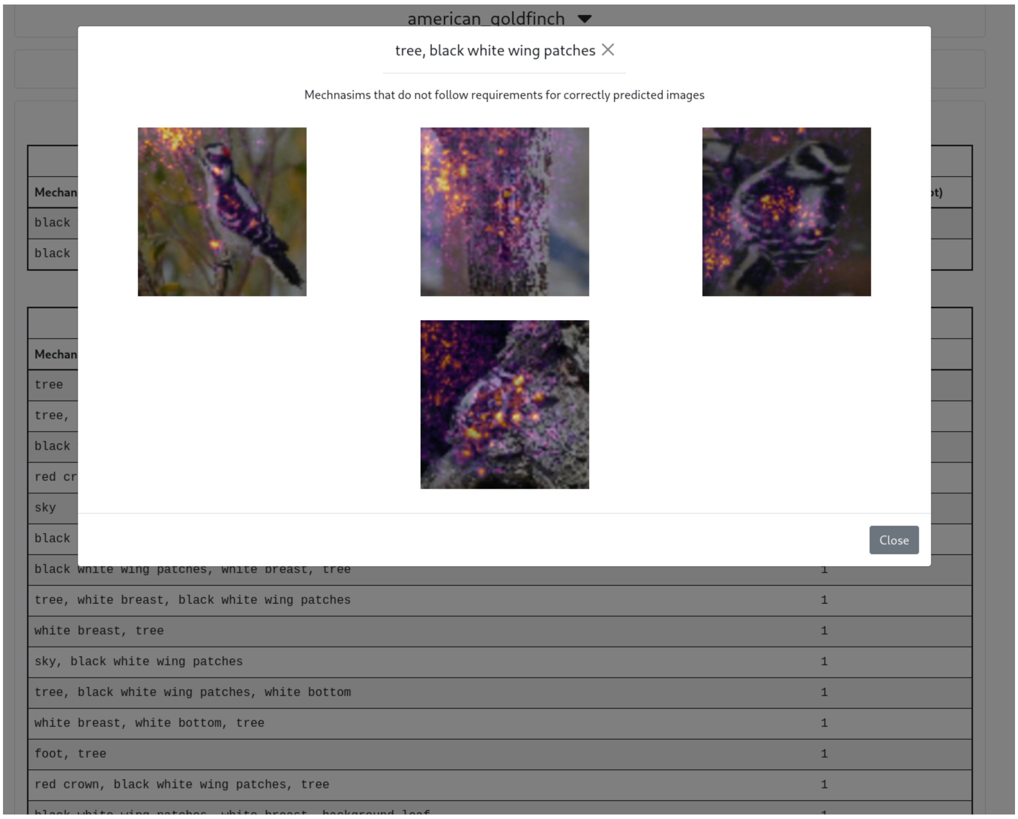
bufflehead

downy_woodpecker

| Mechanisms that follow requirements for correctly predicted images | |
|---|---|
| **Mechanism** | **Count (entire concept)** |
| black white wing patches | 5 (4) |
| black white wing patches, white breast | 3 (3) |

| Mechnasims that do not follow requirements for correctly predicted images | |
|---|---|
| **Mechanism** | **Count** |
| tree | 4 |
| **tree, black white wing patches** | **4** |
| black white wing patches, tree | 4 |
| red crown, tree | 2 |
| sky | 2 |
| black white wing patches, sky | 1 |
| black white wing patches, white breast, tree | 1 |
| tree, white breast, black white wing patches | 1 |
| white breast, tree | 1 |
| sky, black white wing patches | 1 |
| tree, black white wing patches, white bottom | 1 |
| white breast, white bottom, tree | 1 |

(a) Mechanisms explained




(b) Clicked on a mechanism row

Figure 3.14: The overview pane 2/2

## 3.6. Process Overview

After having defined the five workflows of Brickroutine, all the components are in place. It is important to notice in this stage that we cannot truly know what the model has learned because the semantic concepts that are given by the user of the system are a conceptualization of numbers (black box, see section 2.3). As a result, the requirements and corresponding weights that are initially submitted by the user can change over time to represent the images in the data set and the resulting heatmaps of the model better. Moreover, the requirements can change because the level of granularity of the concepts in the requirements is adjusting (recall the definition of seed concepts from section 3.1). Therefore, we propose the sequence of actions that are depicted in the flowchart of Figure 3.15. The colours of

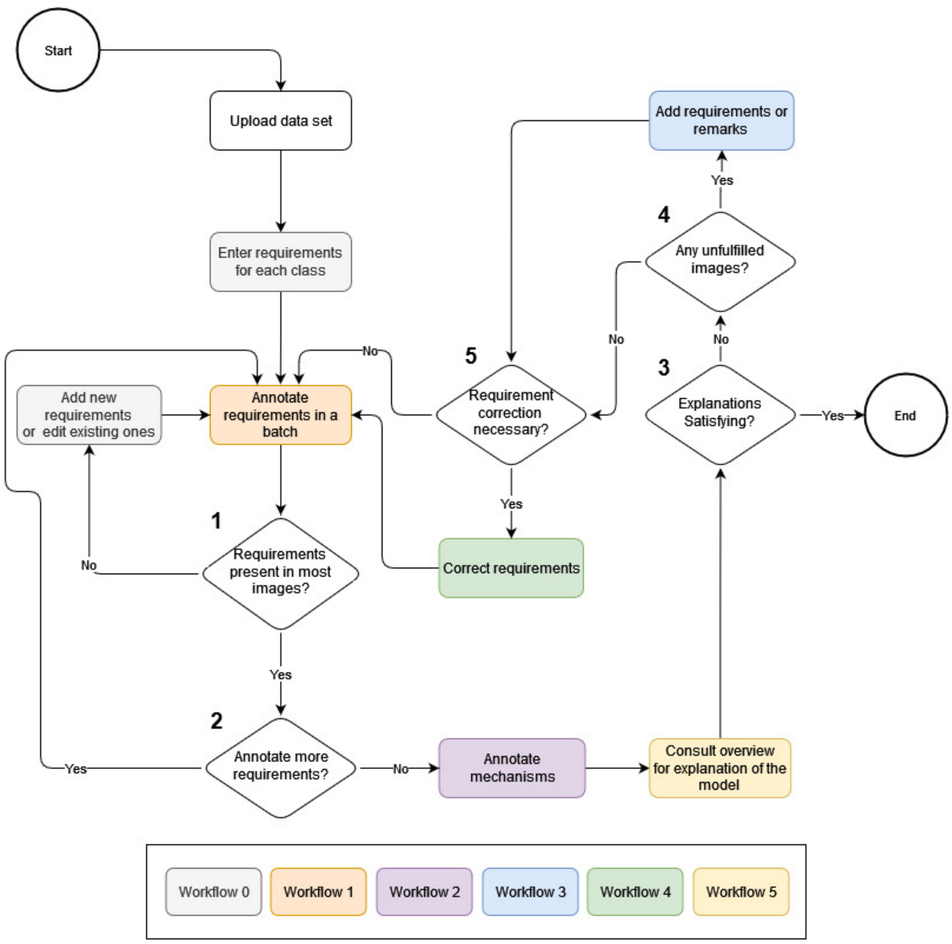



Figure 3.15: A flowchart on how to use Brickroutine

the boxes in the flowcharts indicate to which workflow this specific step corresponds. Some subjective notions on the decision boxes:

1. *Requirements present in most images?*
   At both the home pane and the overview pane, an overview table is shown that indicates for how many images each requirement is present. The user should consult this and act accordingly. If for a large number of images the requirements are not present, the user should alter the requirements or add new ones.

2. *Annotate more requirements?*
   After annotating the requirements, the user should have developed an intuition of the representation of the requirements in the images. If the user is confident that the requirements in this stage are properly defined, more requirements can be annotated right away. Else, the user should assess the models' mechanism first in workflow two, which either verifies or fine-tines the requirements.

3. *Explanations satisfying?*
   Depending on the goal with respect to the upload data set and results, it is up to the user to assess if those goals are met. This is where the overview pane (Workflow 5, subsection 3.5.6) is designed for. If the goal is to look for explanations of misclassified items ("why has my model prediction class $x$ instead of true label $y$"), then the user should search for that combination.

4. *Any unfulfilled images?*
   If there are images that do not have verified requirements, then encouraging the user to add more requirements will improve the coverage of the requirements and therefore, the change that matching mechanisms will be found in subsequent iterations of annotating.

5. *Requirement correction necessary?*
   After every iteration of annotating, the user should have developed an intuition about the relevance and granularity of the defined concepts. By making use of workflow 4 (subsection 3.5.5), the user can see the absolute numbers on (partial or entire) coverage of the requirements and manually assess if the weight for each requirement is appropriate.

We believe that following the sequence defined in this flowchart gives the user flexibility to decide if there are additional iterations required and will result in an efficient cost/informativeness ratio regarding the goal of a specific user. This flowchart is also shown in the system on the home page and the workflow selection page (Figure 3.16).

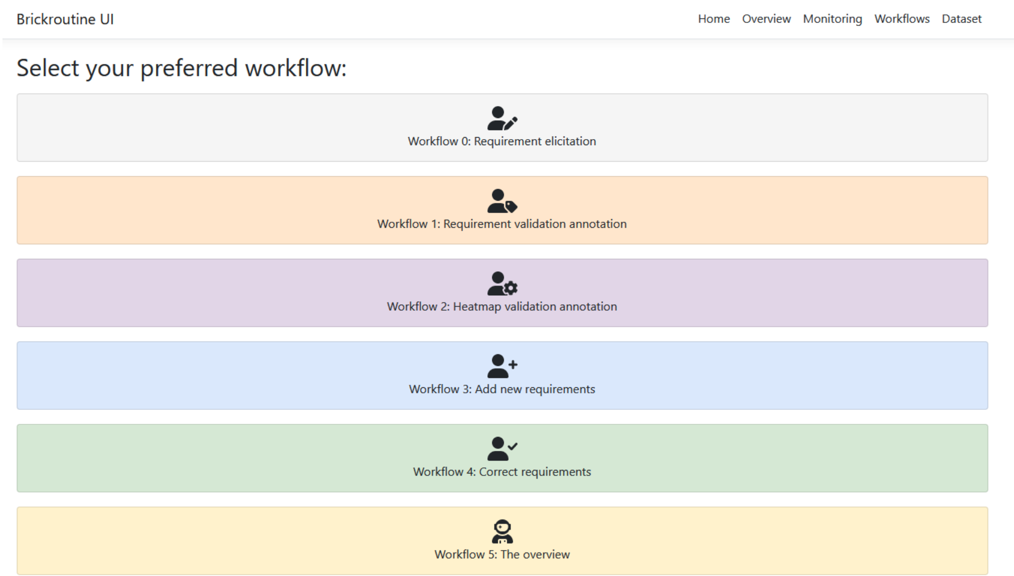



Figure 3.16: The interface for selecting the workflows

Lastly, we present the interface depicted in Figure 3.17, that allows the user to visit external pages of storage and communication mechanisms that are being used.

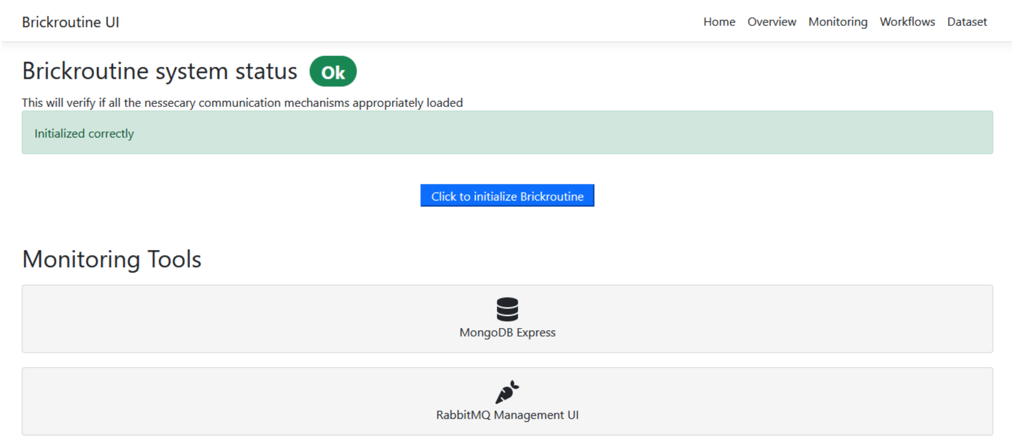



Figure 3.17: The interface for selecting the monitoring tools

# 4

# System Design

After our functional design and requirements, this chapter describes the architectural design and implementation of Brickroutine. First, we will describe architectural requirements that stem from the context and functional requirements of our system. Subsequently, we will describe the seven different building blocks that currently are the embodiment of Brickroutine.

## 4.1. Architectural Requirements

Thus far, we have addressed different technical domains. We want our system to be at the intersection of data science, software architecture and interface design. The requirements for Brickroutine that we take into account with respect to the architecture include:

- **Extensibility**
  Since our system is the first attempt into making a system from scratch, it is likely that changes will follow in the future. As a result, we need to pick our architecture in such a way that it easily allows for extensions without having to comprehend and alter the entire code base.
- **Modularity**
  Modularity is a requirement for achieving extensibility. We want to split up our system into modules because: (1) Brickroutine becomes easier to maintain when we target one specific module at a time while others are operational and (2) it allows for easier testing and debugging. Having small modules in a software system is referred to as the Polylithic Principle in architecture [15].
- **Compatibility**
  In chapter 3, we have already seen that our system consists of a variety of components. Data science applications like extracting the heatmaps in section 3.4 are mostly in Python nowadays. Since we implement our user interface as a web page, we need javascript compatibility. For uploading and storing the data sets and annotations, we need a backend component. Additionally, during the implementation phase and possible follow-up phases after this thesis project has ended, it is highly likely that execution and development will be on local machines. We need to take into account the variety of operating systems and the effort to install the necessary runtime. This suggests that the architecture should be compatible with a variety of languages and frameworks and that it should ideally be compatible with different operating systems.

## 4.2. Backbone: Docker

Nowadays, containerization is a hot topic in the realm of software engineering and, in our application, is a tool to achieve extensibility, modularity and compatibility. A container is an isolated run time environment that runs on a host computer and supports almost all modern software implementations. Instead of running in a dedicated virtual machine, containers sit on top of a physical system and its operating system. Each container shares the host operating system kernel, binaries and libraries are created from scratch when the container is started and no leftover files remain on the host system when this containerized environment is deleted [16]. By using this containerized approach, we are not limited to operating systems or programming languages and hence achieve compatibility.

Using docker containers as building blocks for our system, allows us to isolate different components in our system from each other and satisfy the modular requirement. These components will be discussed in subsequent sections. Every component is unaware of each other's implementation, only the abstractions (data models, API endpoints) have to be known in order for the different components (in this design: containers) to communicate. In this way, we leverage the microservice approach from section 2.5. An overview of Brickroutine's components is given in Figure 4.1. Each component that is implemented in a container has a subtitle indicating it. Two of our chosen components are Software as a Service (SaaS) components that run in the cloud. In the left box of each component, the section elaborating on that specific component is given.

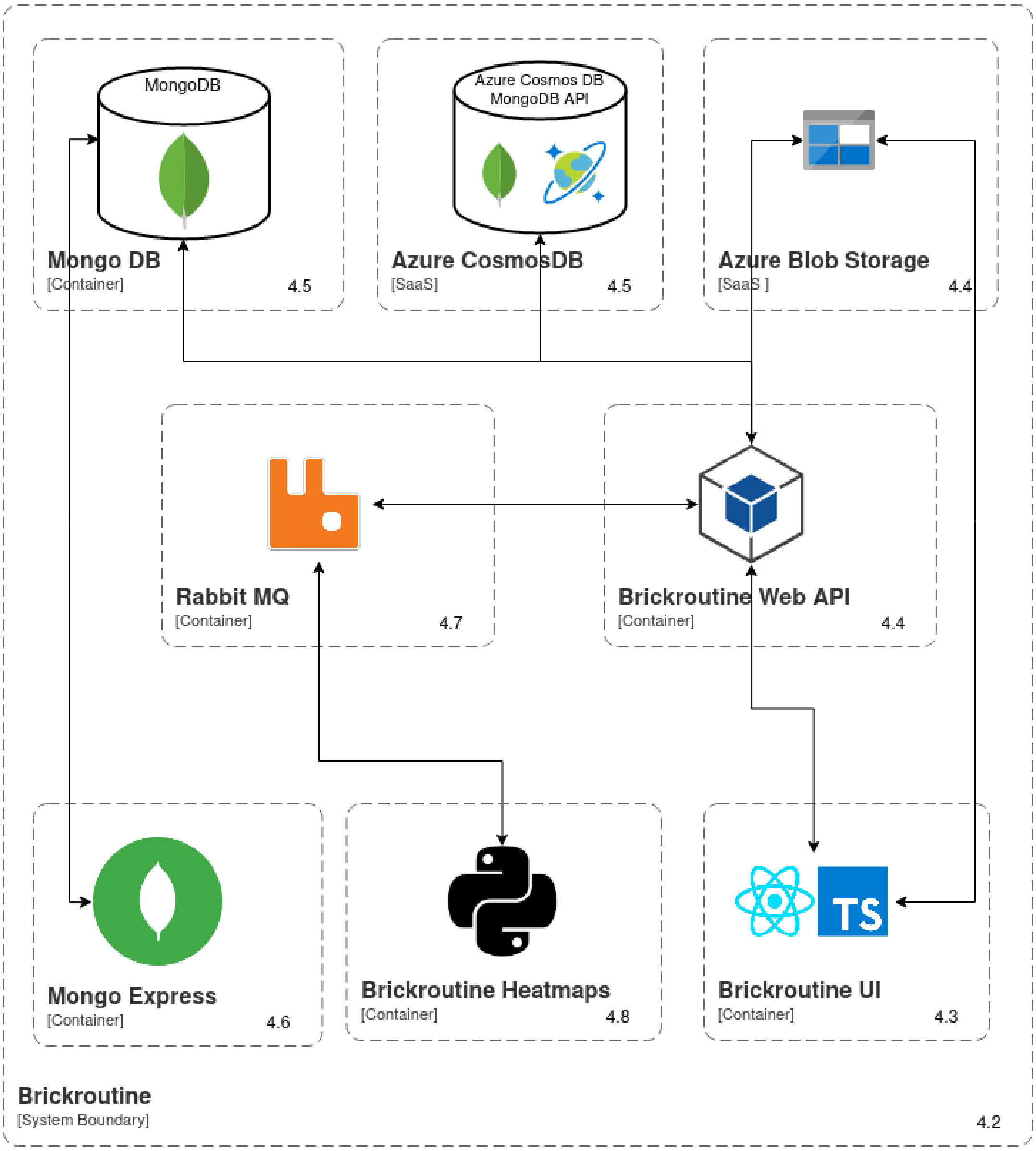



Figure 4.1: Schematic overview of the system architecture

With this multi-container approach for Brickroutine, we need to ensure that they operate as a system and that we can start, stop and modify the containers. We use Docker Compose for this[1]. Docker Compose is a tool for running applications that consists of multiple containers by means of a YAML[2] file [16]. In this file, we can aggregate the contents of multiple Dockerfiles, which can be seen as a

[1] https://docs.docker.com/compose/
[2] https://yaml.org/

standalone (micro)service. This approach allows for multiple configurations. For instance, we have a separate compose file for development configurations that a debugger can be attached to and listens to code changes. Instructions and commands for using the application are listed in the `readme.md` of this project's repository and is included as Appendix D. When using Docker Compose, an internal network is created that allows containers to communicate with each other. For each container, TCP ports have to be explicitly exposed, contributing to the security of the system.

## 4.3. User Interface: React

In chapter 3, we have shown the user interfaces for Brickroutine. Since our containerized approach allows us to pick any technology, we choose a web-based user interface that users can simply run in a web browser. For our implementation, we choose React to build our interfaces. As part of the Progressive Web App (PWA) paradigm, react is used to create applications that load a single web page and update the content based on the users' interaction. Besides the smooth user experience, this results in an expandable interface when components are added during follow-up work after this project, satisfying the extensibility requirement. For using the react framework, we choose to use Typescript as the programming language. Typescript is a syntactic flavour of javascript and adds static typing, which we regard as beneficial for a developer's experience and maintainability.

React leverages a component-based approach. User elements can be defined as components that have a specific set of input data and change based on the data that is passed through the components. Components can consist of other components to make the approach flexible. In chapter 3, we saw different interfaces for different purposes that are visually similar to others. Defining these as components ensures that we do not have to write repeated code. In the end, our React application is built as a plain web application and can be run in an NGNIX [3] container that only is `27MB` in size.

## 4.4. API: .NET

Every user action from the front-end that requires interaction with any other component starts with a REST call to our so Web API. Running in a separate Docker container, this back-end of our application is concerned with, but not limited to the following responsibilities:

- Getting information from the database and returning it to the user.
- Getting information from the user (e.g. annotations or an uploaded data set) and storing it in the database.
- Receiving an action from the user and passing it to the right component (e.g. extraction of the heatmaps)

For our implementation, we picked the *.NET* framework. This is a software development framework from Microsoft that, amongst other purposes, allows for rapid web application development using the Object-Oriented C# programming languages. Our reasons to pick this for Brickroutine include:

- Native support for implementing APIs that are used by the user interface
- Our system does various things like simultaneously, processing data, uploading images, and generating requests for heatmaps. We don't want our user system to be blocked when tasks that have no outcome in the user interface are executed and thus want to implement asynchronous processes. The .NET framework has good support for this with the Task-based asynchronous pattern (TAP) [4].
- Support for multiple platforms and thus suitable for developing and running it in a Linux Docker container.

This setup includes a dependency injection framework out of the box so that we can adhere to the single responsibility and dependency inversion principle [21], which states that classes should rely on concretions instead of abstractions. By using this principle, projects are only aware of the public functions (interfaces) and parameters of others, yet without knowing anything about the implementation itself. Below, we list different components of our API implementations, which all are implemented as separate .NET projects:

[3] https://www.nginx.com/
[4] https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap

- **Brickroutine.WebAPI**
  This project is the first layer that receives the API requests from the Front-End. With Headers in the HTTP calls, we differentiate between different data sets and the used models are compatible with the front-end.
- **Brickroutine.DatabaseService**
  With API libraries from MongoDB, this project is used to write queries in a C# that are compiled into MongoDB statements that are subsequently sent to the database to fetch, write and update our data.
- **Brickroutine.Common**
  Contains definitions for interfaces, and the implementation of the abstractions of other components (recall dependency inversion). Additionally, all constants are defined here to prevent the hard coding of variables and magic numbers.
- **Brickroutine.BlobstoreService**
  We choose to upload our input images and heatmaps to the azure blob storage [5]. This is online storage in which all images can be easily retrieved with an URL. We picked this for the easy integration with our .NET backend. This project handles the uploading of the images.
- **Brickroutine.RabbitMQManager**
  A separate project that is used to send requests to our message broker. All actions that originate from the front end have to go through our API.
- **Brickroutine.ConsumeRabbitMQHostedService**
  Listens to messages that are directed to the API. Makes use of an observable and checks for the appropriate message headers. Currently used to obtain the extracted heatmaps and subsequently upload them to the Azure blob store.

## 4.5. Storage: MongoDB

To persist data that is produced during the usage of Brickroutine, a storage mechanism is required. Because we want our system to be extensible and are unaware of which (un)structured data we might have to store in the future, we choose the NoSQL paradigm over the traditional RDBMS As part of the NoSQL paradigm, we picked MongoDB for Brickroutine, a document-based key-value database [31]. MongoDB stores data in its own format called BSON. BSON stands for binary JSON and supports all ubiquitous data types [8]. For our implementation, the reasons to pick this technology include:

- Integration with the backend through C# API for MongoDB.
- When our system becomes larger and possibly has large chunks of unstructured data, a NoSQL DB like MongoDB system gives flexibility.
- Due to the support for unstructured data that NoSQL systems offer, nullable properties are not saved to the database when they do not contain a value. This results in less storage overhead.
- There are docker images available for MongoDB. Hence, for development or running the system locally, a database can simply be spun up in a docker container.
- MongoDB is a ubiquitous implementation and therefore finds sufficient support on modern cloud platforms when we choose to store the data online. In our implementation, to enable sharing data while still running the application itself locally, we have support to connect to Azure Cosmos DB with a MongoDB implementation. The user should configure the application in such a way that one of those is connected to the backend.

Because of the MongoDB APIs used in our back-end, we can define C# models that are converted into suitable DB models. A UML Diagram of our data models is shown in Figure 4.2. The `DataSet` class is the root and has a $1 : N$ relationship with classes `ConceptsClass` and `Image`, that get added when the user uploads a data set. Although in our workflows we currently support the verification of one requirement per image, we have designed our data models to support multiple for follow-up work.

[5] https://azure.microsoft.com/en-us/services/storage/blobs/

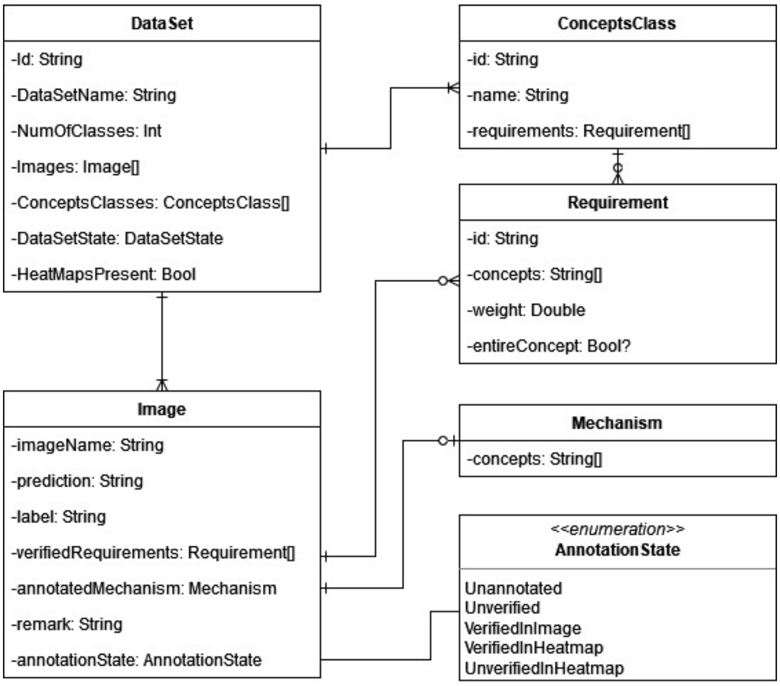



Figure 4.2: Class models used in the Brickroutine database

## 4.6. Data Monitoring: Mongo Express

Exploring and maintaining the data is cumbersome when the database has only command-line interface (cli) support and runs in a containerized environment. To overcome this barrier, we incorporated Mongo Express into the system. Mongo Express is a web-based interface for administrative purposes. The main idea behind this is that users can quickly inspect, modify, delete, and restore their data without the burden of going through complex sets of actions. Like the database itself, this runs in a separate docker container with the appropriate TCP ports exposed so that users can navigate to this web-based interface from the Brickroutine UI as depicted in Figure 3.17. A screen capture of Mongo Express with our evaluation data is shown in Figure 4.3. It gives insights in the data that currently is in the system and allows the Brickroutine developers to modify it accordingly.

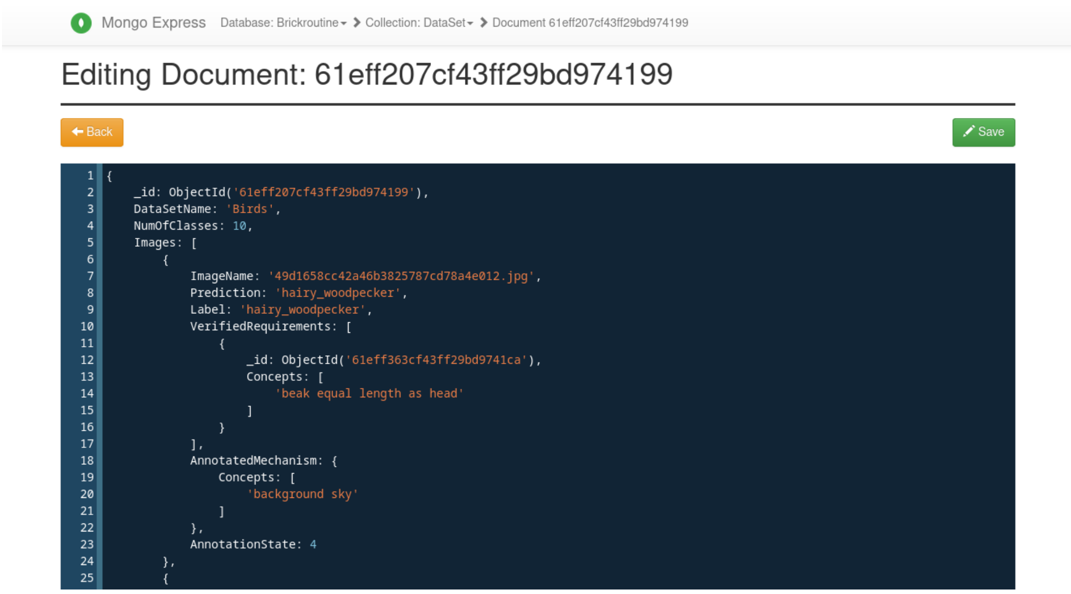



Figure 4.3: Mongo Express with data about an annotated image

## 4.7. Communication: RabbitMQ

Now that we have split up our system into different components with the possibility to add more in the future, a challenge is presented from the way these individual components communicate with each other. In our implementation, we follow an Event-driven Architecture (EDA) where communication occurs through events. An EDA is a common approach to implementing microservices and uses messages to facilitate the integration of separate software components. An EDA usually consists of four parts[15]. Below is a brief description of each of them and the current parts in our system that fulfil that role.

- *Event publisher:* When an event happens, a message is published to a messaging platform. In our system we currently have two publishers: 1) The web API that sends requests for extracting the heatmaps with raw image data and 2) the heat map extractor that sends the completed heatmaps back to the API to process them and upload them to the storage.
- *Event subscriber:* Subscribers are endpoints that listen to specific types of events. Currently, we have two subscribers that are the opposite of the publishers mentioned above.
- *Event broker or routers:* We use RabbitMQ [6] as the message broker. This is an open-source message broker that can be run in a docker container. RabbitMQ makes use of multiple message queues in which publishing software components can place messages that are retrieved from the queues by subscribing components.
- *Event persistence:* RabbitMQ facilitates that, as long as the RabbitMQ service is up, messages are kept in the queue. In our current implementation, we configure that the publishing component should receive acknowledgements, which results in the deletion of the message when it is properly received.

To facilitate complex routing structures, RabbitMQ does not allow producers to publish messages to a queue directly. Instead, it implements a so-called *exchange*, that the messages should be published to. This exchange subsequently forwards messages to the right queue by means of a routing key. We use one single exchange for Brickroutine and have two queues, as presented in Table 4.1.

| Queue name | Routing key | Description |
|---|---|---|
| brickroutine.heatmpas | heatmaps | Queue that is used to receive the input images and requests to extract the heatmaps |
| brickroutine.api | api | Generic queue listener that can processes events |

Table 4.1: Queues and routing keys used in Brickroutine

Although currently, only the process of generating the heatmaps uses this event-driven approach, the system can easily be expanded because RabbitMQ has implementations for 11 commonly used programming languages. To expand this system, we simply have to run the software we want to add in a docker container that is attached to our network and it can communicate with other parts through this message broker.

## 4.8. Heatmap Extraction: Python

The ideas and goals behind the heatmap extraction have been discussed in section 3.4. Technically, this is implemented with Keras[7] and Tensorflow[8] functions. The python program in our heatmaps extraction container continuously listens for requests that originate from the API. If a request is received, it will start the process of extracting them in a separate thread. When completed, it sends the contents of the heatmaps back over a RabbitMQ connection where our API uploads this to Azure Blobstorage (section 4.4). We use routing keys and custom headers to filter for the correct tasks. Because we can only send binary data with RabbitMQ, we serialize the image data to base64 before sending them to other components. After serialization we have a 3D array with floating-point values ranging from 0 to 255, representing the images.

[6]https://www.rabbitmq.com/
[7]https://keras.io/
[8]https://www.tensorflow.org/

## 4.9. Comparison with Existing Solutions

In section 2.1, we mentioned existing solutions that provide model developers with tools for interpretability of their models. Brickroutine distinguishes itself in a number of ways. First, it should be used after training the model. Additionally, it is a user-friendly tool complete with a user interface and tools for monitoring and the system design is done in such a way that it is extensible.

# 5
# Evaluation: Informativeness

This chapter will describe the design, results and discussion of the evaluation with respect to the *informativeness* of Brickroutine.

## 5.1. Goals

We need to evaluate to which extent Brickroutine solved the initial problem of this thesis that is described in section 1.1:

> *"Humans have limited possibilities to disassemble a machine learning model and verify if the algorithm followed a line of reasoning that is comprehensible so that decisions can be assessed in terms of fairness, robustness, and trustworthiness"*

Derived from that problem statement, we assess the informativeness that Brickroutine gives us in order to solve the above-defined problem to a certain extent.

**Definition.** ***Informativeness*** *is the degree to which the system expresses information that can be used to assess the inner workings of an adequately performing model so that we can comprehend its reasoning.*

Eventually, we want to have entries in the overview panel that show that the model reasoned on overlapping concepts in the images. We want the explanations to be meaningful (e.g. in most cases, the model predicted class $x$ based on concepts $a_0 \ldots a_{n-1}$). This shows the users that the model consistently reasoned based on parts in an image and that the decision is not arbitrary. To be informative, our system should explain trends of the model's decisions in terms of semantics. Additionally, these semantics can be combined with the prediction outcomes and give insight into the models' behaviour. We can compare these findings with the costs for these analyses. The exact definition of these costs is given below.

**Definition.** ***Costs****: We measure the costs in time that a human spends on executing actions in Brickroutine. For every specific step that involves any human action, we keep track of the time it takes to complete that action. Additionally, the number of images at each step (if it involves annotating images) will be kept track of so that we can plot it against the amount of annotated images and possibly present some trade-offs.*

## 5.2. Experimental Setup

We use the data set *sea creatures*, consisting of 300 images of 3 classes (*lobster, shark, tench*, 100 images each). This is a subset of imagenet [1] and is selected by the authors of [2]. This used model is trained on Inception v3 and has weights pre-trained weights from imagenet. The predictions in this three-class data set are all correct. We choose to use this because the goal of this set of images, heatmaps and predictions is to give insights in how the model made predictions. The initial requirements that we used in workflow 0 for these classes are listed in Table 5.1.

[1] https://www.image-net.org/

| Class | Requirements |
|---|---|
| **American_lobster (8)** | claw, red shell, tail fin |
| | red shell, tail fin |
| | claw, red shell |
| | claw, tail fin |
| | red shell |
| | thin legs |
| | tail fin |
| | claw |
| **great_white_shark (6)** | mouth, snout, gill openings, dorsal fin, pectoral fin, caudal fin |
| | dorsal fin, pectoral fin, caudal fin |
| | mouth, snout, gill openings |
| | mouth, snout |
| | gill openings |
| | dorsal fin, pectoral fin |
| **tench (8)** | olive skin, dark rounded fins, red eye |
| | olive skin, dark rounded fins |
| | dark rounded fins, red eye |
| | olive skin, red eye |
| | red eye |
| | dark rounded fins |
| | olive skin |
| | lateral line |

Table 5.1: Initial requirements

### 5.2.1. Approach

We follow the iterative procedure from section 3.6 and gather qualitative and quantitative metrics after each iteration. We will do annotations until all annotated images are covered, to investigate how the metrics change over time and see the added value of each iteration. The parts of Brickroutine that are covered are within the border in Figure 5.1. We choose to leave Workflow 3 out of the evaluation because our data sets are of significant size and this will probably result in overhead costs. Even though the additional results might be converging, we continue to do it for all the possible images so that we can compare the gained knowledge against the extra costs. The iteration size in images will be set to roughly 20% of the images per class, resulting in 20 images for *Sea Creatures*

### 5.2.2. Metrics

The novelty of our work has the consequence that it cannot be compared against specific baselines or existing alternatives. To assess the informativeness of our system, we would like to know how much detail with respect to that goal is given. Additionally, since our approach has a crowdsourcing element, we also want to keep track of metrics with respect to the annotations. The resulting metrics are introduced below.

In the context of informativeness, we would like to observe the inner working of the model for the right predictions, i.e. does the model base correct predictions on the right semantic concepts? To evaluate this, we keep track of the relative **amount of verified mechanisms** over time. logically, we also keep track of **time**, because it serves as an important baseline for the applicability. Moreover, the **efficiency** of a system is important since, we want to minimize the number of annotations while maintaining a satisfactory outcome. Efficiency is defined as *the average number of annotations per image*. Because during the search for suitable requirements, image annotations get undone when a requirement changes (subsection 3.5.5, algorithm 5).

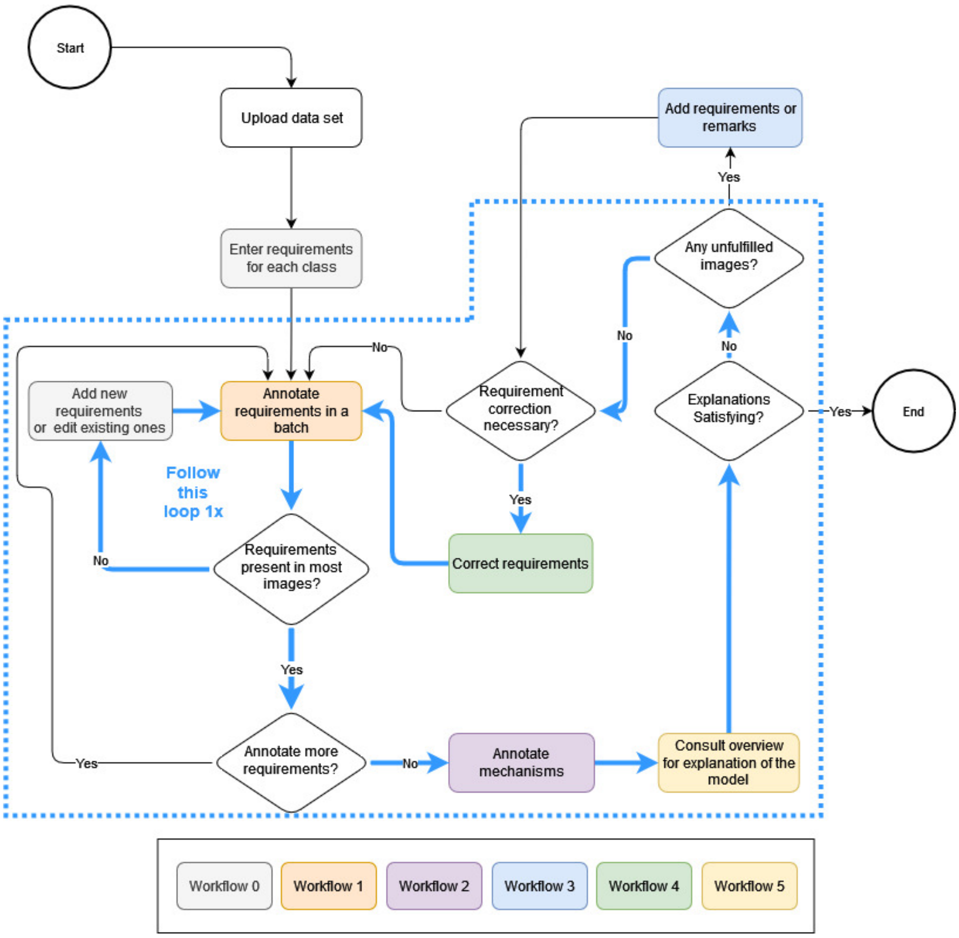



Figure 5.1: A flowchart on how Brickroutine is used in evaluating informativeness

# 5.3. Experimental Results

This section lists the results of the evaluation with respect to informativeness. An extensive table with results is given in Appendix A.

## 5.3.1. First round

The full results are listed in section A.1. Table 5.2 shows that for each class, from a total of 20, for 9,8, and 16 images, a matching mechanism was found.

| **Class (requirements)** | **Requirement in image** | **Mechanism in heatmap** | **Mechanism not in heatmap** | **Requirement not in image** | **Unnannotated images** |
|---|---|---|---|---|---|
| **American lobster (8)** | 18 (18%) | 9 (9%) | 9 (9%) | 2 (2%) | 80 (80%) |
| **great white shark (6)** | 19 (19%) | 8 (8%) | 11 (11%) | 1 (1%) | 80 (80%) |
| **tench (8)** | 18 (18%) | 16 (16%) | 2 (2%) | 2 (2%) | 80 (80%) |

Table 5.2: Requirements overview table after one round for *Sea Creatures*

### Class specific observations for american Lobster

The three most annotated mechanisms with their occurrences are `claw, red shell (5)`, and `red shell (4)`, and `claw, front body part (2)`. We notice that the concept *Red shell* occurs 11 times in the image, however, only in 3 cases, the heatmap appears to be covering the entire concept. This is an indication that the model might have learned a concept of finer granularity. If we click on this mechanism in the overview pane of the interface, the observe the images as depicted in Figure 5.2.

Additionally, we notice the concept *Front body part* appearing in multiple annotations, including images where the lobster's shell is not red on the image. Subsequently, it looks like the part of starting at the lobster's head up until the start of its tail is highlighted by the heatmaps. In lobster terms, this is referred to as the carapace ("*Body piece from eyes to start of tail*") [2]. Even for pictures that contain multiple lobsters, that specific part seems to be highlighted. Therefore, we decide the next action to be changing the initial requirement `red shell` to contain just the concept `carapiece`. Additionally, we adjust the weights according to the number of images that contained the requirement to prevent unnecessary overhead during the annotation of the next batches of images.

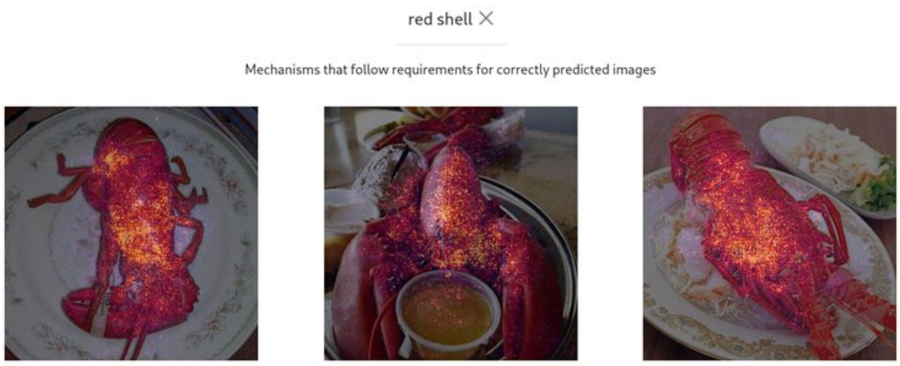



Figure 5.2: Images for which the mechanism `red shell` was annotated

## Class specific observations for great white shark

We notice that eight images have requirements verified in the heatmap. The concepts `mouth`, `snout`, `gill openings`, `dorsal fin`, `pectoral fin` are present in the verified requirements. In addition, both the concepts `eye` (Figure 5.3) and `nostril` have occurrences in mechanisms that did not match requirements . This is an indication that we should incorporate it in the requirements. As a result, we add requirements with these concepts.


Figure 5.3: An image for which the concept *eye* was highlighted by the saliency map

## Class specific observations for tench

Requirements made up of combinations of the concepts *olive skin*, *dark rounded finds*, *red eye* were verified in 16 of the 18 images. Consulting the images (Figure 5.4) shows that these images are outlining these concepts in its entirety, leaving no reason to refine them. Additionally, the only other annotated concept (`camouflage clothing`) refers to humans that were part of these images when fishing. For this class, we leave the requirements with their weights as is.

[2] https://lobsteranywhere.com/seafood-savvy/lobster-lingo/


Figure 5.4: Images for which the mechanism `olive skin, dark rounded fins` was annotated

### Consequences of changing the requirements

Because we modified most requirements, images that contained the changed requirement, need to be annotated again, because there is no way of knowing that the new definition is still present in the image. As a result of the actions described above, 28 images need to be annotated again. Please note that in case a custom mechanism is required again, Brickroutine will automatically resolve the previously annotated mechanism (algorithm 3).

## 5.3.2. Second round

The full results are listed in section A.2. In Table 5.3, we notice that at this stage, for 105 of the 120 annotated images we are able to to verify a requirement. In the case of American Lobster and tench, the overview pane already lists requirements that show consistent model behaviour.

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **American lobster (8)** | 36 (36%) | 28 (28%) | 8 (8%) | 4 (4%) | 60 (60%) |
| **great white shark (7)** | 32 (32%) | 17 (17%) | 15 (15%) | 8 (8%) | 60 (60%) |
| **tench (8)** | 37 (37%) | 25 (25%) | 12 (12%) | 3 (3%) | 60 (60%) |

Table 5.3: Requirements overview table after two rounds for *Sea Creatures*

### Class specific observations for american Lobster

The requirement `claw, carapiece` is present in 17 of the 36 images. However, not all heatmaps were showing coverage of the entire concept. When we inspect images where the concept `claw` was annotated, we notice that not the entire claw was covered in most cases but rather a smaller part (Figure 5.5).

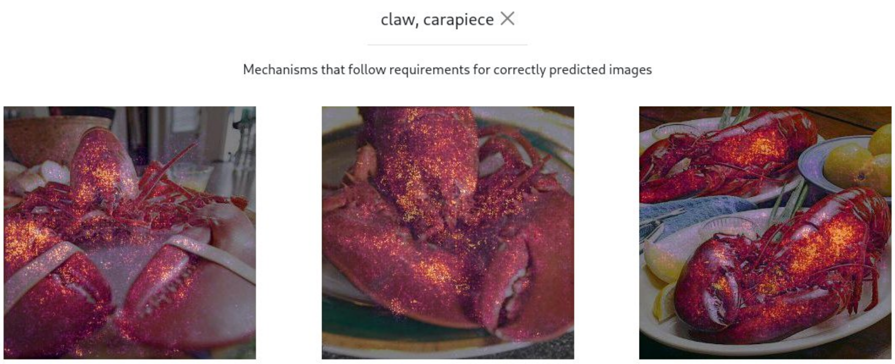



Figure 5.5: The heatmaps show a smaller part of the claw highlighted

As a result, we add the requirement `claw texture, carapiece` and the separate concept `claw texture` with a high weight as new requirements. Note that we could have changed the requirement at this point, but chose to add a new one because otherwise a lot of annotations would be lost without eventually gaining more knowledge about the model's behaviour than we have at this point.

### Class specific observations for great white shark

We notice that 19 unique mechanisms are listed, which is more than for other classes. While it might be a sign of diversity, a closer inspection suggests that (1) most are combinations of the different concepts, and (2) they can roughly be split up between concepts that either are a close up of a shark's head or more a distant composition where the entire body and the fin outline is shown. Based on this requirements, we already have a fairly good idea of what the model behaves like and therefore we do not alter the requirements. We do alter the weights with the expectation to save annotation costs since the fourth and fifth requirements are the most verified ones.

### Class specific observations for tench

We see that for every verified requirement but one, the entire concept is highlighted by the heatmaps. This is a sign that our requirements are at an appropriate level of granularity. For other concepts that are not part of any requirement, we observe that most of them are part of a background concept that is appearing in some images but are not semantically related to the tench itself (human, dog, camouflage clothing). As a result, we choose not to alter anything and proceed to the next iteration.

## 5.3.3. Third round

After three rounds, it seems like we arrived at requirements and concepts that can be used to approximate the model's inner reasoning mechanism (Table 5.4). In general, we notice a trend that the weight factor of a requirement is proportional to the relative amount of occurrences of that requirement, both in the image itself and in the heatmap. The full overview table and mechanisms are given in section A.3.

| **Class (requirements)** | **Requirement in image** | **Mechanism in heatmap** | **Mechanism not in heatmap** | **Requirement not in image** | **Unnannotated images** |
|---|---|---|---|---|---|
| **American lobster (10)** | 56 (56%) | 44 (44%) | 12 (12%) | 4 (4%) | 40 (40%) |
| **great white shark (7)** | 46 (46%) | 22 (22%) | 24 (24%) | 14 (14%) | 40 (40%) |
| **tench (8)** | 55 (55%) | 38 (38%) | 17 (17%) | 5 (5%) | 40 (40%) |

Table 5.4: Requirements overview table after three rounds for *Sea Creatures*

### Class specific observations for american Lobster

Recall that in the previous round subsection 5.3.2, we fine-tuned the requirement of `claw` to `claw texture`. At this point, the requirement `claw texture, carapace` is verified in the heatmap 12 times. The entire concept was present for every occurrence, indicating that the level of granularity of this concept seems appropriate.

### Class specific observations for great white shark

The most occurring requirement is `mouth, snout, gill openings, eye, nostril` followed by one that is made up of a subset of these concepts. Inspecting individual images on the overview page does not suggest that we can refine the requirements any further. In the end, we do see 27 unique mechanisms. However, the majority of them are made up of combinations of the concepts we defined. At this point, the model seems to have learned multiple angles and viewpoints. Some are more from a close-up point with concepts like `nostril` and `eye` being represented, whereas other views contained an image of the shark as a whole where concepts like `dorsal fin` and `caudal fin` were highlighted.

### Class specific observations for tench

The most verified requirement is *olive skin, dark rounded fins, red eye*, occurring 16 times out of the 60 images that we addressed so far. In fact, combinations of these concepts are present 38 times ($\approx 63\%$ ) and are solely responsible for any of the verified requirements. Concepts that are present in images that do not follow requirements are either too insignificant to be transformed into a requirement (`gill cover`) or do semantically not relate the class *tench* (`human`, `dog` or `camouflage clothing`).

### Actions after this round

Since both the coverage of the requirements and the granularity of the respective concepts seem appropriate, we do not modify any requirements after this round. We do however adjust some weights in correspondence with the frequency a requirement occurs. Additionally, since the requirements and their weights did not require any breaking changes, we choose to annotate the last 60 120 images (40 per class) in one batch in the next round.

### 5.3.4. Fourth round

The full results are included in section A.4. We notice that for each class (*American lobster*: `claw, carapiece & claw texture, carapiece`, *Great White Shark*: `mouth, snout, gill openings, eye, nostril & mouth, snout, eye, nostrill & dorsal fin, pectoral fin`, *Tench:* `olive skin, dark rounded fins, red eye & olive skin, dark rounded fins & olive skin, red eye`), the most two occurring mechanisms are equivalent to the one in our previous round. This suggests that we did not gain a lot more information while we still had annotation expenses.

## 5.4. Overview

Table 5.5 shows the duration for each stage and a requirements overview table from Brickroutine is given in Table 5.6. In an annotation time of 103 minutes, we got mechanisms for 149 images, with for each class some distinguishing trends that can explain the model's behaviour. The total amount of annotation steps is 623, resulting from some images losing their annotated requirement when they change (workflow 4, subsection 3.5.5). Therefore, on average we have:

$$\text{Avg. Annotations/Image} = \frac{623}{149} \approx 4.2$$

| **Action** | **Time** | **Images** | **avg. time/image** |
|---|---|---|---|
| Requirement annotation: first round | 0:09:44 | 60 | 0:00:10 |
| Mechanism annotation: first round | 0:09:57 | 57 | 0:00:10 |
| Requirement annotation: second round | 0:11:10 | 88 | 0:00:08 |
| Mechanism annotation: second round | 0:13:41 | 78 | 0:00:11 |
| Requirement annotation: third round | 0:06:34 | 60 | 0:00:07 |
| Mechanism annotation: third round | 0:10:35 | 52 | 0:00:12 |
| Requirement annotation: fourth round | 0:19:34 | 120 | 0:00:10 |
| Mechanism annotation: fourth round | 0:22:07 | 108 | 0:00:12 |
| **total** | 1:43:22 | 623 | 0:00:10 |

Table 5.5: Statistics about the time taken to do the annotations

| **Class (requirements)** | **Requirement in image** | **Mechanism in heatmap** | **Mechanism not in heatmap** | **Requirement not in image** | **Unnannotated images** |
|---|---|---|---|---|---|
| **American lobster (10)** | 96 (96%) | 71 (71%) | 25 (25%) | 4 (4%) | 0 |
| **great white shark (7)** | 80 (80%) | 41 (41%) | 39 (39%) | 20 (20%) | 0 |
| **tench (8)** | 89 (89%) | 60 (60%) | 29 (29%) | 11 (11%) | 0 |

Table 5.6: Requirements overview table after four rounds for *Sea Creatures*

In Table 5.6 we notice that for the class american lobster, 96% of the images have the eventual requirement. Analogously, this number is 80% and 89% for great white shark and tench. For 71%, 41% and 60% the mechanism was also verified by our approach, meaning that for these cases an approximation of the model's mechanism was extracted. Figure 5.6 show how these numbers change over time.

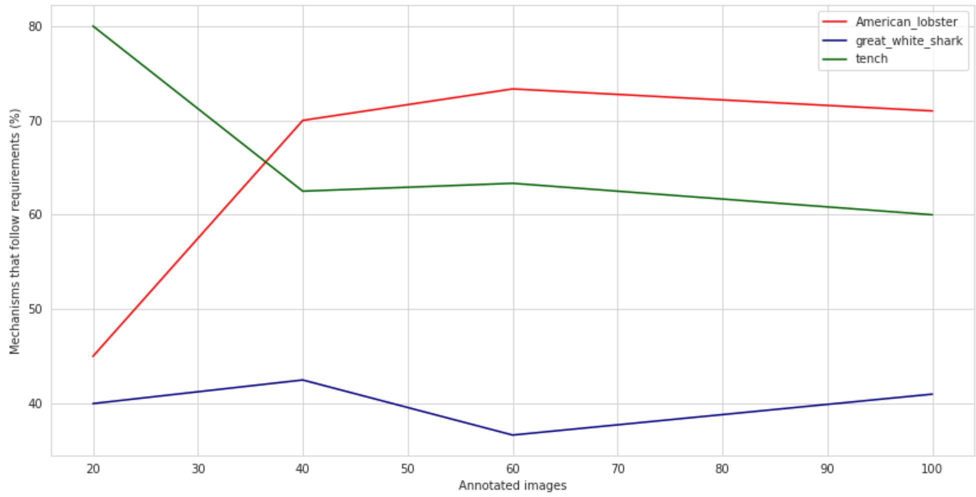



Figure 5.6: Annotated images and verified requirements

## 5.5. Discussion

We noticed a few things as a result of this evaluation:

- The percentage of verified requirements for each class did not change significantly after annotating images after round 2. The top verified mechanisms did not change either, implying that we were able to explain the model's decisions as good after 2 rounds (with a total of 40 images per class and 283 total annotations) in 45 minutes as after 4 rounds (with a total of 100 images per class and 623 total annotations) in 103 minutes.

- The design of our workflows enabled us to obtain finer-grained requirements for *American Lobster* (`carapiece`) and *Great White Shark* (`gill openings, dorsal fin, pectoral fin`). This confirms the suitability of splitting up our evaluation into rounds and that for these cases, the number is appropriate. However, we enforced this tactic ourselves and an improvement for the system would be to tunnel the user more into using this approach by making adapting the interface to this so that to results of each round could be consulted separately afterwards.

- As a result of changing the requirements and adjusting the weights accordingly, we expect the user to spend less time annotating the requirements since higher weighting ones appear first. The idea of this is that the user can annotate the most likely requirement by simply clicking yes in the interface (Figure 3.8). We noticed that as a result, the average annotation time per image was not significantly influenced by that as Table 5.5 depicts. The risk that this approach introduced is that a select requirement could not be verified in the subsequent mechanism annotation step and another requirement with a lower weight, does match the eventual annotated mechanism. As a result, more time is spent in the mechanism annotation step, because the user has to cherry-pick the concepts manually. This suggests that future work should investigate the efficiency of not limiting ourselves to annotating just one requirement.

- An improvement would be to include an additional step to transform an annotated mechanism into a requirement. Since after annotating the mechanisms the user has obtained new knowledge about how the model possibly behaves, we experienced that we added requirements in which all the concepts appeared in a previously annotated mechanism. Therefore, an interface element to picking a requirement from the mechanisms saves time and decreased the odds of ambiguous requirements.

- In our current setup, requirements are entered purely based on domain knowledge and not linked to the specific set of images, an improvement therefore would be to show a randomly taken sample to the user to guide them in the initial annotation process.

In conclusion, we think that the design of our workflows and interfaces already helps the user in assessing the inner workings of an adequately performing model, and that it can be further expanded by tailoring towards the needs of this specific scenario by following the made suggestions.

# 6
# Evaluation: Validity

This chapter describes the design, results and discussion of the evaluation with respect to the *validity* of Brickroutine.

## 6.1. Goals

In addition to the previous chapter, we can imagine a use case in which users want to know how valid the predictions of a model are. If we recall the initial problem of this thesis that is described in section 1.1:

> *"Humans have limited possibilities to disassemble a machine learning model and verify if the algorithm followed a line of reasoning that is comprehensible so that decisions can be assessed in terms of fairness, robustness, and trustworthiness"*

To start with an example, suppose that a model was trained on a data set and the accuracy on the test data set is 70 %. We imagine the developers of the model want to know at least two things:

1. Can the causes for confusion between classes be traced back to semantic concepts?
2. Are the right predictions based on concepts that actually belong to the respective classes?

In short, we can state we want to investigate how valid these predictions are with respect to semantic concepts that we would use to describe those classes.

**Definition.** ***Validity*** *is the degree to which the system expresses information that can be used to assess if the model makes valid predictions, meaning that it bases its reasoning on aspects of the input data that are in line with training procedures and expectations. Can wrong predictions be explained (model predicted class $x$ for class $y$ because it appeared to have learned concepts $a_0 \ldots a_{n-1}$)? If we can find biases in the data set or reason why the model is behaving a certain way, we should have some pointers about the models' (in)validity.*

By making use of our iterative approach, the costs form an important metric too. Especially in use cases with lower model performance because the users are more likely to annotate custom requirements, leading to higher annotation times.

**Definition.** ***Costs****: We measure the costs in time that a human spends on executing actions in Brickroutine. For every specific annotation step that involves any human actions, we keep track of the time it takes to complete that action. Additionally, the number of images at each step (if it involves annotating images) will be kept track of so that we can plot it against the amount of annotated images and possibly present some trade-offs.*

## 6.2. Experimental Setup

We use a data set *Birds*, consisting of 494 images in 10 classes, with each class having 49 or 50 images. This data set is known to have biases in some classes, which is why we use this specific data set to address the validity and attempt to obtain explanations from it. This model is trained on imagenet and was subsequently trained by using training data consisting of 100 images for each class.

This data set has not only the right predictions. In fact, quite often a wrong prediction was made. The distribution of the predicted class for this data set is shown in Table 6.1. In this table, only combinations of 5 or more occurrences are shown and each incorrect prediction is assigned an identifier $M_i$ to use later in the evaluation. Given this distribution, we can use the images of this data set to evaluate if we can identify reasons for any of the incorrect predictions $M_i$. For example, in the case of *gila woodpecker*, $11$ times *downy woodpecker* was predicted. Likewise, the model mistook a *bufflehead* for a *hooded merganser* $12$ times. The concepts in these input images are sometimes similar since the high degree of visual similarity between different birds.

| True label | Id | Predicted label | Count |
|---|---|---|---|
| **gila woodpecker** | | gila woodpecker | 25 |
| | M1 | downy woodpecker | 11 |
| **hairy woodpecker** | | hairy woodpecker | 38 |
| | M2 | downy woodpecker | 8 |
| **american goldfinch** | | american goldfinch | 48 |
| **monk parakeet** | | monk parakeet | 45 |
| **lesser goldfinch** | | lesser goldfinch | 43 |
| | M3 | american goldfinch | 5 |
| **pine grosbeak** | | pine grosbeak | 31 |
| | M4 | american goldfinch | 7 |
| **mandarin duck** | | mandarin duck | 35 |
| | M5 | lesser goldfinch | 5 |
| | M6 | hooded merganser | 5 |
| **bufflehead** | | bufflehead | 36 |
| | M7 | hooded merganser | 12 |
| **downy woodpecker** | | downy woodpecker | 45 |
| **hooded merganser** | | hooded merganser | 40 |
| | M8 | bufflehead | 5 |

Table 6.1: Model and Prediction distribution with 5 or more occurrences

To have a bit of background knowledge, the jargon depicted in Figure 6.1 is used to define unambiguous semantic concepts. The initial requirements we use for each class are shown in Table 6.2.

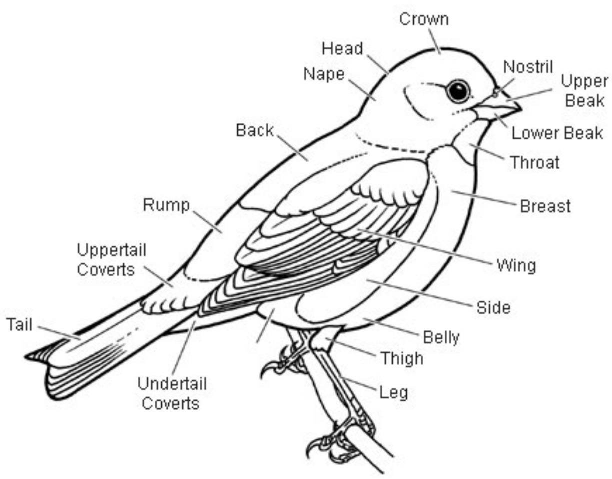



Figure 6.1: The parts of a bird to be used [24]

| Class | Requirements |
|---|---|
| **american_goldfinch (3)** | yellow breast, yellow belly, yellow back, black wings |
| | yellow breast, yellow belly, yellow back |

Continued on next page

continued from previous page

| Class | Requirements |
|---|---|
| | black wings |
| **bufflehead (4)** | white spot, rainbow crest |
| | dark head, white spot |
| | black wings, white feathers |
| | black wings, grey feathers |
| **downy_woodpecker (4)** | black white wing patches, white breast |
| | black white wing patches |
| | white breast |
| | red crown |
| **gila_woodpecker (4)** | red crown, green neck, green belly, green breast, black white striped wings |
| | green neck, green belly, green breast, black white striped wings |
| | black white striped wings |
| | green neck, green belly, green breast |
| **hairy_woodpecker (5)** | black wings, white breast |
| | white breast |
| | black wings |
| | throat stripe |
| | red crown |
| **hooded_merganser (3)** | black crest with white spot [1] |
| | cinnamon crest [1] |
| | brown sides |
| **lesser_goldfinch (3)** | black crown, yellow breast, yellow belly, yellow back |
| | yellow breast, yellow belly, yellow back |
| | black wings, lighter wing patches |
| **mandarin_duck (5)** | rainbow crest |
| | brown feathers |
| | white stripe below eye |
| | long brown neck feathers |
| | golden sides |
| **monk_parakeet (3)** | green feathers, light breast, light crown |
| | green feathers |
| | light crown, light throat |
| **pine_grosbeak (3)** | pink feathers, grey wings |
| | grey feathers, orange head |
| | heavy chest |

Table 6.2: Initial requirements for bids data set

Additionally, this data set was trained specifically to introduce some biases in the data. These biases are described in Table 6.3. The goal of injecting biases $B1$-$B4$ is to create expectations that we can verify later on. Biases $B5$ and $B6$ are characterized by species of birds that are visually very similar. We want to use Brickroutine to determine if the model “randomly” mistakes these classes or if some consistent patterns can be extracted. Bias $B7$ functions as a baseline because a monk parakeet has some visually unique characteristics in comparison to the other birds belonging to this data set.

| Id | Class | Training data | Test data | Expectation |
|---|---|---|---|---|
| B1 | Gila woodpecker | Primarily images with a cactus | Images with and without a cactus | The concept cactus should appear in mechanisms of correct predictions and should not appear in mechanisms of incorrect predictions for gila woodpecker |
| B2 | Pine grossbeak | Only male (pink) variants | Male (pink) and female (orange) variants | Mostly correct prediction for the male variants and incorrect predictions for the female variants |
| B3 | Mandarin duck | Only male (colorful) variants | Male (colorful) and female (monochromatic) variants | Correct predictions on the male variant. Incorrect predictions on the females that might get confused with other species. |

Continued on next page

[1] https://www.allaboutbirds.org/guide/Hooded_Merganser/id

continued from previous page

| Id | Class | Training data | Test data | Expectaction |
|---|---|---|---|---|
| B4 | Bufflehead | Not too many sitting idly in the water, more are flying or standing up | Both idly in the water as standing up and flying | Male versions look like hooded merganser or mandarin duck, the ones that are sitting idly in the water could be classified as one of those. |
| B5 | Downy woodpecker & Hairy woodpecker | - | - | Since these two species look very much alike, confusion should ideally only occur between these two classes |
| B6 | American goldfinch & Lesser goldfinch | - | - | Since these two species look very much alike, confusion should ideally only occur between these two classes |
| B7 | Monk parakeet | - | - | The concepts that are exclusive to this species, green colours and blue wingtips should appear at the end |

Table 6.3: The bias in the training data of *Birds*

### 6.2.1. Approach

We follow the iterative procedure from section 3.6 and gather metrics after each iteration. To investigate how the metrics change over time and see the added value of each iteration, we will annotate until no unannotated images remain. The parts of Brickroutine that are covered are within the border in Figure 6.2. We choose to leave Workflow 3 out of the evaluation because our data sets are of significant size and this will result in overhead costs. Even though the additional results might be converging, we continue to do it for all the possible images so that we see can compare the gained knowledge against the extra costs. The iteration size in images will be set to roughly 20% of the images per class, resulting in 20 images for *Birds*

### 6.2.2. Metrics

With the nature of the evaluation goals in mind, two types of statistics will be gathered:

- **Found explanations:** For each combination of incorrectly predicted images in Table 6.1, we will add an explanation if the system has provided us with sufficient information to do so. If there is sufficient information in the system after running an iteration, we will use this information and add an explanation for a specific misprediction $M_n$. This explanation is in the form of a rule in human language e.g. "the model confused a gila woodpecker with a woodpecker because the wing patches are visually similar. Eventually, by keeping track of the iteration number in which this explanation was found, we can state how many iterations are required for this type of data set.

- **Traced biases:** For each bias that was injected into the data set, we have an expectation (Table 6.3). If there is sufficient information in the system after running an iteration, we will use this information and add an explanation for a specific bias $B_n$. Again, we can use the round in which this information was obtained to estimate the amount of annotated images that are required to establish the goals. We gather this by appending a column to the table and fill it with yes/no values depending on if the expectation is verified by the information from Brickroutine.

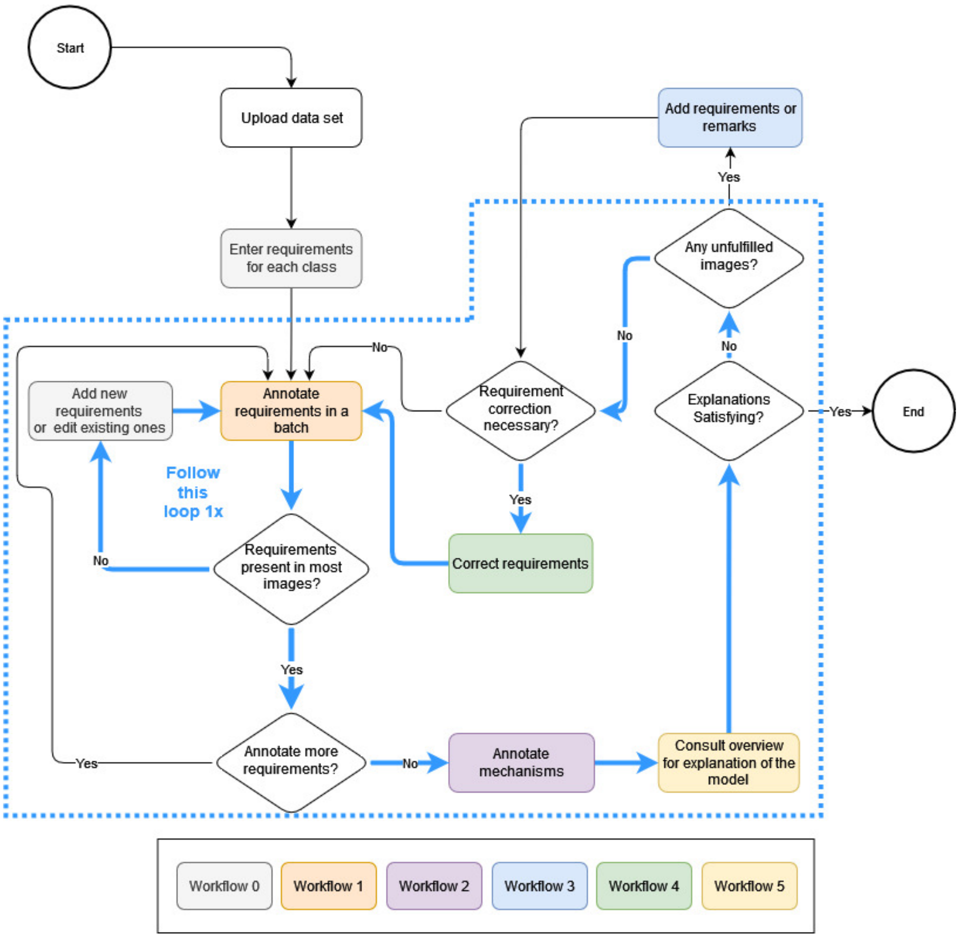



Figure 6.2: A flowchart on how Brickroutine is used in evaluating validity

# 6.3. Experimental Results

This section describes the evaluation round for this data set with the goal to find biases and explanations in the model. An extensive table with results is given in Appendix B.

## 6.3.1. First round

The full results are given in section B.1. What immediately stands out is that despite the requirements being present in all the images, for only three classes (*American Goldfinch*, *Monk Parakeet), and Pine Grosbeak*, requirements are entirely verified in the heatmap. Two of these classes are classes with the highest prediction accuracy. An overview is given in Table 6.4. In this table we see that for 8 of the 100 images a mechanism is found in this stage.

Investigating further, we notice a lot of annotated concepts that are not part of the semantics we would typically use to describe a bird such as `tree`, `sky`, `cactus` and water. For instance, the concepts depicted in Figure 6.3 show that irrelevant concepts are learnt.

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **american goldfinch (3)** | 10 (20%) | 3 (6%) | 7 (14%) | 0 | 39 (80%) |
| **bufflehead (4)** | 10 (20%) | 0 | 10 (20%) | 0 | 39 (80%) |
| **downy woodpecker (4)** | 10 (20%) | 0 | 10 (20%) | 0 | 40 (80%) |
| **gila woodpecker (4)** | 10 (20%) | 0 | 10 (20%) | 0 | 40 (80%) |
| **hairy woodpecker (5)** | 10 (20%) | 0 | 10 (20%) | 0 | 40 (80%) |
| **hooded merganser (3)** | 10 (20%) | 0 | 10 (20%) | 0 | 40 (80%) |
| **lesser goldfinch (3)** | 9 (18%) | 0 | 9 (18%) | 1 (2%) | 39 (80%) |
| **mandarin duck (5)** | 10 (20%) | 0 | 10 (20%) | 0 | 40 (80%) |
| **monk parakeet (3)** | 10 (20%) | 3 (6%) | 7 (14%) | 0 | 39 (80%) |
| **pine grosbeak (3)** | 10 (21%) | 2 (4%) | 8 (17%) | 0 | 38 (79%) |

Table 6.4: Requirements overview table after one round for *birds*

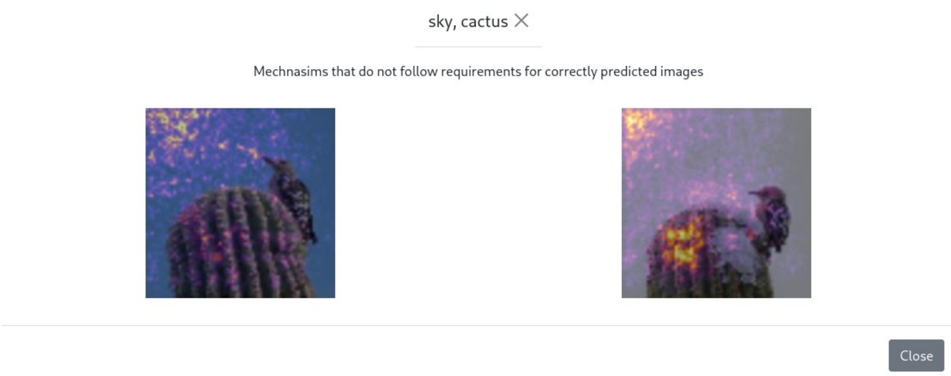



Figure 6.3: *Gila woodpecker* for which the concepts `sky` and `cactus` were annotated

After this round first round of annotating both the requirements and the heatmaps in the image, we have a list with initial hypotheses about misclassified images in Table 6.5. Note that not all combinations of misclassified images are yet present. Since the goal of this evaluation is to gain information about misclassified images and the requirements are appropriate, we do not modify them and proceed to annotate more images without modifying the requirements.

Additionally, from the biases described in Table 6.3, at this stage we are able to verify 4, although only a few images exist to support these conclusions yet. These are listed in Table 6.6.

| Id | Label | Predicted | Concepts | Explanation |
|---|---|---|---|---|
| M1 | gila woodpecker | downy woodpecker | black white striped wings, sky, green crown, tree | The wing patches are similar. Both males have a red crown. The concept tree is annotated for most correctly classified downy woodpeckers and not at all for gila woodpecker |
| M2 | hairy woodpecker | downy woodpecker | tree, black wings, white breast, throat stripe, sky, tree | The wing patterns of these two are similar, Both males have a red crown. For both classes, the model has learnt tree. |
| M4 | pine grosbeak | american goldfinch | orange head, tree, green background, grey feathers | The mispredicted images are images in trees, were most images have a snow background. The female pine grosbeak looks more like an american goldfinch because of the absence of pink colors |
| M5 | mandarin duck | lesser goldfinch | golden sides, water, soil, background ornament | The duck is smaller than on most pictures and there is a lot of water. The gold color might look like the yellow of the finch. |
| M6 | mandarin duck | hooded merganser | neck, water, white stripe below eye | The female mandarin duck looks like the female hooded merganser |
| M7 | bufflehead | hooded merganser | water, grey feathers, white spot, neck | For both classes, the concept water was learnt, The females of these two birds look alike |

Table 6.5: Hypotheses about misclassified images

| Id | Class | Expectation | Verified |
|---|---|---|---|
| B1 | Gila woodpecker | The concept cactus should appear in mechanisms of correct predictions and should not appear in mechanisms of incorrect predictions for gila woodpecker | Yes |
| B2 | Pine grossbeak | Mostly correct prediction for the male variants and incorrect predictions for the female variants | Yes |
| B4 | Bufflehead | Male versions look like hooded merganser or mandarin duck, the ones that are sitting idly in the water could be classified as one of those. | Yes |
| B7 | Monk parakeet | The concepts that are exclusive to this species, green colours and blue wingtips should appear at the end | Yes |

Table 6.6: Verified biases after one round

An illustration of bias $B7$ from Table 6.6, is given in Figure 6.4 that shows a concept with green colours that is exclusive to the monk parakeet in this data set.

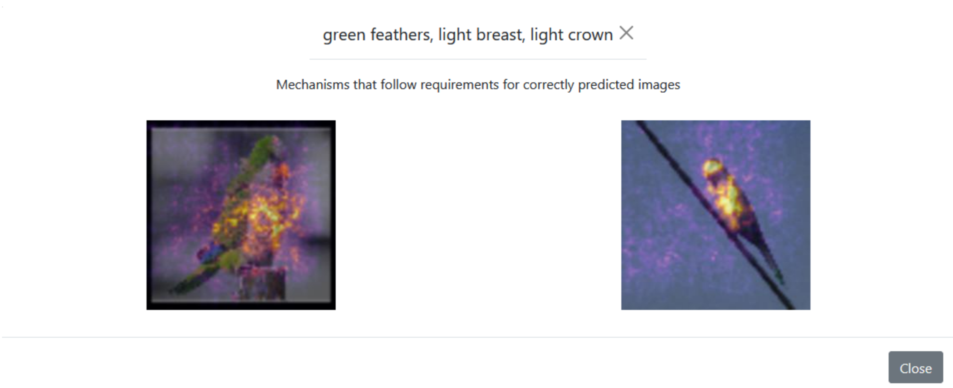



Figure 6.4: Concepts that are exclusive for monk parakeet

### 6.3.2. Second round

The full results can be consulted in section B.2. After this round of annotation again 10 images for all 10 classes, the requirement overview table is shown in Table 6.7.

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **american goldfinch (3)** | 20 (41%) | 4 (8%) | 16 (33%) | 0 | 29 (59%) |
| **bufflehead (4)** | 20 (41%) | 0 | 20 (41%) | 0 | 29 (59%) |
| **downy woodpecker (4)** | 20 (40%) | 0 | 20 (40%) | 0 | 30 (60%) |
| **gila woodpecker (4)** | 20 (40%) | 0 | 20 (40%) | 0 | 30 (60%) |
| **hairy woodpecker (5)** | 20 (40%) | 2 (4%) | 18 (36%) | 0 | 30 (60%) |
| **hooded merganser (3)** | 20 (40%) | 0 | 20 (40%) | 0 | 30 (60%) |
| **lesser goldfinch (4)** | 19 (39%) | 1 (2%) | 18 (37%) | 1 (2%) | 29 (59%) |
| **mandarin duck (6)** | 20 (40%) | 0 | 20 (40%) | 0 | 30 (60%) |
| **monk parakeet (3)** | 19 (39%) | 3 (6%) | 16 (33%) | 1 (2%) | 29 (59%) |
| **pine grosbeak (3)** | 20 (42%) | 3 (6%) | 17 (35%) | 0 | 28 (58%) |

Table 6.7: Requirements overview table after two rounds for *birds*

We make some general observations after having annotated 20 images per class:

- 58 of the 185 mechanisms in total contain the concept `tree`. This suggests that the model learned that concept and used it for the prediction of multiple classes (*hairy woodpecker, gila woodpecker, downy woodpecker*).
- The model learnt the same concepts in the classes that are highly similar (e.g. hairy and downy woodpecker). Given that most predictions were right, it either learnt a concept of a finer granularity or co-existence with irrelevant background concepts.
- The concept `green background` was used to predict *american goldfinch* and *lesser goldfinch*. This is both for correct and incorrect predictions.

We can extend our table of explanations for incorrect predictions with the rows from Table 6.8. We have some explanations for all our major incorrect predictions.

| Id | Label | Predicted | Concepts | Explanation |
|---|---|---|---|---|
| M3 | lesser goldfinch | American goldfinch | black wings, yellow belly, sky | Both the male and female are visually similar and in both classes the concept tree was present |
| M8 | hooded merganser | bufflehead | cinnamon crest, water, grey belly, dark feather texture | The head, crest of the females are similar. Hooded merganser contains more annotations for black crest, which is exclusively featured in the male birds |

Table 6.8: additional hypotheses about misclassified images after two rounds

Moreover, we can add the two rows from Table 6.9 to our table of verified biases. For *mandarin duck*, no correct predictions contained the concept `grey feathers` (that female variants have) while this does occur in predictions for other *lesser goldfinch* and *gila woodpecker*. Additionally, for all incorrect predictions of *lesser goldfinch*, the predicted class was *american goldfinch*, verifying B6 in Table 6.9. An example for this case is shown in figure Figure 6.5, where we see that apart form a correct concept that both *american goldfinch* and *lesser goldfinch* feature, the model also learnt `tree` and `sky`.

| Id | Class | Expectation | Verified |
|---|---|---|---|
| B3 | Mandarin duck | Correct predictions on the male variant. Incorrect predictions on the females that might get confused with other species. | Yes |
| B6 | American goldfinch & Lesser goldfinch | Since these two species look very much alike, confusion should ideally only occur between these two classes | Yes |

Table 6.9: additionally verified biases after two rounds

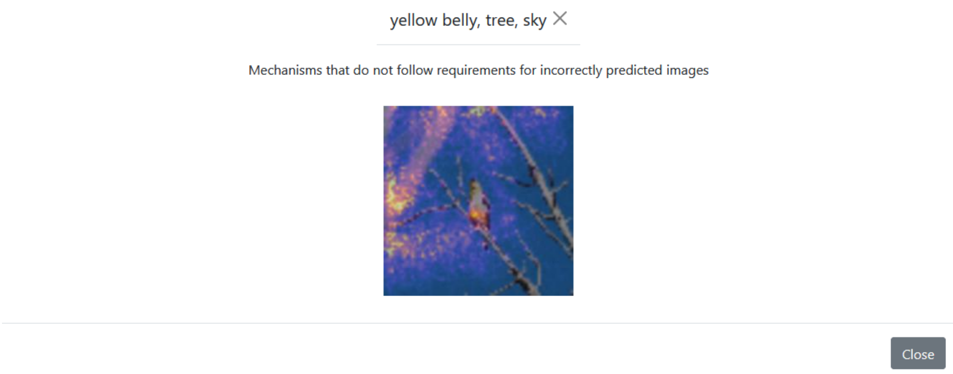



Figure 6.5: The model predicted *american goldfinch* instead of *lesser goldfinch*

Again, since the main goal is to characterize bugs in the predictions and assess the informativeness we decide to leave the requirements as is. We proceed to the next round with again 10 images for each of the 10 classes.

### 6.3.3. Third round

The full results are in Table B.5 and a requirement overview table is given in Table 6.10. In summary, we conclude that even though some mechanisms do have more than 1 occurrence now, still a vast amount (289/307) of unique, custom annotated mechanisms exist per class. If we inspect the mechanisms more closely, we see that there are still a lot of concepts in the mechanisms that seem irrelevant for those specific classes. For instance, the `tree` is mentioned 76 time and `sea` is part of the mechanism 51 times.

| **Class (requirements)** | **Requirement in image** | **Mechanism in heatmap** | **Mechanism not in heatmap** | **Requirement not in image** | **Unnannotated images** |
|---|---|---|---|---|---|
| **american goldfinch (3)** | 29 (59%) | 4 (8%) | 25 (51%) | 1 (2%) | 19 (39%) |
| **bufflehead (4)** | 32 (65%) | 0 | 32 (65%) | 0 | 17 (35%) |
| **downy woodpecker (4)** | 30 (60%) | 1 (2%) | 29 (58%) | 0 | 20 (40%) |
| **gila woodpecker (4)** | 30 (60%) | 2 (4%) | 28 (56%) | 0 | 20 (40%) |
| **hairy woodpecker (5)** | 30 (60%) | 4 (8%) | 26 (52%) | 0 | 20 (40%) |
| **hooded merganser (3)** | 30 (60%) | 3 (6%) | 27 (54%) | 0 | 20 (40%) |
| **lesser goldfinch (4)** | 29 (59%) | 2 (4%) | 27 (55%) | 1 (2%) | 19 (39%) |
| **mandarin duck (6)** | 29 (58%) | 1 (2%) | 28 (56%) | 1 (2%) | 20 (40%) |
| **monk parakeet (3)** | 29 (59%) | 5 (10%) | 24 (49%) | 1 (2%) | 19 (39%) |
| **pine grosbeak (3)** | 30 (63%) | 4 (8%) | 26 (54%) | 0 | 18 (38%) |

Table 6.10: Requirements overview table after three rounds for *birds*

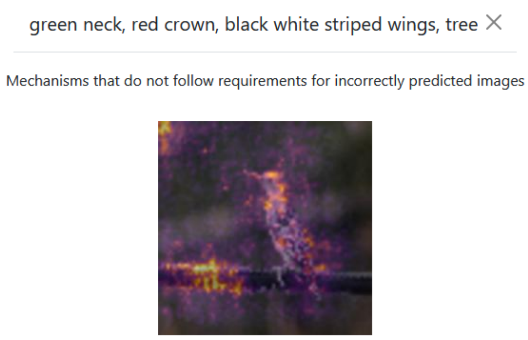
Since our search for explanations for incorrect predictions was completed before this round of evaluation, we have one left for assessment. The row shown in Table 6.11 explains that we can not verify $B5$. Two reasons for this exist: 1) For the class *Gila woodpecker*, 11 times the class *downy woodpecker* is predicted by the model and 2) this confusion is caused by concepts that refer to the birds' attributes. We already had the knowledge of reason 1 since the outcome of predictions can be seen in Brickroutine right after uploading a data set (Figure 3.3). However, after doing 3 iterations we also have knowledge that this indeed is based on semantic concepts. If we consult our overview pane, we see that:

- A *gila woodpecker* was incorrectly classified as a *downy woodpecker* based on the concepts `green neck, red crown, black white striped wings and tree`.
- A gila woodpecker was correctly classified based on the concepts `A red crown, black white striped wings, green breast, green belly`.

The interface from Brickroutine for these cases are shown in Figure 6.6a and Figure 6.6b. From these images (more examples exist), we see that the model did confuse these two classes based on similar concepts.

| Id | Class | Expectation | Verified |
|---|---|---|---|
| B5 | Downy woodpecker & Hairy woodpecker | Since these two species look very much alike, confusion should ideally only occur between these two classes | No |

Table 6.11: additionally verified biases after three rounds




(a) gila woodpecker classified as downy woodpecker


(b) gila woodpecker classified as gila woodpecker

Figure 6.6: The models' confusion about different species of woodpeckers

### 6.3.4. Fourth round

After doing this round, we were not able to explain more incorrect predictions or biases. We annotated 192 more requirements and 186 more mechanisms. We had one explanation for a bias ($B_5$) left to verify but we noticed that the confusion for *Hairy Woodpecker* and *Downy Woodpecker* was not limited to those classes. The final results are listed in Table 6.12. In the fourth column, mechanism not in heatmap, we see that for roughly 87% of the images, a custom mechanism was annotated.

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **american_goldfinch (5)** | 48 (98%) | 7 (14%) | 41 (84%) | 1 (2%) | 0 |
| **bufflehead (4)** | 48 (98%) | 5 (10%) | 43 (88%) | 1 (2%) | 0 |
| **downy_woodpecker (4)** | 50 (100%) | 8 (16%) | 42 (84%) | 0 | 0 |
| **gila_woodpecker (4)** | 50 (100%) | 3 (6%) | 47 (94%) | 0 | 0 |
| **hairy_woodpecker (5)** | 50 (100%) | 4 (8%) | 46 (92%) | 0 | 0 |
| **hooded_merganser (3)** | 50 (100%) | 6 (12%) | 44 (88%) | 0 | 0 |
| **lesser_goldfinch (5)** | 46 (94%) | 3 (6%) | 43 (88%) | 3 (6%) | 0 |
| **mandarin_duck (6)** | 47 (94%) | 4 (8%) | 43 (86%) | 3 (6%) | 0 |
| **monk_parakeet (3)** | 48 (98%) | 5 (10%) | 43 (88%) | 1 (2%) | 0 |
| **pine_grosbeak (3)** | 47 (98%) | 12 (25%) | 35 (73%) | 1 (2%) | 0 |

Table 6.12: Requirements overview table after four rounds for *birds*

## 6.4. Overview

In Table 6.13, we see the **costs** in time for each round of requirement and mechanism annotation. For average annotations per image we observe:

$$\text{Avg. Annotations/Image} = \frac{978}{497} \approx 1.96$$

This number is close to two, which implies that for most images we did only one requirement annotation and one mechanism annotation step. Images that have fewer than two annotations are images that have no matching requirement in workflow 1, with the consequence that they do not appear in the selected images for workflow 2.

| **Action** | **Time** | **Images** | **avg. time/image** |
|---|---|---|---|
| Requirement annotation: first round | 0:05:53 | 100 | 0:00:04 |
| Mechanism annotation: first round | 0:29:33 | 100 | 0:00:18 |
| Requirement annotation: second round | 0:05:36 | 100 | 0:00:03 |
| Mechanism annotation: second round | 0:41:10 | 100 | 0:00:25 |
| Requirement annotation: third round | 0:08:01 | 100 | 0:00:05 |
| Mechanism annotation: third round | 0:35:26 | 100 | 0:00:21 |
| Requirement annotation: fourth round | 0:17:59 | 192 | 0:00:06 |
| Mechanism annotation: fourth round | 0:51:32 | 186 | 0:00:17 |
| **total** | 3:15:10 | 978 | 0:00:12 |

Table 6.13: Statistics about the time taken to do the annotations (birds)

A full table with explanations about the incorrect predictions of this data set is given in Table 6.14. For each combination of incorrectly classified classes with more than 5 occurrences, an explanation was found.

| **Id** | **Label** | **Predicted** | **Concepts** | **Explanation** |
|---|---|---|---|---|
| M1 | gila woodpecker | downy woodpecker | black white striped wings, sky, green crown, tree | The wing patches are similar. Both males have a red crown. The concept tree is annotated for most correctly classified downy woodpeckers and not at all for gila woodpecker |
| M2 | hairy woodpecker | downy woodpecker | tree, black wings, white breast, throat stripe, sky, tree | The wing patterns of these two are similar, Both males have a red crown. For both classes, the model has learnt tree. |
| M3 | lesser goldfinch | American goldfinch | black wings, yellow belly, sky | Both the male and female are visually similar and in both classes, the concept tree was present |
| M4 | pine grosbeak | american goldfinch | orange head, tree, green background, grey feathers | The mispredicted images are images in trees, were most images have a snow background. The female pine grosbeak looks more like an american goldfinch because of the absence of pink colors |
| M5 | mandarin duck | lesser goldfinch | golden sides, water, soil, background ornament | The duck is smaller than on most pictures and there is a lot of water. The gold color might look like the yellow of the finch. |
| M6 | mandarin duck | hooded merganser | neck, water, white stripe below eye | The female mandarin duck looks like the female hooded merganser |
| M7 | bufflehead | hooded merganser | water, grey feathers, white spot, neck | For both classes, the concept water was learnt, The females of these two birds look alike |
| M8 | hooded merganser | bufflehead | cinnamon crest, water, grey belly, dark feather texture | The head, the crest of the females are similar. Hooded merganser contains more annotations for black crest, which is exclusively featured in the male birds |

Table 6.14: Hypotheses about misclassified images

Analogously, A full table with explanations about the expected biases of this data set is given in Table 6.15. We notice that for all but one of the biases, our expectations could be verified.

| Id | Class | Training data | Test data | Expectation | Verified |
|---|---|---|---|---|---|
| B1 | Gila woodpecker | Primarily images with a cactus | Images with and without a cactus | The concept cactus should appear in mechanisms of correct predictions and should not appear in mechanisms of incorrect predictions for gila woodpecker | Yes |
| B2 | Pine grossbeak | Only male (pink) variants | Male (pink) and female (orange) variants | Mostly correct prediction for the male variants and incorrect predictions for the female variants | Yes |
| B3 | Mandarin duck | Only male (colorful) variants | Male (colorful) and female (monochromatic) variants | Correct predictions on the male variant. Incorrect predictions on the females that might get confused with bufflehead | Yes |
| B4 | Bufflehead | Not too many sitting idly in the water, more are flying or standing up | Both idly in the water as standing up and flying | Male versions look like hooded merganser or mandarin duck, the ones that are sitting idly in the water could be classified as one of those. | Yes |
| B5 | Downy woodpecker & Hairy woodpecker | - | - | Since these two species look very much alike, confusion should ideally only occur between these two classes | No |
| B6 | American goldfinch & Lesser goldfinch | - | - | Since these two species look very much alike, confusion should ideally only occur between these two classes | Yes |
| B7 | Monk parakeet | - | - | The concepts that are exclusive to this species, green colors and blue wingtips should appear at the end | Yes |

Table 6.15: The bias in the training data of *Birds*

## 6.5. Discussion

After doing this evaluation, we gained some interesting insights. We will describe them below and evaluate how Brickroutine handled these observations and how we can use this knowledge for improvements to the system.

- We notice that the users end up adding similar concepts for different classes, for instance `yellow breast` was added for *American Goldfinch*, *Lesser Goldfinch*, and `sky` is part of the mechanism for *American Goldfinch*, *Downy Woodpecker*, *Gila Woodpecker*, *Hairy Woodpecker*, *Lesser Goldfinch*, *Monk Parakeet* and *Pine Grosbeak*. This observation currently has to be done manually by consulting the overview pane of Brickroutine. An improvement would be to let Brickroutine output these concepts so that the user is automatically provided with this information because it helps to determine the validity of the predictions.

- From Table 6.12, we notice that on average 11.5% of the mechanisms follow a requirement. This is caused by the fact that quite often a custom mechanism was annotated with irrelevant background concepts that are not part of the requirement. Therefore, as opposed to the previous chapter about informativeness, the added value of doing a requirement annotation first is much less. In this case, leaving out the initial requirement annotating step and doing the mechanism annotation from the heatmaps would have saved annotation costs, while not sacrificing the information with respect to the validity of this model.

- In Figure 6.7, we see that after two rounds we found all but one bias and mispredictions, meaning that we had spent 113 minutes for annotating 578 images that only provided us with 1 additional found bias. This suggests that with the current setup, doing two rounds would be sufficient to achieve our goals.

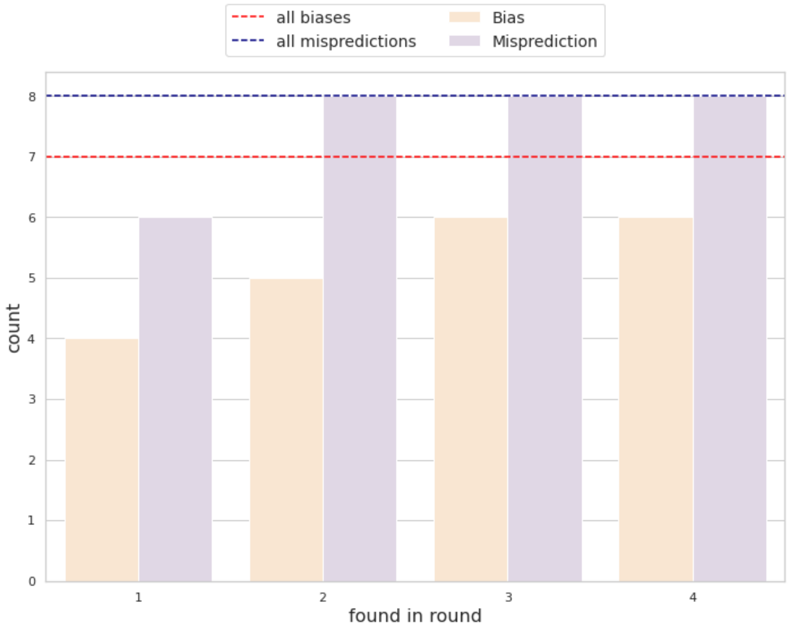



Figure 6.7: Overview of the found biases and mispredictions per round

- During this evaluation, we elicited explanations for biases and mispredictions from Brickroutine. Regarding these, finding the mispredictions was a straightforward action since in the UI we could go to the overview pane and look for that specific combination. However, an extension would be a query function where we could enter the *true label* and *predicted class* and get an overview of all the concepts so that the user gets this information right away. For the biases, we could implement a form of hypothesis testing, in which the user is allowed to link specific heatmaps to hypotheses so that explanations could be stored.

# 7 Discussion

This chapter serves to evaluate the design and evaluation of Brickroutine from the previous chapters. First, we will address design choices that have been made and the implications for the usage of the system. Subsequently, we will discuss lessons learnt from this and how this could impact possible future work. We will conclude this chapter with perceptions about the generalizability of our approach and the practical applications of our work in the current state.

## 7.1. Requirement-first vs Concept-first

During the design of our system, we choose to first let a human-defined requirement be verified in an image and from this presence verify if the model reasoned on mathematical representations of these concepts. We choose to do so because the goal we want to achieve is to test if an ML-model reasons like a human does. This approach can metaphorically be interpreted as what in software testing is referred to as a AAA (Arrange-Act-Assert) pattern. We set up the necessary elements by defining what a requirement is, then we act by letting the annotators work on the heatmaps and eventually we test to assess the similarity between the two. In our opinion, this approach leads to three caveats:

- A requirement that is present in the image is not annotated in the requirement annotation step because a requirement with a higher weight is verified first. Although the requirement is marked as verified if the user selects the exact set of concepts as a custom mechanism, this is at the expense of additional annotation costs and would have been more efficient when the correct requirement was selected in the first place.
- During the evaluation, we noticed that annotators end up adding similar requirements that only differ on some concepts. For instance, in the case of a great white shark, we had one requirement consisting of the concepts: `mouth`, `snout`, `gill openings`, `eye` and `nostril` and another one in which the `nostril` was omitted. For some images, the `nostril` was visually present in the input image but was not sufficiently highlighted. The user has to enter both of these requirements, leading to more annotation time. Currently, in the mechanism annotation step the user can choose from this distinct set that is ordered alphabetically.
- Our current setup to test a requirement imposes a binary constraint: a requirement is either entirely verified or not at all. While there could be a situation in which just $n-1$ of the $n$ concepts in a requirement are verified, this information is not picked up by Brickroutine.

We can label our current approach as a *Requirement-first* approach. Another view to look at this would be a *Concept-first* approach in which we introduce some alterations in the design of our workflow and system as described below:

1. In workflow 1, we present the input images to the user and ask them to name the concepts that appear in the images using an interface equivalent to that of annotating the mechanisms (Figure 3.10). The user should be directed into annotating concepts that are likely to use be used to classify a specific class and at the same time is distinctive between different classes that are part of the data set. Annotated concepts should be reappearing as a checkbox to prevent time overhead as a result of repeatedly typing in the concept.

2. Execute workflow 2 (subsection 3.5.3) as-is to obtain the mechanism from the heatmaps.
3. Requirement correction is not applicable since users can add concepts in the first step and the weight is not used.
4. Incorporate an additional element to let domain experts verify if this is valid reasoning. We can use the knowledge that concepts that have been entered in the first step will contribute to meaningful explanations and concepts that solely exist in the mechanism annotation step as irrelevant or background concepts. The goal of this step is to filter out concepts that might have been learnt by the model and do not contribute to the class itself.
5. In the overview pane, present the validated mechanisms as well as the invalidated ones aggregated by classes.

In summary, with the approach presented above, we move the connection between requirements and concepts to a later stage. This could introduce a bias for the domain experts since they can pick from mechanisms that have been annotated instead of reasoning from their own knowledge. With the approach presented above, we can design views with concept-based statistics, such as the most learnt concepts by the model and co-occurring concepts. Additionally, when the system has obtained a decent knowledge base of the concepts, we can apply inference techniques like Markov Logic Networks (MLN) to automatically extract the most likely requirement [26].

## 7.2. Multiple Requirements Annotation

As stated in the previous section, in this work we limited ourselves to testing a single requirement for each image. A small addition would be to expand our system to have multiple requirements per image. This will inevitably lead to higher annotation costs in the requirement annotation phase but will reduce the costs in the mechanism annotation phase since the users do not have to enter the concepts manually in the case the highlighted concepts in the heatmap are represented by a requirement. There is however a trade-off in the case a custom mechanism is still required when none of the verified requirements is highlighted in the heatmap. We propose this trade-off between more costs in either workflow 1 or workflow as an opportunity for future work.

## 7.3. Differences Between the Experiments

During the evaluation, we tested two different use-cases with different goals. We evaluated the *Informativeness* (data set *Sea Creatures*) for a data set with high accuracy to see how much information about the right predictions the system could give us. Secondly, we evaluated the *Validity* (data set *Birds*) for a data set that had lower accuracy and we had expected the model to reason significantly different from humans due to the injected biases. Our findings as a result of these different evaluations include:

- For *Informativeness*, the percentage of verified requirements is a valid metric since it is an indication of the degree to which the model reasons like a human would. For this data set, an average of 88% of the images featured the requirements and for an average of 57% of the images, the mechanism followed these requirements.

- For Validity, the percentage of verified requirements is not a suitable metric since it does not represent the information we get from the system in a quantitative way. Instead, we found explanations for confusion between different classes and verified hypotheses based on how we trained the model. Since we deem these goals as valid use cases for interpreting the model, an improvement would be to design user interface elements or workflows for these specific scenarios and present the findings to the user. In our current setup, finding these causes heavily relied on the human assessment of the overview pane. We envision a dashboard that allows users to design specific experiments so that they can explicitly search for specific behaviour of their models.

- We found that on average, the mechanism annotation for *Validity* was more costly than for *Informativeness*. In Figure 7.1, these differences are depicted. This can be easily explained by the fact that for *Validity*, the user is resorted to annotating a custom mechanism, which is more time-intensive than simply clicking a yes button to verify that requirement in the heatmap. We also notice that the requirement annotation time is lower for *Validity* than for *Informativeness*. This is

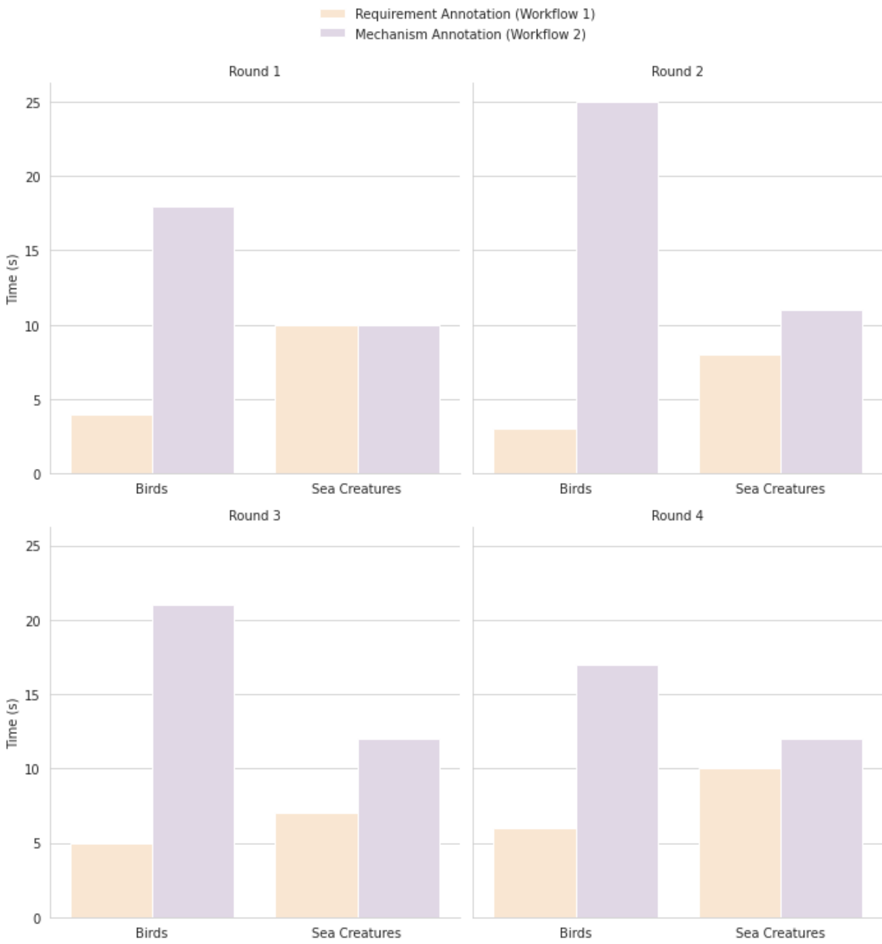



Figure 7.1: Difference per data set in annotation times

explained by the fact that we had initially fewer and coarser-grained requirements for the *Birds* data set that are more likely to feature in the images. As a result, the users spend less time until a suitable requirement is found or until all requirements have been tested.

The observations above lead to the fundamental question if both types of experiments are actually eligible for an approach that differentiates between requirements and mechanisms. For cases like *Informativeness* that have rather an explanatory nature, we think this approach is suitable since this eventual comparison between what we think the model should learn and actually has learnt is appropriate. The nature of *Validity* on the other hand is more exploratory, and we have no intuitive expectations of how the model actually decided. For these cases, limiting ourselves to the mechanism annotation workflow 2 would have yielded similar outcomes.

## 7.4. Iterative Approach

During the design of Brickroutine, we proposed the iterative approach that is depicted inFigure 3.15. The idea behind this is that all semantic concepts are eventually conceptualizations of the models' decision and these conceptualizations can change over time depending on the number of annotations. In the end, we never know what the model truly learnt and all information that is extracted from Brickroutine is only a linguistic representation of numerical decisions.

We found that for our evaluation regarding the *Informativeness* this approach proved to be suitable since in the first two rounds our requirements changed. Examples are the evolution from `front body part` to `carapiece` (finer-grained concept) for *American Lobster* and constructing requirements based on the angle the picture is taken from for *Great White Shark*. For the use-case of *Validity*, on the other hand, we noticed that our initial requirements did not change much. The reason behind this was that emphasis was put on annotating the mechanisms because the granularity of the existing concepts was appropriate to reach our goals.

A simple heuristic would be that the more requirements are changing over time, the more suitable an iterative approach is. However, the overhead of doing more smaller iterations versus fewer larger annotations is marginal. Since in both evaluations we found that the explanations were sufficient after two rounds of annotating with 40% of the images for each data set, we still believe this is feasible for our problem statement. An improvement for Brickroutine would be to suggest a number of images to annotate that is based on the degree to which requirements have changed in previous annotation rounds.

## 7.5. Usability

Since this thesis and as a result, Brickroutine is the first attempt for an initial product, the usability is proportional to the scope of a graduation project. Currently, the system can be executed on any windows or UNIX-like machine that supports docker (≥8GB RAM is advised). The source code is stored on a TU-Delft repository and should be requested in collaboration with owners[1]. Below, we list some of the possibilities and limitations of the current software product.

- Heatmaps are automatically generated but the usage of Inception V3[29] and ImageNet weights is fixed. To change this, the python code in the Docker container that extracts the heatmaps should be altered accordingly.
- Because the system runs in Docker containers, it can be deployed on ubiquitous cloud platforms to make it accessible over the public internet. Ideally, this should be in an authorized environment or authorization should be added to the application itself (React and Dotnet support OAuth2 flows [2]).
- The source code allows users to use a local MongoDB database in a docker container as well as an online database on Microsoft Azure[3]. This can be easily substituted by for example an AWS S3 instance[4]. For multiple simultaneous users, the system should be adapted for concurrent annotations.
- Due to a container-based approach, debugging the individual components is easily done by making use of remote containers[5].
- In section 2.1, we mentioned existing solutions that provide model developers with tools for interpretability of their models. Brickroutine distinguishes itself in a number of ways. First, it should be used after training the model. Additionally, it is a user-friendly tool complete with a user interface and tools for monitoring and the system design is done in such a way that it is extensible.
- For the current functionalities, having RabbitMQ could be seen as slightly over-engineered, since we use it for only one use-case, extracting the heatmaps. In hindsight, this could also be solved with less complex solutions. However, given that this Event-Driven Architecture enables possibilities for follow-up work, we regard this as an appropriate design choice in retrospect.

A list of proposed improvements for the current implementation from a developer's perspective is given in Appendix C.

## 7.6. Future work

In previous sections of this chapter, we already mentioned improvements to the system that fit within the current system boundaries. These can be characterized as *in-depth* improvements and are based on the same level of input (input images and a trained model) and output (semantic explanations of the model). Since we aim for Brickroutine to be a novel cornerstone for Machine-Learning interpretability in a broader sense, we envision some *in-breadth* improvements that are eligible for future work. These are conceptually described below:

[1] https://github.com/delftcrowd/brickroutine
[2] https://oauth.net/2/
[3] https://docs.microsoft.com/en-us/azure/cosmos-db/mongodb
[4] https://aws.amazon.com/s3/
[5] https://code.visualstudio.com/docs/remote/containers

- Currently, the process halts when explanations of the model have been obtained. In a real use case, we believe this information is consumed by the models' developers and used to optimize the model. Therefore, we envision Brickroutine to be a full-fledged ML suite in which developers can adapt hyper-parameters, retrain their models and gather explanations for different versions of their models. Due to the event-driven architecture, we can upload and download models by for example submitting a python file or trained model in the same way we currently upload images. We can communicate the status of these training sessions back from Docker containers to the web API. The current user interface could be expanded with for instance web sockets to enable real-time monitoring of these training sessions.
- In our current setup we upload a trained model to subsequently annotate concepts in them. Eventually, we have knowledge of what concepts are important for the model. We think of a scenario in which we let domain experts annotate specific regions where they think differences between different classes will specifically be visually present. Let us take two similar-looking birds as an example: if the domain expert can by means of a bounding box indicate in which parts of different birds the differences primarily occur, then the model is guided into looking for specific patterns. Then we would shift the added value that the domain experts have from *post-training* to *pre-training*. After doing predictions, the current setup of Brickroutine can be leveraged to determine the added value of this procedure.
- As an extension to the previous point, we can adjust the system in such a way that concepts are given a semantic label like is done in [1]. In this way, annotations can be partially automated and we would gradually decrease the annotation costs when there are enough samples.
- In our current work, every image is annotated by strictly one annotator. For more performance add in the possibility to let more annotators use the system to have a more diverse base of human knowledge.
- To broaden the horizon, we envision Brickroutine could be expanded to be applicable to other problems than image classification problems and a deep learning model. In the end, we add semantic meaning to a conceptualization of numerical features. If a problem is suitable for human-in-the-loop machine learning, then an approach like Brickroutine could be eligible to assess if the model follows some kind of human reasoning. Use cases we imagine are genre-classification of music or language-based models. We would like to leave this to the imagination of the reader.

# 8 Conclusion

In this work, we have presented the design and implementation of Brickroutine: a system that uses a trained model, input images and a human-in-the-loop approach to give semantic interpretations to image classification problems. By giving an iterative approach in terms of workflows and technically designing it in a modular, salable way using Docker, we hope to have inspired researchers and software developers to keep developing cutting edge solutions for interpretability and combine this in a ready-to-use expandable system with modern user interfaces.

We have shown that the current setup allows users to construct requirements for an image classification problem and test this against the mechanisms an AI model uses for making predictions. We have shown the differences between well-performing and less performing combinations of models and data sets. We did two types of evaluation on data sets that featured 300 and 500 images and found that doing two iterations of annotations for 40% of the images in each data set was sufficient to explain the model to a certain extent. Lastly, we found that the combination of model performance and visual differences between images are fundamental in designing workflows to serve specific goals. For interpretations of an explanatory nature, which we call *informativeness*, we think our current approach is suitable. For exploratory interpretations, which we refer to as *validity*, we believe the current setup is inappropriate. In conclusion, we hope to have given insight into an iterative approach for the interpretation of the inner workings of ML models. We believe that by contributing to this work, we have laid a new cornerstone that serves as a foundation for follow-up work. As a result, we have made suggestions for follow-up work. We hope to have contributed to new inspirations and insights into the area of interpretability, for image recognition problems, and beyond.

# A

# Results for sea creatures

## A.1. Results for sea creatures round 1

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **American lobster (8)** | 18 (18%) | 9 (9%) | 9 (9%) | 2 (2%) | 80 (80%) |
| **great white shark (6)** | 19 (19%) | 8 (8%) | 11 (11%) | 1 (1%) | 80 (80%) |
| **tench (8)** | 18 (18%) | 16 (16%) | 2 (2%) | 2 (2%) | 80 (80%) |

Table A.1: Requirements overview table after one round for *Sea Creatures*

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| American lobster | American lobster | claw, red shell | yes | 5 |
| | | red shell | yes | 4 |
| | | claw, front body part | no | 2 |
| | | head, eye | no | 1 |
| | | front body part | no | 1 |
| | | disassembled claw, human mouth | no | 1 |
| | | front body part, claw | no | 1 |
| | | claw, head | no | 1 |
| | | head, front body part | no | 1 |
| | | claw, front body part, head | no | 1 |
| great white shark | great white shark | mouth, snout, gill openings | yes | 2 |
| | | mouth, dorsal fin, snout | no | 2 |
| | | dorsal fin, pectoral fin | yes | 2 |
| | | mouth, snout | yes | 2 |
| | | mouth, snout, eye | no | 1 |
| | | mouth, eye, snout | no | 1 |
| | | dorsal fin, pectoral fin, caudal fin | yes | 1 |
| | | eye, nostrill | no | 1 |
| | | mouth, snout, pectoral fin | no | 1 |
| | | eye, gill openings, dorsal fin, mouth | no | 1 |
| | | gill openings, dorsal fin | no | 1 |
| | | gill openings | yes | 1 |
| | | eye, mouth, snout, fishing equipment | no | 1 |
| | | eye, nostrill, mouth | no | 1 |
| tench | tench | olive skin, dark rounded fins | yes | 7 |
| | | olive skin, dark rounded fins, red eye | yes | 5 |
| | | olive skin | yes | 3 |
| | | olive skin, red eye | yes | 1 |
| | | camouflage clothing, olive skin, dark rounded fins, red eye | no | 1 |
| | | camouflage clothing, dark rounded fins | no | 1 |

Table A.2: Mechanisms overview table after one round for *Sea Creatures*

## A.2. Results for sea creatures round 2

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **American lobster (8)** | 36 (36%) | 28 (28%) | 8 (8%) | 4 (4%) | 60 (60%) |
| **great white shark (7)** | 32 (32%) | 17 (17%) | 15 (15%) | 8 (8%) | 60 (60%) |
| **tench (8)** | 37 (37%) | 25 (25%) | 12 (12%) | 3 (3%) | 60 (60%) |

Table A.3: Requirements overview table after two rounds for *Sea Creatures*

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| American lobster | American lobster | claw, carapiece | yes | 17 |
| | | carapiece | yes | 8 |
| | | claw, front body part | no | 2 |
| | | claw | yes | 2 |
| | | thin legs | yes | 1 |
| | | front body part, claw | no | 1 |
| | | claw, head | no | 1 |
| | | bottom body, claw | no | 1 |
| | | claw, front body part, head | no | 1 |
| | | eye, thin legs | no | 1 |
| | | claw, eye | no | 1 |
| great white shark | great white shark | mouth, snout, gill openings, eye, nostrill | yes | 6 |
| | | mouth, snout, eye, nostrill | yes | 6 |
| | | dorsal fin, pectoral fin | yes | 3 |
| | | dorsal fin, pectoral fin, caudal fin | yes | 2 |
| | | mouth, gill openings, pectoral fin, caudal fin, dorsal fin, snout, eye | no | 1 |
| | | dorsal fin, eye, mouth, snout, pectoral fin | no | 1 |
| | | eye, nostrill | no | 1 |
| | | gill openings, dorsal fin | no | 1 |
| | | gill openings | no | 1 |
| | | mouth, eye, snout | no | 1 |
| | | dorsal fin, gill openings | no | 1 |
| | | dorsal fin, pectoral fin, gill openings | no | 1 |
| | | eye, mouth, fishing equipment | no | 1 |
| | | fishing equipment, dorsal fin, caudal fin | no | 1 |
| | | dorsal fin, pectoral fin, eye | no | 1 |
| | | pectoral fin, dorsal fin, mouth, eye, snout, nostrill | no | 1 |
| | | snout, mouth, fishing equipment, pectoral fin | no | 1 |
| | | mouth, snout, nostrill | no | 1 |
| | | dorsal fin, fishing equipment | no | 1 |
| tench | tench | olive skin, dark rounded fins, red eye | yes | 9 |
| | | olive skin, dark rounded fins | yes | 9 |
| | | olive skin | yes | 4 |
| | | olive skin, red eye | yes | 3 |
| | | camouflage clothing, olive skin, dark rounded fins, red eye | no | 1 |
| | | camouflage clothing, dark rounded fins, olive skin | no | 1 |
| | | red eye, fin attachment | no | 1 |
| | | olive skin, red eye, human, dark rounded fins | no | 1 |
| | | gill cover, dark rounded fins, olive skin | no | 1 |
| | | olive skin, human | no | 1 |
| | | gill cover, red eye, olive skin | no | 1 |
| | | olive skin, dark rounded fins, human | no | 1 |
| | | dog, olive skin, dark rounded fins | no | 1 |
| | | human, olive skin | no | 1 |
| | | human, olive skin, red eye | no | 1 |
| | | dark rounded fins, gill cover, lips, red eye, olive skin | no | 1 |

Table A.4: Mechanisms overview table after two rounds for *Sea Creatures*

# A.3. Results for sea creatures round 3

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **American lobster (10)** | 56 (56%) | 44 (44%) | 12 (12%) | 4 (4%) | 40 (40%) |
| **great white shark (7)** | 46 (46%) | 22 (22%) | 24 (24%) | 14 (14%) | 40 (40%) |
| **tench (8)** | 55 (55%) | 38 (38%) | 17 (17%) | 5 (5%) | 40 (40%) |

Table A.5: Requirements overview table after three rounds for *Sea Creatures*

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| American lobster | American lobster | claw, carapiece | yes | 18 |
| | | claw texture, carapiece | yes | 12 |
| | | carapiece | yes | 8 |
| | | claw, front body part | no | 2 |
| | | thin legs | yes | 2 |
| | | claw | yes | 2 |
| | | claw texture | yes | 2 |
| | | front body part, claw | no | 1 |
| | | claw, head | no | 1 |
| | | bottom body, claw | no | 1 |
| | | claw, front body part, head | no | 1 |
| | | eye, thin legs | no | 1 |
| | | claw, eye | no | 1 |
| | | tail fin, carapiece, claw | no | 1 |
| | | human, eye, front body part, thin legs | no | 1 |
| | | human, claw, carapiece | no | 1 |
| | | carapiece, human | no | 1 |
| great white shark | great white shark | mouth, snout, gill openings, eye, nostrill | yes | 9 |
| | | mouth, snout, eye, nostrill | yes | 7 |
| | | dorsal fin, pectoral fin | yes | 3 |
| | | dorsal fin, pectoral fin, caudal fin | yes | 2 |
| | | gill openings | no | 2 |
| | | dorsal fin, pectoral fin, eye | no | 2 |
| | | mouth, gill openings, pectoral fin, caudal fin, dorsal fin, snout, eye | no | 1 |
| | | dorsal fin, eye, mouth, snout, pectoral fin | no | 1 |
| | | eye, nostrill | no | 1 |
| | | gill openings, dorsal fin | no | 1 |
| | | mouth, eye, snout | no | 1 |
| | | dorsal fin, gill openings | no | 1 |
| | | dorsal fin, pectoral fin, gill openings | no | 1 |
| | | eye, mouth, fishing equipment | no | 1 |
| | | fishing equipment, dorsal fin, caudal fin | no | 1 |
| | | pectoral fin, dorsal fin, mouth, eye, snout, nostrill | no | 1 |
| | | snout, mouth, fishing equipment, pectoral fin | no | 1 |
| | | mouth, snout, nostrill | no | 1 |
| | | dorsal fin, fishing equipment | no | 1 |
| | | mouth, snout, nostrill, gill openings | no | 1 |
| | | snout, eye, nostrill | no | 1 |
| | | mouth, snout, gill openings, dorsal fin, pectoral fin, caudal fin, eye, nostrill | yes | 1 |
| | | dorsal fin, eye, mouth, caudal fin | no | 1 |
| | | eye, dorsal fin, snout | no | 1 |
| | | eye, mouth, dorsal fin | no | 1 |
| | | fishing equipment, mouth, eye, snout | no | 1 |
| | | eye, snout | no | 1 |
| tench | tench | olive skin, dark rounded fins, red eye | yes | 16 |
| | | olive skin, dark rounded fins | yes | 11 |
| | | olive skin, red eye | yes | 5 |
| | | olive skin | yes | 5 |
| | | camouflage clothing, olive skin, dark rounded fins, red eye | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | camouflage clothing, dark rounded fins, olive skin | no | 1 |
| | | red eye, fin attachment | no | 1 |
| | | olive skin, red eye, human, dark rounded fins | no | 1 |
| | | gill cover, dark rounded fins, olive skin | no | 1 |
| | | olive skin, human | no | 1 |
| | | gill cover, red eye, olive skin | no | 1 |
| | | olive skin, dark rounded fins, human | no | 1 |
| | | dog, olive skin, dark rounded fins | no | 1 |
| | | human, olive skin | no | 1 |
| | | human, olive skin, red eye | no | 1 |
| | | dark rounded fins, gill cover, lips, red eye, olive skin | no | 1 |
| | | dark rounded fins, red eye | yes | 1 |
| | | human, dark rounded fins, silver skin | no | 1 |
| | | gill cover, silver skin | no | 1 |
| | | human, olive skin, dark rounded fins | no | 1 |
| | | silver skin, dark rounded fins | no | 1 |
| | | human, jaw texture | no | 1 |

Table A.6: Mechanisms overview table after three rounds for *Sea Creatures*

## A.4. Results for sea creatures round 4

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **American lobster (10)** | 96 (96%) | 71 (71%) | 25 (25%) | 4 (4%) | 0 |
| **great white shark (7)** | 80 (80%) | 41 (41%) | 39 (39%) | 20 (20%) | 0 |
| **tench (8)** | 89 (89%) | 60 (60%) | 29 (29%) | 11 (11%) | 0 |

Table A.7: Requirements overview table after four rounds for *Sea Creatures*

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| American_lobster | American_lobster | claw, carapiece | yes | 18 |
| | | claw texture, carapiece | yes | 12 |
| | | carapiece | yes | 8 |
| | | claw, front body part | no | 2 |
| | | thin legs | yes | 2 |
| | | claw | yes | 2 |
| | | claw texture | yes | 2 |
| | | front body part, claw | no | 1 |
| | | claw, head | no | 1 |
| | | bottom body, claw | no | 1 |
| | | claw, front body part, head | no | 1 |
| | | eye, thin legs | no | 1 |
| | | claw, eye | no | 1 |
| | | tail fin, carapiece, claw | no | 1 |
| | | human, eye, front body part, thin legs | no | 1 |
| | | human, claw, carapiece | no | 1 |
| | | carapiece, human | no | 1 |
| great_white_shark | great_white_shark | mouth, snout, gill openings, eye, nostrill | yes | 9 |
| | | mouth, snout, eye, nostrill | yes | 7 |
| | | dorsal fin, pectoral fin | yes | 3 |
| | | dorsal fin, pectoral fin, caudal fin | yes | 2 |
| | | gill openings | no | 2 |
| | | dorsal fin, pectoral fin, eye | no | 2 |
| | | mouth, gill openings, pectoral fin, caudal fin, dorsal fin, snout, eye | no | 1 |
| | | dorsal fin, eye, mouth, snout, pectoral fin | no | 1 |
| | | eye, nostrill | no | 1 |
| | | gill openings, dorsal fin | no | 1 |
| | | mouth, eye, snout | no | 1 |
| | | dorsal fin, gill openings | no | 1 |
| | | dorsal fin, pectoral fin, gill openings | no | 1 |
| | | eye, mouth, fishing equipment | no | 1 |
| | | fishing equipment, dorsal fin, caudal fin | no | 1 |
| | | pectoral fin, dorsal fin, mouth, eye, snout, nostrill | no | 1 |
| | | snout, mouth, fishing equipment, pectoral fin | no | 1 |
| | | mouth, snout, nostrill | no | 1 |
| | | dorsal fin, fishing equipment | no | 1 |
| | | mouth, snout, nostrill, gill openings | no | 1 |
| | | snout, eye, nostrill | no | 1 |
| | | mouth, snout, gill openings, dorsal fin, pectoral fin, caudal fin, eye, nostrill | yes | 1 |
| | | dorsal fin, eye, mouth, caudal fin | no | 1 |
| | | eye, dorsal fin, snout | no | 1 |
| | | eye, mouth, dorsal fin | no | 1 |
| | | fishing equipment, mouth, eye, snout | no | 1 |
| | | eye, snout | no | 1 |
| tench | tench | olive skin, dark rounded fins, red eye | yes | 16 |
| | | olive skin, dark rounded fins | yes | 11 |
| | | olive skin, red eye | yes | 5 |
| | | olive skin | yes | 5 |
| | | camouflage clothing, olive skin, dark rounded fins, red eye | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | camouflage clothing, dark rounded fins, olive skin | no | 1 |
| | | red eye, fin attachment | no | 1 |
| | | olive skin, red eye, human, dark rounded fins | no | 1 |
| | | gill cover, dark rounded fins, olive skin | no | 1 |
| | | olive skin, human | no | 1 |
| | | gill cover, red eye, olive skin | no | 1 |
| | | olive skin, dark rounded fins, human | no | 1 |
| | | dog, olive skin, dark rounded fins | no | 1 |
| | | human, olive skin | no | 1 |
| | | human, olive skin, red eye | no | 1 |
| | | dark rounded fins, gill cover, lips, red eye, olive skin | no | 1 |
| | | dark rounded fins, red eye | yes | 1 |
| | | human, dark rounded fins, silver skin | no | 1 |
| | | gill cover, silver skin | no | 1 |
| | | human, olive skin, dark rounded fins | no | 1 |
| | | silver skin, dark rounded fins | no | 1 |
| | | human, jaw texture | no | 1 |

Table A.8: Mechanisms overview table after four rounds for *Sea Creatures*

# B

# Results for Birds

## B.1. Results for birds round 1

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **american goldfinch (3)** | 10 (20%) | 3 (6%) | 7 (14%) | 0 | 39 (80%) |
| **bufflehead (4)** | 10 (20%) | 0 | 10 (20%) | 0 | 39 (80%) |
| **downy woodpecker (4)** | 10 (20%) | 0 | 10 (20%) | 0 | 40 (80%) |
| **gila woodpecker (4)** | 10 (20%) | 0 | 10 (20%) | 0 | 40 (80%) |
| **hairy woodpecker (5)** | 10 (20%) | 0 | 10 (20%) | 0 | 40 (80%) |
| **hooded merganser (3)** | 10 (20%) | 0 | 10 (20%) | 0 | 40 (80%) |
| **lesser goldfinch (3)** | 9 (18%) | 0 | 9 (18%) | 1 (2%) | 39 (80%) |
| **mandarin duck (5)** | 10 (20%) | 0 | 10 (20%) | 0 | 40 (80%) |
| **monk parakeet (3)** | 10 (20%) | 3 (6%) | 7 (14%) | 0 | 39 (80%) |
| **pine grosbeak (3)** | 10 (21%) | 2 (4%) | 8 (17%) | 0 | 38 (79%) |

Table B.1: Requirements overview table after one rounds for *birds*

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| american goldfinch | american goldfinch | yellow breast, yellow belly, yellow back, black wings | yes | 2 |
| | | yellow breast, yellow belly, yellow back | yes | 1 |
| | | white bottom, tree | no | 1 |
| | | yellow back, tree, yellow crown | no | 1 |
| | | tree, yellow crown, yellow back | no | 1 |
| | | tree, yellow crown | no | 1 |
| | | yellow crown, tree, white bottom | no | 1 |
| | | yellow crown, green background, wing patch | no | 1 |
| | | foot, tree, green background, yellow belly, black crown | no | 1 |
| bufflehead | bufflehead | water | no | 2 |
| | hooded merganser | water, eye, neck | no | 1 |
| | | water, grey feathers, rainbow crest, white spot | no | 1 |
| | bufflehead | water, white feathers, top wing | no | 1 |
| | | dark head, white spot, top wing, beak | no | 1 |
| | | beak, white feathers, water | no | 1 |
| | hooded merganser | water, white spot, grey feathers, neck | no | 1 |
| | bufflehead | water, grey feathers | no | 1 |
| | | top wing, water | no | 1 |
| downy woodpecker | downy woodpecker | black white wing patches, tree | no | 2 |
| | | tree | no | 1 |
| | | black white wing patches, sky | no | 1 |
| | | black white wing patches, white breast, tree | no | 1 |
| | | tree, white breast, black white wing patches | no | 1 |
| | | white breast, tree | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | tree, black white wing patches | no | 1 |
| | lesser goldfinch | tree, black white wing patches, white breast | no | 1 |
| | downy woodpecker | sky, black white wing patches | no | 1 |
| gila woodpecker | gila woodpecker | sky, cactus | no | 2 |
| | | green belly, sky | no | 1 |
| | | green neck, red crown | no | 1 |
| | monk parakeet | green belly, black white striped wings | no | 1 |
| | downy woodpecker | black white striped wings, sky, green crown | no | 1 |
| | | green crown, black white striped wings, tree | no | 1 |
| | | green neck, red crown, black white striped wings, tree | no | 1 |
| | american goldfinch | sky, black white striped wings, green crown | no | 1 |
| | downy woodpecker | tree, green belly | no | 1 |
| hairy woodpecker | hairy woodpecker | sky | no | 1 |
| | | black wings, sky, throat stripe, beak | no | 1 |
| | monk parakeet | tree, black wings | no | 1 |
| | downy woodpecker | red crown, white breast, black wings | no | 1 |
| | hairy woodpecker | red crown, black and white wing patches, sky, tree | no | 1 |
| | | white breast, tree | no | 1 |
| | | throat stripe, white breast, white belly | no | 1 |
| | | white belly, tree, black and white wing patches | no | 1 |
| | | throat stripe, sky, white back stripe | no | 1 |
| | downy woodpecker | throat stripe, sky | no | 1 |
| hooded merganser | hooded merganser | dark feather texture, eye | no | 1 |
| | | brown sides, eye | no | 1 |
| | | black crest with white spot, eye, neck | no | 1 |
| | | eye, brown sides, water | no | 1 |
| | | eye, neck, water, brown sides | no | 1 |
| | | neck, water | no | 1 |
| | | cinnamon crest, water, dark feather texture | no | 1 |
| | | eye, dark feather texture | no | 1 |
| | | brown sides, eye, water | no | 1 |
| | | dark feather texture, water | no | 1 |
| lesser goldfinch | lesser goldfinch | yellow belly, tree | no | 1 |
| | | beak, sky, yellow breast | no | 1 |
| | | beak, sky | no | 1 |
| | | black wings, lighter wing patches, beak, green background | no | 1 |
| | american goldfinch | black wings, yellow belly, sky | no | 1 |
| | lesser goldfinch | yellow back, yellow belly, lighter wing patches, green background | no | 1 |
| | | green background, lighter wing patches | no | 1 |
| | | lighter wing patches, tree | no | 1 |
| | | black crown, yellow belly, neck accent | no | 1 |
| mandarin duck | mandarin duck | rainbow crest, golden sides, brown feathers, water | no | 1 |
| | | brown feathers, golden sides, water | no | 1 |
| | lesser goldfinch | golden sides, water, soil, background ornament | no | 1 |
| | mandarin duck | rainbow crest, brown feathers, neck | no | 1 |
| | american goldfinch | golden sides, brown feathers, neck, water | no | 1 |
| | mandarin duck | neck, golden sides, water, red beak | no | 1 |
| | | rainbow crest, neck, red beak, water | no | 1 |
| | hooded merganser | neck, water, white stripe below eye | no | 1 |
| | mandarin duck | rainbow crest, soil, brown feathers | no | 1 |
| | | red beak, golden sides, water | no | 1 |
| monk parakeet | monk parakeet | green feathers, light breast, light crown | yes | 2 |
| | | green feathers, light crown, sky, tree | no | 1 |
| | | green feathers | yes | 1 |
| | | fence, light crown, sky, green feathers | no | 1 |
| | | blue wingtips, beak, green feathers, tree | no | 1 |
| | | green feathers, tree, sky | no | 1 |
| | | sky, fence, beak | no | 1 |
| | | sky, tree, green feathers, light breast | no | 1 |
| | | light breast, tree | no | 1 |
| pine grosbeak | pine grosbeak | pink feathers, grey wings | yes | 2 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | pink feathers, snow, cheek | no | 1 |
| | american goldfinch | orange head, tree, green background | no | 1 |
| | | grey feathers, orange head, green background | no | 1 |
| | pine grosbeak | grey feathers, orange head, sky | no | 1 |
| | | grey feathers, orange head, tree | no | 1 |
| | monk parakeet | sky, snow, tree, wing patches | no | 1 |
| | pine grosbeak | wing patches, snow, grey feathers | no | 1 |
| | | wing patches, pink feathers, sky | no | 1 |

Table B.2: Mechanisms overview table after one rounds for *birds*

## B.2. Results for birds round 2

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **american goldfinch (3)** | 20 (41%) | 4 (8%) | 16 (33%) | 0 | 29 (59%) |
| **bufflehead (4)** | 20 (41%) | 0 | 20 (41%) | 0 | 29 (59%) |
| **downy woodpecker (4)** | 20 (40%) | 0 | 20 (40%) | 0 | 30 (60%) |
| **gila woodpecker (4)** | 20 (40%) | 0 | 20 (40%) | 0 | 30 (60%) |
| **hairy woodpecker (5)** | 20 (40%) | 2 (4%) | 18 (36%) | 0 | 30 (60%) |
| **hooded merganser (3)** | 20 (40%) | 0 | 20 (40%) | 0 | 30 (60%) |
| **lesser goldfinch (4)** | 19 (39%) | 1 (2%) | 18 (37%) | 1 (2%) | 29 (59%) |
| **mandarin duck (6)** | 20 (40%) | 0 | 20 (40%) | 0 | 30 (60%) |
| **monk parakeet (3)** | 19 (39%) | 3 (6%) | 16 (33%) | 1 (2%) | 29 (59%) |
| **pine grosbeak (3)** | 20 (42%) | 3 (6%) | 17 (35%) | 0 | 28 (58%) |

Table B.3: Requirements overview table after two rounds for *birds*

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| american goldfinch | american goldfinch | yellow breast, yellow belly, yellow back | yes | 2 |
| | | yellow breast, yellow belly, yellow back, black wings | yes | 2 |
| | | white bottom, tree | no | 1 |
| | | yellow back, tree, yellow crown | no | 1 |
| | | tree, yellow crown, yellow back | no | 1 |
| | | tree, yellow crown | no | 1 |
| | | yellow crown, tree, white bottom | no | 1 |
| | | yellow crown, green background, wing patch | no | 1 |
| | | foot, tree, green background, yellow belly, black crown | no | 1 |
| | | yellow belly, yellow breast, black crown | no | 1 |
| | | green background, black crown | no | 1 |
| | | yellow belly, yellow breast, foot | no | 1 |
| | | green background, beak, eye | no | 1 |
| | | snow, black wings, yellow belly, yellow breast, yellow crown | no | 1 |
| | | green background, yellow back, black wings, green belly, green breast | no | 1 |
| | | tree, sky | no | 1 |
| | | green background, wing patch, tree | no | 1 |
| | | green background, yellow breast, green belly, yellow crown | no | 1 |
| bufflehead | bufflehead | water | no | 2 |
| | hooded merganser | water, eye, neck | no | 1 |
| | | water, grey feathers, rainbow crest, white spot | no | 1 |
| | bufflehead | water, white feathers, top wing | no | 1 |
| | | dark head, white spot, top wing, beak | no | 1 |
| | | beak, white feathers, water | no | 1 |
| | hooded merganser | water, white spot, grey feathers, neck | no | 1 |
| | bufflehead | water, grey feathers | no | 1 |
| | | top wing, water | no | 1 |
| | | water, black wings | no | 1 |
| | | rainbow crest, water | no | 1 |
| | | tree | no | 1 |
| | hooded merganser | water, neck | no | 1 |
| | bufflehead | eye, water, white feathers, black wings | no | 1 |
| | hooded merganser | dark head, white spot, black wings, grey feathers, neck, water | no | 1 |
| | bufflehead | neck, wing patch | no | 1 |
| | hooded merganser | dark head, grey feathers, wing patch, neck | no | 1 |
| | | neck, water | no | 1 |
| | bufflehead | wing patch, neck, grey feathers | no | 1 |
| downy woodpecker | downy woodpecker | tree | no | 4 |
| | | black white wing patches, tree | no | 3 |
| | | black white wing patches, sky | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | black white wing patches, white breast, tree | no | 1 |
| | | tree, white breast, black white wing patches | no | 1 |
| | | white breast, tree | no | 1 |
| | | tree, black white wing patches | no | 1 |
| | lesser goldfinch | tree, black white wing patches, white breast | no | 1 |
| | downy woodpecker | sky, black white wing patches | no | 1 |
| | | tree, black white wing patches, white bottom | no | 1 |
| | hairy woodpecker | tree, neck | no | 1 |
| | downy woodpecker | white breast, white bottom, tree | no | 1 |
| | | foot, tree | no | 1 |
| | hairy woodpecker | white breast, black white wing patches, sky, tree | no | 1 |
| | downy woodpecker | red crown, black white wing patches, tree | no | 1 |
| gila woodpecker | gila woodpecker | sky, cactus | no | 2 |
| | | green belly, sky | no | 1 |
| | | green neck, red crown | no | 1 |
| | monk parakeet | green belly, black white striped wings | no | 1 |
| | downy woodpecker | black white striped wings, sky, green crown | no | 1 |
| | | green crown, black white striped wings, tree | no | 1 |
| | | green neck, red crown, black white striped wings, tree | no | 1 |
| | american goldfinch | sky, black white striped wings, green crown | no | 1 |
| | downy woodpecker | tree, green belly | no | 1 |
| | lesser goldfinch | green background | no | 1 |
| | gila woodpecker | red crown, green breast, green background | no | 1 |
| | downy woodpecker | red crown, black white striped wings, green background, tree, beak | no | 1 |
| | | tree, beak, black white striped wings, green belly, green breast, green neck | no | 1 |
| | | sky, tree, red crown, green neck | no | 1 |
| | monk parakeet | green neck, black white striped wings, green belly, tree, sky | no | 1 |
| | pine grosbeak | water feeder, red crown | no | 1 |
| | gila woodpecker | beak, green belly, green breast, green neck, sky | no | 1 |
| | | green neck, black white striped wings, cactus, sky | no | 1 |
| | pine grosbeak | red crown, green neck, water feeder, green background | no | 1 |
| hairy woodpecker | hairy woodpecker | black and white wing patches, tree | no | 2 |
| | | sky | no | 1 |
| | | black wings, sky, throat stripe, beak | no | 1 |
| | monk parakeet | tree, black wings | no | 1 |
| | downy woodpecker | red crown, white breast, black wings | no | 1 |
| | hairy woodpecker | red crown, black and white wing patches, sky, tree | no | 1 |
| | | white breast, tree | no | 1 |
| | hairy woodpecker | throat stripe, white breast, white belly | no | 1 |
| | | white belly, tree, black and white wing patches | no | 1 |
| | | throat stripe, sky, white back stripe | no | 1 |
| | downy woodpecker | throat stripe, sky | no | 1 |
| | hairy woodpecker | beak, tree, eye | no | 1 |
| | bufflehead | white breast | yes | 1 |
| | hairy woodpecker | tree | no | 1 |
| | | beak, red crown, white back stripe, black wings, green background | no | 1 |
| | downy woodpecker | eye, green background, black and white wing patches | no | 1 |
| | | red crown, black and white wing patches, sky, tree | no | 1 |
| | hairy woodpecker | green background, tree, black and white wing patches | no | 1 |
| | | white breast | yes | 1 |
| hooded merganser | hooded merganser | dark feather texture, eye | no | 1 |
| | | brown sides, eye | no | 1 |
| | | black crest with white spot, eye, neck | no | 1 |
| | | eye, brown sides, water | no | 1 |
| | | eye, neck, water, brown sides | no | 1 |
| | | neck, water | no | 1 |
| | | cinnamon crest, water, dark feather texture | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | eye, dark feather texture | no | 1 |
| | | brown sides, eye, water | no | 1 |
| | | dark feather texture, water | no | 1 |
| hooded merganser | monk parakeet | brown sides, eye | no | 1 |
| | bufflehead | cinnamon crest, water, grey belly, dark feather texture | no | 1 |
| | lesser goldfinch | cinnamon crest, eye, dark feather texture, reed | no | 1 |
| | hooded merganser | cinnamon crest, water, beak, neck | no | 1 |
| | | black crest with white spot, eye, water | no | 1 |
| | | brown sides, water | no | 1 |
| | | cinnamon crest, water, dark feather texture, brown sides | no | 1 |
| | american goldfinch | water, cinnamon crest, dark feather texture | no | 1 |
| | hooded merganser | neck, brown sides | no | 1 |
| | | water, dark feather texture, black crest with white spot | no | 1 |
| lesser goldfinch | lesser goldfinch | green background, lighter wing patches | no | 2 |
| | | yellow belly, tree | no | 1 |
| | | flower, neck | no | 1 |
| | | beak, sky, yellow breast | no | 1 |
| | | beak, sky | no | 1 |
| | | black wings, lighter wing patches, beak, green background | no | 1 |
| | american goldfinch | black wings, yellow belly, sky | no | 1 |
| | lesser goldfinch | yellow back, yellow belly, lighter wing patches, green background | no | 1 |
| | | lighter wing patches, tree | no | 1 |
| | | black crown, yellow belly, neck accent | no | 1 |
| | | yellow belly, yellow breast, black crown | no | 1 |
| | | green background | no | 1 |
| | | green background, yellow back, neck, tree | no | 1 |
| | | black crown, yellow breast, yellow belly, yellow back | yes | 1 |
| | | green background, yellow breast, yellow belly | no | 1 |
| | | yellow belly, yellow breast, tree, beak | no | 1 |
| | | yellow belly, yellow breast, green background, neck | no | 1 |
| | american goldfinch | yellow belly, tree, sky | no | 1 |
| mandarin duck | mandarin duck | rainbow crest, golden sides, brown feathers, water | no | 1 |
| | | brown feathers, golden sides, water | no | 1 |
| | lesser goldfinch | golden sides, water, soil, background ornament | no | 1 |
| | mandarin duck | rainbow crest, brown feathers, neck | no | 1 |
| | american goldfinch | golden sides, brown feathers, neck, water | no | 1 |
| | mandarin duck | neck, golden sides, water, red beak | no | 1 |
| | | rainbow crest, neck, red beak, water | no | 1 |
| | hooded merganser | neck, water, white stripe below eye | no | 1 |
| | mandarin duck | rainbow crest, soil, brown feathers | no | 1 |
| | | red beak, golden sides, water | no | 1 |
| | | rainbow crest, golden sides, brown feathers, water, soil | no | 1 |
| | | rainbow crest, neck, dotted bottom | no | 1 |
| | lesser goldfinch | grey feathers, soil, dotted bottom | no | 1 |
| | mandarin duck | golden sides, rainbow crest, water | no | 1 |
| | | golden sides, brown feathers, long brown neck feathers | no | 1 |
| | | rainbow crest, dotted bottom, brown feathers, water, golden sides, long brown neck feathers | no | 1 |
| | | rainbow crest, long brown neck feathers, neck | no | 1 |
| | gila woodpecker | grey feathers, brown feathers, snow | no | 1 |
| | mandarin duck | rainbow crest, neck, golden sides, water | no | 1 |
| | pine grosbeak | long brown neck feathers, red beak, water, neck | no | 1 |
| monk parakeet | monk parakeet | green feathers, light breast, light crown | yes | 2 |
| | | green feathers, light crown, sky, tree | no | 1 |
| | | green feathers | yes | 1 |
| | | fence, light crown, sky, green feathers | no | 1 |
| | | blue wingtips, beak, green feathers, tree | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | green feathers, tree, sky | no | 1 |
| | | sky, fence, beak | no | 1 |
| | | sky, tree, green feathers, light breast | no | 1 |
| | | light breast, tree | no | 1 |
| | | green feathers, tree | no | 1 |
| | | sand | no | 1 |
| | | sky, tree | no | 1 |
| | | sky, green feathers, light crown | no | 1 |
| | | beak, light throat, light crown, sky | no | 1 |
| | | light crown, light throat, sky, green feathers | no | 1 |
| | american goldfinch | light breast, green feathers, tree | no | 1 |
| | monk parakeet | light throat, urban objects | no | 1 |
| | | blue wingtips, green background | no | 1 |
| pine grosbeak | pine grosbeak | pink feathers, grey wings | yes | 3 |
| | | pink feathers, snow, cheek | no | 1 |
| | american goldfinch | orange head, tree, green background | no | 1 |
| | | grey feathers, orange head, green background | no | 1 |
| | pine grosbeak | grey feathers, orange head, sky | no | 1 |
| | | grey feathers, orange head, tree | no | 1 |
| | monk parakeet | sky, snow, tree, wing patches | no | 1 |
| | pine grosbeak | wing patches, snow, grey feathers | no | 1 |
| | | wing patches, pink feathers, sky | no | 1 |
| | | grey feathers, heavy chest | no | 1 |
| | monk parakeet | grey feathers, tree | no | 1 |
| | pine grosbeak | snow, pink feathers | no | 1 |
| | | sky, pink feathers | no | 1 |
| | | wing patches, tree, green background | no | 1 |
| | | grey feathers, orange head, snow | no | 1 |
| | | heavy chest, tree, wing patches | no | 1 |
| | | grey wings, wing patches, pink feathers | no | 1 |
| | | tree, pink feathers, eye, flower | no | 1 |

Table B.4: Mechanisms overview table after two rounds for *birds*

## B.3. Results for birds round 3

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **american goldfinch (3)** | 29 (59%) | 4 (8%) | 25 (51%) | 1 (2%) | 19 (39%) |
| **bufflehead (4)** | 32 (65%) | 0 | 32 (65%) | 0 | 17 (35%) |
| **downy woodpecker (4)** | 30 (60%) | 1 (2%) | 29 (58%) | 0 | 20 (40%) |
| **gila woodpecker (4)** | 30 (60%) | 2 (4%) | 28 (56%) | 0 | 20 (40%) |
| **hairy woodpecker (5)** | 30 (60%) | 4 (8%) | 26 (52%) | 0 | 20 (40%) |
| **hooded merganser (3)** | 30 (60%) | 3 (6%) | 27 (54%) | 0 | 20 (40%) |
| **lesser goldfinch (4)** | 29 (59%) | 2 (4%) | 27 (55%) | 1 (2%) | 19 (39%) |
| **mandarin duck (6)** | 29 (58%) | 1 (2%) | 28 (56%) | 1 (2%) | 20 (40%) |
| **monk parakeet (3)** | 29 (59%) | 5 (10%) | 24 (49%) | 1 (2%) | 19 (39%) |
| **pine grosbeak (3)** | 30 (63%) | 4 (8%) | 26 (54%) | 0 | 18 (38%) |

Table B.5: Requirements overview table after three rounds for *birds*

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| american goldfinch | american goldfinch | yellow breast, yellow belly, yellow back | yes | 2 |
| | | yellow breast, yellow belly, yellow back, black wings | yes | 2 |
| | | white bottom, tree | no | 1 |
| | | yellow back, tree, yellow crown | no | 1 |
| | | tree, yellow crown, yellow back | no | 1 |
| | | tree, yellow crown | no | 1 |
| | | yellow crown, tree, white bottom | no | 1 |
| | | yellow crown, green background, wing patch | no | 1 |
| | | foot, tree, green background, yellow belly, black crown | no | 1 |
| | | yellow belly, yellow breast, black crown | no | 1 |
| | | green background, black crown | no | 1 |
| | | yellow belly, yellow breast, foot | no | 1 |
| | | green background, beak, eye | no | 1 |
| | | snow, black wings, yellow belly, yellow breast, yellow crown | no | 1 |
| | | green background, yellow back, black wings, green belly, green breast | no | 1 |
| | | tree, sky | no | 1 |
| | | green background, wing patch, tree | no | 1 |
| | | green background, yellow breast, green belly, yellow crown | no | 1 |
| | | black crown, black wings, green breast, tree | no | 1 |
| | | foot, black wings, green breast, beak | no | 1 |
| | | green belly, green breast, green background, tree | no | 1 |
| | | foot | no | 1 |
| | | green belly, green breast, eye | no | 1 |
| | | green background, tree, wing patch | no | 1 |
| | | black wings, green belly, green breast, wing patch | no | 1 |
| | | green background, yellow back | no | 1 |
| | | yellow belly, black wings, green background | no | 1 |
| bufflehead | bufflehead | water | no | 2 |
| | | top wing, water | no | 2 |
| | | water, neck | no | 2 |
| | hooded merganser | water, eye, neck | no | 1 |
| | | water, grey feathers, rainbow crest, white spot | no | 1 |
| | bufflehead | water, white feathers, top wing | no | 1 |
| | | dark head, white spot, top wing, beak | no | 1 |
| | | beak, white feathers, water | no | 1 |
| | hooded merganser | water, white spot, grey feathers, neck | no | 1 |
| | bufflehead | water, grey feathers | no | 1 |
| | | water, black wings | no | 1 |
| | | rainbow crest, water | no | 1 |
| | | tree | no | 1 |
| | hooded merganser | water, neck | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | bufflehead | eye, water, white feathers, black wings | no | 1 |
| | hooded merganser | dark head, white spot, black wings, grey feathers, neck, water | no | 1 |
| | bufflehead | neck, wing patch | no | 1 |
| | hooded merganser | dark head, grey feathers, wing patch, neck | no | 1 |
| | | neck, water | no | 1 |
| | bufflehead | wing patch, neck, grey feathers | no | 1 |
| | | rainbow crest, wing patch, water | no | 1 |
| | | white feathers, tree | no | 1 |
| | | white spot, rainbow crest, water | no | 1 |
| | hooded merganser | water | no | 1 |
| | bufflehead | water, neck, wing patch, white spot | no | 1 |
| | | top wing, neck, rainbow crest, water | no | 1 |
| | | dark head, beak, black wings, neck, water | no | 1 |
| | hooded merganser | white spot, neck, water | no | 1 |
| | bufflehead | wing patch, neck, water | no | 1 |
| downy woodpecker | downy woodpecker | tree | no | 4 |
| | | tree, black white wing patches | no | 3 |
| | | black white wing patches, tree | no | 3 |
| | | black white wing patches, sky | no | 1 |
| | | black white wing patches, white breast, tree | no | 1 |
| | | tree, white breast, black white wing patches | no | 1 |
| | | white breast, tree | no | 1 |
| | lesser goldfinch | tree, black white wing patches, white breast | no | 1 |
| | downy woodpecker | sky, black white wing patches | no | 1 |
| | | tree, black white wing patches, white bottom | no | 1 |
| | hairy woodpecker | tree, neck | no | 1 |
| | downy woodpecker | white breast, white bottom, tree | no | 1 |
| | | foot, tree | no | 1 |
| | hairy woodpecker | white breast, black white wing patches, sky, tree | no | 1 |
| | downy woodpecker | red crown, black white wing patches, tree | no | 1 |
| | | black white wing patches, white breast | yes | 1 |
| | | black white wing patches, white breast, background leaf | no | 1 |
| | hairy woodpecker | tree | no | 1 |
| | downy woodpecker | black white wing patches, white breast, white bottom, tree | no | 1 |
| | | tree, sky | no | 1 |
| | | tree, white breast, sky | no | 1 |
| | | red crown, white breast | no | 1 |
| | | sky, white breast, black white wing patches | no | 1 |
| gila woodpecker | gila woodpecker | sky, cactus | no | 3 |
| | | green belly, sky | no | 1 |
| | | green neck, red crown | no | 1 |
| | monk parakeet | green belly, black white striped wings | no | 1 |
| | downy woodpecker | black white striped wings, sky, green crown | no | 1 |
| | | green crown, black white striped wings, tree | no | 1 |
| | | green neck, red crown, black white striped wings, tree | no | 1 |
| | american goldfinch | sky, black white striped wings, green crown | no | 1 |
| | downy woodpecker | tree, green belly | no | 1 |
| | lesser goldfinch | green background | no | 1 |
| | gila woodpecker | red crown, green breast, green background | no | 1 |
| | downy woodpecker | red crown, black white striped wings, green background, tree, beak | no | 1 |
| | | tree, beak, black white striped wings, green belly, green breast, green neck | no | 1 |
| | | sky, tree, red crown, green neck | no | 1 |
| | monk parakeet | green neck, black white striped wings, green belly, tree, sky | no | 1 |
| | pine grosbeak | water feeder, red crown | no | 1 |
| | gila woodpecker | beak, green belly, green breast, green neck, sky | no | 1 |
| | | green neck, black white striped wings, cactus, sky | no | 1 |
| | pine grosbeak | red crown, green neck, water feeder, green background | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | gila woodpecker | green crown, cactus, black white striped wings | no | 1 |
| | downy woodpecker | black white striped wings | yes | 1 |
| | hairy woodpecker | sky, green neck, red crown, black white striped wings | no | 1 |
| | monk parakeet | green neck, green belly, green breast, black white striped wings | yes | 1 |
| | gila woodpecker | red crown, black white striped wings, green breast, green belly | no | 1 |
| | | sky, green background, green breast, green neck, black white striped wings | no | 1 |
| | | cactus, green breast, green crown | no | 1 |
| | | sky, tree, green neck | no | 1 |
| | | green breast, green crown | no | 1 |
| hairy woodpecker | hairy woodpecker | tree | no | 2 |
| | | black and white wing patches, tree | no | 2 |
| | | sky | no | 1 |
| | | black wings, sky, throat stripe, beak | no | 1 |
| | monk parakeet | tree, black wings | no | 1 |
| | downy woodpecker | red crown, white breast, black wings | no | 1 |
| | hairy woodpecker | red crown, black and white wing patches, sky, tree | no | 1 |
| | | white breast, tree | no | 1 |
| | | throat stripe, white breast, white belly | no | 1 |
| | | white belly, tree, black and white wing patches | no | 1 |
| | | throat stripe, sky, white back stripe | no | 1 |
| | downy woodpecker | throat stripe, sky | no | 1 |
| | hairy woodpecker | beak, tree, eye | no | 1 |
| | bufflehead | white breast | yes | 1 |
| | hairy woodpecker | beak, red crown, white back stripe, black wings, green background | no | 1 |
| | downy woodpecker | eye, green background, black and white wing patches | no | 1 |
| | | red crown, black and white wing patches, sky, tree | no | 1 |
| | hairy woodpecker | green background, tree, black and white wing patches | no | 1 |
| | | white breast | yes | 1 |
| | | black wings, white breast | yes | 1 |
| | | red crown, tree | no | 1 |
| | downy woodpecker | tree, black and white wing patches, throat stripe | no | 1 |
| | hairy woodpecker | white belly, white breast, red crown | no | 1 |
| | | green background, tree, black and white wing patches, red crown | no | 1 |
| | | eye, green background | no | 1 |
| | | black wings, white breast, white belly | no | 1 |
| | downy woodpecker | tree, black wings, sky | no | 1 |
| | | black wings, white breast | yes | 1 |
| hooded merganser | hooded merganser | brown sides, eye | no | 2 |
| | | cinnamon crest | yes | 2 |
| | | dark feather texture, eye | no | 1 |
| | | black crest with white spot, eye, neck | no | 1 |
| | | eye, brown sides, water | no | 1 |
| | | eye, neck, water, brown sides | no | 1 |
| | | neck, water | no | 1 |
| | | cinnamon crest, water, dark feather texture | no | 1 |
| | | eye, dark feather texture | no | 1 |
| | | brown sides, eye, water | no | 1 |
| | | dark feather texture, water | no | 1 |
| | monk parakeet | brown sides, eye | no | 1 |
| | bufflehead | cinnamon crest, water, grey belly, dark feather texture | no | 1 |
| | lesser goldfinch | cinnamon crest, eye, dark feather texture, reed | no | 1 |
| | hooded merganser | cinnamon crest, water, beak, neck | no | 1 |
| | | black crest with white spot, eye, water | no | 1 |
| | | brown sides, water | no | 1 |
| | | cinnamon crest, water, dark feather texture, brown sides | no | 1 |
| | american goldfinch | water, cinnamon crest, dark feather texture | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | hooded merganser | neck, brown sides | no | 1 |
| | | water, dark feather texture, black crest with white spot | no | 1 |
| | | cinnamon crest, water | no | 1 |
| | bufflehead | brown sides, water | no | 1 |
| | | cinnamon crest, water | no | 1 |
| | hooded merganser | brown sides, neck | no | 1 |
| | | cinnamon crest, neck, water | no | 1 |
| | | water, dark feather texture, cinnamon crest | no | 1 |
| | | black crest with white spot | yes | 1 |
| lesser goldfinch | lesser goldfinch | green background, lighter wing patches | no | 2 |
| | | yellow belly, tree | no | 1 |
| | | flower, neck | no | 1 |
| | | beak, sky, yellow breast | no | 1 |
| | | beak, sky | no | 1 |
| | | black wings, lighter wing patches, beak, green background | no | 1 |
| | american goldfinch | black wings, yellow belly, sky | no | 1 |
| | lesser goldfinch | yellow back, yellow belly, lighter wing patches, green background | no | 1 |
| | | lighter wing patches, tree | no | 1 |
| | | black crown, yellow belly, neck accent | no | 1 |
| | | yellow belly, yellow breast, black crown | no | 1 |
| | | green background | no | 1 |
| | | green background, yellow back, neck, tree | no | 1 |
| | | black crown, yellow breast, yellow belly, yellow back | yes | 1 |
| | | green background, yellow breast, yellow belly | no | 1 |
| | | yellow belly, yellow breast, tree, beak | no | 1 |
| | | yellow belly, yellow breast, green background, neck | no | 1 |
| | american goldfinch | yellow belly, tree, sky | no | 1 |
| | lesser goldfinch | tree | no | 1 |
| | | neck, neck accent, sky | no | 1 |
| | | neck, yellow back, yellow belly | no | 1 |
| | | black wings, lighter wing patches | yes | 1 |
| | | sky | no | 1 |
| | | black wings, lighter wing patches, yellow back, yellow belly, sky | no | 1 |
| | | neck, black wings, yellow belly | no | 1 |
| | | darker wing patches, green background | no | 1 |
| | | sky, black wings, lighter wing patches | no | 1 |
| mandarin duck | mandarin duck | rainbow crest, golden sides, brown feathers, water | no | 1 |
| | | brown feathers, golden sides, water | no | 1 |
| | lesser goldfinch | golden sides, water, soil, background ornament | no | 1 |
| | mandarin duck | rainbow crest, brown feathers, neck | no | 1 |
| | american goldfinch | golden sides, brown feathers, neck, water | no | 1 |
| | mandarin duck | neck, golden sides, water, red beak | no | 1 |
| | | rainbow crest, neck, red beak, water | no | 1 |
| | hooded merganser | neck, water, white stripe below eye | no | 1 |
| | mandarin duck | rainbow crest, soil, brown feathers | no | 1 |
| | | red beak, golden sides, water | no | 1 |
| | | rainbow crest, golden sides, brown feathers, water, soil | no | 1 |
| | | rainbow crest, neck, dotted bottom | no | 1 |
| | lesser goldfinch | grey feathers, soil, dotted bottom | no | 1 |
| | mandarin duck | golden sides, rainbow crest, water | no | 1 |
| | | golden sides, brown feathers, long brown neck feathers | no | 1 |
| | | rainbow crest, dotted bottom, brown feathers, water, golden sides, long brown neck feathers | no | 1 |
| | | rainbow crest, long brown neck feathers, neck | no | 1 |
| | gila woodpecker | grey feathers, brown feathers, snow | no | 1 |
| | mandarin duck | rainbow crest, neck, golden sides, water | no | 1 |
| | pine grosbeak | long brown neck feathers, red beak, water, neck | no | 1 |
| | mandarin duck | brown feathers, rainbow crest, golden sides | no | 1 |
| | hooded merganser | golden sides, water | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | mandarin duck | rainbow crest, golden sides, white stripe below eye, neck, long brown neck feathers | no | 1 |
| | | rainbow crest | yes | 1 |
| | | rainbow crest, golden sides, grey feathers | no | 1 |
| | hooded merganser | water, rainbow crest, golden sides | no | 1 |
| | mandarin duck | long brown neck feathers, water, neck | no | 1 |
| | | golden sides, long brown neck feathers | no | 1 |
| | | water, rainbow crest | no | 1 |
| monk parakeet | monk parakeet | green feathers, light breast, light crown | yes | 4 |
| | | green feathers, light crown, sky, tree | no | 1 |
| | | green feathers | yes | 1 |
| | | fence, light crown, sky, green feathers | no | 1 |
| | | blue wingtips, beak, green feathers, tree | no | 1 |
| | | green feathers, tree, sky | no | 1 |
| | | sky, fence, beak | no | 1 |
| | | sky, tree, green feathers, light breast | no | 1 |
| | | light breast, tree | no | 1 |
| | | green feathers, tree | no | 1 |
| | | sand | no | 1 |
| | | sky, tree | no | 1 |
| | | sky, green feathers, light crown | no | 1 |
| | | beak, light throat, light crown, sky | no | 1 |
| | | light crown, light throat, sky, green feathers | no | 1 |
| | american goldfinch | light breast, green feathers, tree | no | 1 |
| | monk parakeet | light throat, urban objects | no | 1 |
| | | blue wingtips, green background | no | 1 |
| | | beak, green feathers, green background | no | 1 |
| | | beak, blue wingtips | no | 1 |
| | | green background | no | 1 |
| | | sky | no | 1 |
| | | light breast, green feathers | no | 1 |
| | | green background, green feathers | no | 1 |
| | | light breast, blue wingtips | no | 1 |
| | | beak, light breast, light crown | no | 1 |
| pine grosbeak | pine grosbeak | pink feathers, grey wings | yes | 3 |
| | | pink feathers, snow, cheek | no | 1 |
| | american goldfinch | orange head, tree, green background | no | 1 |
| | | grey feathers, orange head, green background | no | 1 |
| | pine grosbeak | grey feathers, orange head, sky | no | 1 |
| | | grey feathers, orange head, tree | no | 1 |
| | monk parakeet | sky, snow, tree, wing patches | no | 1 |
| | pine grosbeak | wing patches, snow, grey feathers | no | 1 |
| | | wing patches, pink feathers, sky | no | 1 |
| | | grey feathers, heavy chest | no | 1 |
| | monk parakeet | grey feathers, tree | no | 1 |
| | pine grosbeak | snow, pink feathers | no | 1 |
| | | sky, pink feathers | no | 1 |
| | | wing patches, tree, green background | no | 1 |
| | | grey feathers, orange head, snow | no | 1 |
| | | heavy chest, tree, wing patches | no | 1 |
| | | grey wings, wing patches, pink feathers | no | 1 |
| | | tree, pink feathers, eye, flower | no | 1 |
| | mandarin duck | grey feathers, orange head | yes | 1 |
| | pine grosbeak | sky, heavy chest | no | 1 |
| | lesser goldfinch | grey feathers, grey wings | no | 1 |
| | pine grosbeak | grey wings, tree | no | 1 |
| | monk parakeet | sky, tree | no | 1 |
| | american goldfinch | tree, heavy chest, grey wings | no | 1 |
| | | yellow feathers, grey feathers, green background | no | 1 |
| | bufflehead | grey feathers, orange head, sky | no | 1 |
| | american goldfinch | grey feathers, sky, tree | no | 1 |
| | lesser goldfinch | orange head, pink feathers | no | 1 |

Table B.6: Mechanisms overview table after three rounds for *birds*

# B.4. Results for birds round 4

| Class (requirements) | Requirement in image | Mechanism in heatmap | Mechanism not in heatmap | Requirement not in image | Unnannotated images |
|---|---|---|---|---|---|
| **american_goldfinch (5)** | 48 (98%) | 7 (14%) | 41 (84%) | 1 (2%) | 0 |
| **bufflehead (4)** | 48 (98%) | 5 (10%) | 43 (88%) | 1 (2%) | 0 |
| **downy_woodpecker (4)** | 50 (100%) | 8 (16%) | 42 (84%) | 0 | 0 |
| **gila_woodpecker (4)** | 50 (100%) | 3 (6%) | 47 (94%) | 0 | 0 |
| **hairy_woodpecker (5)** | 50 (100%) | 4 (8%) | 46 (92%) | 0 | 0 |
| **hooded_merganser (3)** | 50 (100%) | 6 (12%) | 44 (88%) | 0 | 0 |
| **lesser_goldfinch (5)** | 46 (94%) | 3 (6%) | 43 (88%) | 3 (6%) | 0 |
| **mandarin_duck (6)** | 47 (94%) | 4 (8%) | 43 (86%) | 3 (6%) | 0 |
| **monk_parakeet (3)** | 48 (98%) | 5 (10%) | 43 (88%) | 1 (2%) | 0 |
| **pine_grosbeak (3)** | 47 (98%) | 12 (25%) | 35 (73%) | 1 (2%) | 0 |

Table B.7: Requirements overview table after four rounds for *birds*

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| american goldfinch | american goldfinch | green background | no | 3 |
| | | yellow breast, yellow belly, yellow back | yes | 2 |
| | | yellow breast, yellow belly, yellow back, black wings | yes | 2 |
| | | yellow breast, yellow belly, tree | no | 2 |
| | | yellow breast, yellow belly, black wings | yes | 2 |
| | | white bottom, tree | no | 1 |
| | | yellow back, tree, yellow crown | no | 1 |
| | | tree, yellow crown, yellow back | no | 1 |
| | | tree, yellow crown | no | 1 |
| | | yellow crown, tree, white bottom | no | 1 |
| | | yellow crown, green background, wing patch | no | 1 |
| | | foot, tree, green background, yellow belly, black crown | no | 1 |
| | | yellow belly, yellow breast, black crown | no | 1 |
| | | green background, black crown | no | 1 |
| | | yellow belly, yellow breast, foot | no | 1 |
| | | green background, beak, eye | no | 1 |
| | | snow, black wings, yellow belly, yellow breast, yellow crown | no | 1 |
| | | green background, yellow back, black wings, green belly, green breast | no | 1 |
| | | tree, sky | no | 1 |
| | | green background, wing patch, tree | no | 1 |
| | | green background, yellow breast, green belly, yellow crown | no | 1 |
| | | black crown, black wings, green breast, tree | no | 1 |
| | | foot, black wings, green breast, beak | no | 1 |
| | | green belly, green breast, green background, tree | no | 1 |
| | | foot | no | 1 |
| | | green belly, green breast, eye | no | 1 |
| | | green background, tree, wing patch | no | 1 |
| | | black wings, green belly, green breast, wing patch | no | 1 |
| | | green background, yellow back | no | 1 |
| | | yellow belly, black wings, green background | no | 1 |
| | | black wings, green background, wing patch | no | 1 |
| | | sky, eye | no | 1 |
| | | sky | no | 1 |
| | | sky, yellow crown, green breast, green belly | no | 1 |
| | | wing patch, green background | no | 1 |
| | | yellow breast, yellow belly, sky | no | 1 |
| | | yellow belly, yellow breast, sky | no | 1 |
| | pine grosbeak | yellow breast, yellow belly, black wings | yes | 1 |
| | american goldfinch | yellow breast, yellow belly, yellow crown, green background | no | 1 |
| | | sky, white bottom | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | green background, wing patch, black wings, eye | no | 1 |
| | | green background, yellow belly, yellow breast | no | 1 |
| bufflehead | bufflehead | water | no | 3 |
| | | white feathers, water | no | 3 |
| | | top wing, water | no | 2 |
| | | water, neck | no | 2 |
| | hooded merganser | white spot, rainbow crest | yes | 2 |
| | bufflehead | white spot, rainbow crest | yes | 2 |
| | hooded merganser | water, eye, neck | no | 1 |
| | | water, grey feathers, rainbow crest, white spot | no | 1 |
| | bufflehead | water, white feathers, top wing | no | 1 |
| | | dark head, white spot, top wing, beak | no | 1 |
| | | beak, white feathers, water | no | 1 |
| | hooded merganser | water, white spot, grey feathers, neck | no | 1 |
| | bufflehead | water, grey feathers | no | 1 |
| | | water, black wings | no | 1 |
| | | rainbow crest, water | no | 1 |
| | | tree | no | 1 |
| | hooded merganser | water, neck | no | 1 |
| | bufflehead | eye, water, white feathers, black wings | no | 1 |
| | hooded merganser | dark head, white spot, black wings, grey feathers, neck, water | no | 1 |
| | bufflehead | neck, wing patch | no | 1 |
| | hooded merganser | dark head, grey feathers, wing patch, neck | no | 1 |
| | | neck, water | no | 1 |
| | bufflehead | wing patch, neck, grey feathers | no | 1 |
| | | rainbow crest, wing patch, water | no | 1 |
| | | white feathers, tree | no | 1 |
| | | white spot, rainbow crest, water | no | 1 |
| | hooded merganser | water | no | 1 |
| | bufflehead | water, neck, wing patch, white spot | no | 1 |
| | | top wing, neck, rainbow crest, water | no | 1 |
| | | dark head, beak, black wings, neck, water | no | 1 |
| | hooded merganser | white spot, neck, water | no | 1 |
| | bufflehead | wing patch, neck, water | no | 1 |
| | | white spot, water | no | 1 |
| | hooded merganser | neck, dark head, white spot, grey feathers | no | 1 |
| | bufflehead | water, eye, grey feathers | no | 1 |
| | hairy woodpecker | neck, grey feathers, water | no | 1 |
| | bufflehead | rainbow crest, white feathers, water | no | 1 |
| | | black wings, grey feathers | yes | 1 |
| | | wing patch, water | no | 1 |
| | | rainbow crest, water, white feathers, white spot | no | 1 |
| downy woodpecker | downy woodpecker | black white wing patches | yes | 5 |
| | | tree | no | 4 |
| | | tree, black white wing patches | no | 4 |
| | | black white wing patches, tree | no | 4 |
| | | black white wing patches, white breast | yes | 3 |
| | | red crown, tree | no | 2 |
| | | sky | no | 2 |
| | | black white wing patches, sky | no | 1 |
| | | black white wing patches, white breast, tree | no | 1 |
| | | tree, white breast, black white wing patches | no | 1 |
| | | white breast, tree | no | 1 |
| | lesser goldfinch | tree, black white wing patches, white breast | no | 1 |
| | downy woodpecker | sky, black white wing patches | no | 1 |
| | | tree, black white wing patches, white bottom | no | 1 |
| | hairy woodpecker | tree, neck | no | 1 |
| | downy woodpecker | white breast, white bottom, tree | no | 1 |
| | | foot, tree | no | 1 |
| | hairy woodpecker | white breast, black white wing patches, sky, tree | no | 1 |
| | downy woodpecker | red crown, black white wing patches, tree | no | 1 |
| | | black white wing patches, white breast, background leaf | no | 1 |
| | hairy woodpecker | tree | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | downy woodpecker | black white wing patches, white breast, white bottom, tree | no | 1 |
| | | tree, sky | no | 1 |
| | | tree, white breast, sky | no | 1 |
| | | red crown, white breast | no | 1 |
| | | sky, white breast, black white wing patches | no | 1 |
| | hairy woodpecker | black white wing patches, sky, tree | no | 1 |
| | downy woodpecker | white breast, sky | no | 1 |
| | | tree, black white wing patches, white breast | no | 1 |
| | | tree, white breast | no | 1 |
| | | sky, black white wing patches, tree | no | 1 |
| | | black white wing patches, tree, sky | no | 1 |
| | | red crown, black white wing patches, sky | no | 1 |
| gila woodpecker | gila woodpecker | sky, cactus | no | 3 |
| | | green belly, sky | no | 1 |
| | | green neck, red crown | no | 1 |
| | monk parakeet | green belly, black white striped wings | no | 1 |
| | downy woodpecker | black white striped wings, sky, green crown | no | 1 |
| | | green crown, black white striped wings, tree | no | 1 |
| | downy woodpecker | green neck, red crown, black white striped wings, tree | no | 1 |
| | american goldfinch | sky, black white striped wings, green crown | no | 1 |
| | downy woodpecker | tree, green belly | no | 1 |
| | lesser goldfinch | green background | no | 1 |
| | gila woodpecker | red crown, green breast, green background | no | 1 |
| | downy woodpecker | red crown, black white striped wings, green background, tree, beak | no | 1 |
| | | tree, beak, black white striped wings, green belly, green breast, green neck | no | 1 |
| | | sky, tree, red crown, green neck | no | 1 |
| | monk parakeet | green neck, black white striped wings, green belly, tree, sky | no | 1 |
| | pine grosbeak | water feeder, red crown | no | 1 |
| | gila woodpecker | beak, green belly, green breast, green neck, sky | no | 1 |
| | | green neck, black white striped wings, cactus, sky | no | 1 |
| | pine grosbeak | red crown, green neck, water feeder, green background | no | 1 |
| | gila woodpecker | green crown, cactus, black white striped wings | no | 1 |
| | downy woodpecker | black white striped wings | yes | 1 |
| | hairy woodpecker | sky, green neck, red crown, black white striped wings | no | 1 |
| | monk parakeet | green neck, green belly, green breast, black white striped wings | yes | 1 |
| | gila woodpecker | red crown, black white striped wings, green breast, green belly | no | 1 |
| | | sky, green background, green breast, green neck, black white striped wings | no | 1 |
| | | cactus, green breast, green crown | no | 1 |
| | | sky, tree, green neck | no | 1 |
| | | green breast, green crown | no | 1 |
| | downy woodpecker | black white striped wings, sky | no | 1 |
| | gila woodpecker | red crown, black white striped wings, sky, tree | no | 1 |
| | downy woodpecker | tree, green neck | no | 1 |
| | american goldfinch | green breast, green belly, green neck, green background | no | 1 |
| | hairy woodpecker | green belly, green breast, sky | no | 1 |
| | gila woodpecker | black white striped wings, sky | no | 1 |
| | | red crown, green neck, black white striped wings, cactus | no | 1 |
| | | green crown, sky, green belly, cactus | no | 1 |
| | monk parakeet | green belly, green breast | no | 1 |
| | lesser goldfinch | tree | no | 1 |
| | hairy woodpecker | black white striped wings | yes | 1 |
| | gila woodpecker | cactus, red crown, black white striped wings, green background | no | 1 |
| | | cactus, green belly, green breast, sky | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | red crown, cactus, black white striped wings, sky | no | 1 |
| | | cactus, red crown, sky, black white striped wings | no | 1 |
| | | green crown, black white striped wings, sky | no | 1 |
| | downy woodpecker | tree, sky, green neck | no | 1 |
| | gila woodpecker | cactus, black white striped wings, sky | no | 1 |
| | | red crown, green neck, green breast | no | 1 |
| | hairy woodpecker | tree, sky | no | 1 |
| hairy woodpecker | hairy woodpecker | black and white wing patches, tree | no | 3 |
| | | tree | no | 2 |
| | | tree, red crown, black and white wing patches | no | 2 |
| | | red crown, sky | no | 2 |
| | | sky | no | 1 |
| | | black wings, sky, throat stripe, beak | no | 1 |
| | monk parakeet | tree, black wings | no | 1 |
| | downy woodpecker | red crown, white breast, black wings | no | 1 |
| | hairy woodpecker | red crown, black and white wing patches, sky, tree | no | 1 |
| | | white breast, tree | no | 1 |
| | | throat stripe, white breast, white belly | no | 1 |
| | | white belly, tree, black and white wing patches | no | 1 |
| | hairy woodpecker | throat stripe, sky, white back stripe | no | 1 |
| | downy woodpecker | throat stripe, sky | no | 1 |
| | hairy woodpecker | beak, tree, eye | no | 1 |
| | bufflehead | white breast | yes | 1 |
| | hairy woodpecker | beak, red crown, white back stripe, black wings, green background | no | 1 |
| | downy woodpecker | eye, green background, black and white wing patches | no | 1 |
| | | red crown, black and white wing patches, sky, tree | no | 1 |
| | hairy woodpecker | green background, tree, black and white wing patches | no | 1 |
| | hairy woodpecker | white breast | yes | 1 |
| | | black wings, white breast | yes | 1 |
| | | red crown, tree | no | 1 |
| | downy woodpecker | tree, black and white wing patches, throat stripe | no | 1 |
| | hairy woodpecker | white belly, white breast, red crown | no | 1 |
| | | green background, tree, black and white wing patches, red crown | no | 1 |
| | | eye, green background | no | 1 |
| | | black wings, white breast, white belly | no | 1 |
| | downy woodpecker | tree, black wings, sky | no | 1 |
| | | black wings, white breast | yes | 1 |
| | hairy woodpecker | black and white wing patches, green background, tree | no | 1 |
| | monk parakeet | tree, red crown, black and white wing patches | no | 1 |
| | hairy woodpecker | red crown, tree, black and white wing patches | no | 1 |
| | | black and white wing patches, sky | no | 1 |
| | | red crown, black and white wing patches, sky | no | 1 |
| | | tree, red crown, white belly | no | 1 |
| | | green background, sky | no | 1 |
| | | tree, red crown, black and white wing patches, white belly | no | 1 |
| | | tree, red crown | no | 1 |
| | | black and white wing patches, red crown, tree | no | 1 |
| | | green background, tree | no | 1 |
| | lesser goldfinch | tree, red crown | no | 1 |
| | hairy woodpecker | black and white wing patches, tree, throat stripe, red crown | no | 1 |
| | downy woodpecker | throat stripe, white belly, tree | no | 1 |
| | hairy woodpecker | tree, sky | no | 1 |
| hooded merganser | hooded merganser | brown sides, eye | no | 2 |
| | | eye, brown sides, water | no | 2 |
| | | brown sides, water | no | 2 |
| | | cinnamon crest | yes | 2 |
| | | black crest with white spot | yes | 2 |
| | | water, brown sides | no | 2 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | brown sides | yes | 2 |
| | | dark feather texture, eye | no | 1 |
| | | black crest with white spot, eye, neck | no | 1 |
| | | eye, neck, water, brown sides | no | 1 |
| | | neck, water | no | 1 |
| | | cinnamon crest, water, dark feather texture | no | 1 |
| | | eye, dark feather texture | no | 1 |
| | | brown sides, eye, water | no | 1 |
| | | dark feather texture, water | no | 1 |
| | monk parakeet | brown sides, eye | no | 1 |
| | bufflehead | cinnamon crest, water, grey belly, dark feather texture | no | 1 |
| | lesser goldfinch | cinnamon crest, eye, dark feather texture, reed | no | 1 |
| | hooded merganser | cinnamon crest, water, beak, neck | no | 1 |
| | | black crest with white spot, eye, water | no | 1 |
| | | cinnamon crest, water, dark feather texture, brown sides | no | 1 |
| | american goldfinch | water, cinnamon crest, dark feather texture | no | 1 |
| | hooded merganser | neck, brown sides | no | 1 |
| | | water, dark feather texture, black crest with white spot | no | 1 |
| | | cinnamon crest, water | no | 1 |
| | bufflehead | brown sides, water | no | 1 |
| | bufflehead | cinnamon crest, water | no | 1 |
| | hooded merganser | brown sides, neck | no | 1 |
| | | cinnamon crest, neck, water | no | 1 |
| | | water, dark feather texture, cinnamon crest | no | 1 |
| | | cinnamon crest, neck, brown sides | no | 1 |
| | | cinnamon crest, dark feather texture, water | no | 1 |
| | pine grosbeak | cinnamon crest, brown sides | no | 1 |
| | hooded merganser | black crest with white spot, eye, brown sides, water | no | 1 |
| | | cinnamon crest, eye, dark feather texture | no | 1 |
| | | black crest with white spot, neck, brown sides, water | no | 1 |
| | bufflehead | cinnamon crest, snow | no | 1 |
| | hooded merganser | grey belly, cinnamon crest, water | no | 1 |
| | gila woodpecker | cinnamon crest, dark feather texture | no | 1 |
| | hooded merganser | black crest with white spot, eye | no | 1 |
| | | water | no | 1 |
| | | neck, black crest with white spot, water | no | 1 |
| | bufflehead | water, cinnamon crest | no | 1 |
| lesser goldfinch | lesser goldfinch | green background | no | 3 |
| | | green background, lighter wing patches | no | 2 |
| | | yellow belly, tree | no | 1 |
| | | flower, neck | no | 1 |
| | | beak, sky, yellow breast | no | 1 |
| | | beak, sky | no | 1 |
| | | black wings, lighter wing patches, beak, green background | no | 1 |
| | american goldfinch | black wings, yellow belly, sky | no | 1 |
| | lesser goldfinch | yellow back, yellow belly, lighter wing patches, green background | no | 1 |
| | | lighter wing patches, tree | no | 1 |
| | | black crown, yellow belly, neck accent | no | 1 |
| | | yellow belly, yellow breast, black crown | no | 1 |
| | | green background, yellow back, neck, tree | no | 1 |
| | | black crown, yellow breast, yellow belly, yellow back | yes | 1 |
| | | green background, yellow breast, yellow belly | no | 1 |
| | | yellow belly, yellow breast, tree, beak | no | 1 |
| | | yellow belly, yellow breast, green background, neck | no | 1 |
| | american goldfinch | yellow belly, tree, sky | no | 1 |
| | lesser goldfinch | tree | no | 1 |
| | | neck, neck accent, sky | no | 1 |
| | | neck, yellow back, yellow belly | no | 1 |
| | | black wings, lighter wing patches | yes | 1 |
| | | sky | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | | black wings, lighter wing patches, yellow back, yellow belly, sky | no | 1 |
| | | lighter wing patches, black wings, sky, yellow belly, yellow breast | no | 1 |
| | | neck, black wings, yellow belly | no | 1 |
| | | darker wing patches, green background | no | 1 |
| | | sky, black wings, lighter wing patches | no | 1 |
| | american goldfinch | green background, tree | no | 1 |
| | | flower, tree, black back | no | 1 |
| | lesser goldfinch | yellow belly | no | 1 |
| | | darker wing patches, sky, lighter wing patches, pale green feathers | no | 1 |
| | | sky, lighter wing patches, tree | no | 1 |
| | | sky, green background | no | 1 |
| | | beak, black back, green background | no | 1 |
| | | yellow back, yellow belly, green background | no | 1 |
| | | darker wing patches, lighter wing patches, green background | no | 1 |
| | | sky, yellow breast, yellow belly | no | 1 |
| | | black crown, green background, yellow breast | no | 1 |
| | | darker wing patches, lighter wing patches, sky | no | 1 |
| | hairy woodpecker | yellow breast, yellow belly, yellow back | yes | 1 |
| | lesser goldfinch | sky, tree, darker wing patches, lighter wing patches | no | 1 |
| | | neck, lighter wing patches, pale green feathers | no | 1 |
| mandarin duck | mandarin duck | rainbow crest | yes | 3 |
| | | golden sides, long brown neck feathers | no | 2 |
| | | rainbow crest, golden sides, brown feathers, water | no | 1 |
| | | brown feathers, golden sides, water | no | 1 |
| | lesser goldfinch | golden sides, water, soil, background ornament | no | 1 |
| | mandarin duck | rainbow crest, brown feathers, neck | no | 1 |
| | american goldfinch | golden sides, brown feathers, neck, water | no | 1 |
| | mandarin duck | neck, golden sides, water, red beak | no | 1 |
| | | rainbow crest, neck, red beak, water | no | 1 |
| | hooded merganser | neck, water, white stripe below eye | no | 1 |
| | mandarin duck | rainbow crest, soil, brown feathers | no | 1 |
| | | red beak, golden sides, water | no | 1 |
| | | rainbow crest, golden sides, brown feathers, water, soil | no | 1 |
| | | rainbow crest, neck, dotted bottom | no | 1 |
| | lesser goldfinch | grey feathers, soil, dotted bottom | no | 1 |
| | mandarin duck | golden sides, rainbow crest, water | no | 1 |
| | | golden sides, brown feathers, long brown neck feathers | no | 1 |
| | | rainbow crest, dotted bottom, brown feathers, water, golden sides, long brown neck feathers | no | 1 |
| | | rainbow crest, long brown neck feathers, neck | no | 1 |
| | gila woodpecker | grey feathers, brown feathers, snow | no | 1 |
| | mandarin duck | rainbow crest, neck, golden sides, water | no | 1 |
| | pine grosbeak | long brown neck feathers, red beak, water, neck | no | 1 |
| | mandarin duck | brown feathers, rainbow crest, golden sides | no | 1 |
| | hooded merganser | golden sides, water | no | 1 |
| | mandarin duck | rainbow crest, golden sides, white stripe below eye, neck, long brown neck feathers | no | 1 |
| | | rainbow crest, golden sides, grey feathers | no | 1 |
| | hooded merganser | water, rainbow crest, golden sides | no | 1 |
| | mandarin duck | long brown neck feathers, water, neck | no | 1 |
| | | water, rainbow crest | no | 1 |
| | downy woodpecker | rainbow crest, grey feathers, water | no | 1 |
| | mandarin duck | brown feathers, rainbow crest | no | 1 |
| | | long brown neck feathers, dotted bottom, golden sides | no | 1 |
| | lesser goldfinch | water | no | 1 |
| | mandarin duck | golden sides, water | no | 1 |
| | | water, golden sides | no | 1 |
| | hooded merganser | water | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | mandarin duck | golden sides | yes | 1 |
| | | rainbow crest, water | no | 1 |
| | | rainbow crest, golden sides | no | 1 |
| | | golden sides, long brown neck feathers, beak | no | 1 |
| | lesser goldfinch | long brown neck feathers, soil, green background | no | 1 |
| | mandarin duck | water | no | 1 |
| | | golden sides, long brown neck feathers, soil | no | 1 |
| | | golden sides, long brown neck feathers, rainbow crest, neck | no | 1 |
| monk parakeet | monk parakeet | green feathers, light breast, light crown | yes | 4 |
| | | sky, tree | no | 3 |
| | | green background, green feathers | no | 2 |
| | | green feathers, light crown, sky, tree | no | 1 |
| | | green feathers | yes | 1 |
| | | fence, light crown, sky, green feathers | no | 1 |
| | | blue wingtips, beak, green feathers, tree | no | 1 |
| | | green feathers, tree, sky | no | 1 |
| | | sky, fence, beak | no | 1 |
| | | sky, tree, green feathers, light breast | no | 1 |
| | | light breast, tree | no | 1 |
| | | green feathers, tree | no | 1 |
| | | sand | no | 1 |
| | | sky, green feathers, light crown | no | 1 |
| | | beak, light throat, light crown, sky | no | 1 |
| | | light crown, light throat, sky, green feathers | no | 1 |
| | american goldfinch | light breast, green feathers, tree | no | 1 |
| | monk parakeet | light throat, urban objects | no | 1 |
| | | blue wingtips, green background | no | 1 |
| | | beak, green feathers, green background | no | 1 |
| | | beak, blue wingtips | no | 1 |
| | | green background | no | 1 |
| | | sky | no | 1 |
| | | light breast, green feathers | no | 1 |
| | | light breast, blue wingtips | no | 1 |
| | | beak, light breast, light crown | no | 1 |
| | | sky, light breast | no | 1 |
| | | sky, green feathers | no | 1 |
| | | light crown, light breast, green feathers, sky | no | 1 |
| | | tree, sky, green feathers | no | 1 |
| | american goldfinch | green background | no | 1 |
| | monk parakeet | beak, light crown, green background, green feathers | no | 1 |
| | | tree, light breast, beak | no | 1 |
| | | green feathers, light breast, sky | no | 1 |
| | american goldfinch | sky, tree, light breast | no | 1 |
| | monk parakeet | sky, beak, light breast, green feathers | no | 1 |
| | | beak, urban objects, green feathers, light breast, sky | no | 1 |
| | | light crown, sky, light breast, beak, green feathers | no | 1 |
| | | beak, green background, light breast | no | 1 |
| | | green feathers, tree, light breast | no | 1 |
| | | human, green feathers | no | 1 |
| | american goldfinch | sky, tree | no | 1 |
| pine grosbeak | pine grosbeak | pink feathers, grey wings | yes | 4 |
| | | grey feathers, orange head | yes | 4 |
| | american goldfinch | grey feathers, orange head | yes | 2 |
| | pine grosbeak | pink feathers, grey wings, snow | no | 2 |
| | | pink feathers, snow, cheek | no | 1 |
| | american goldfinch | orange head, tree, green background | no | 1 |
| | | grey feathers, orange head, green background | no | 1 |
| | pine grosbeak | grey feathers, orange head, sky | no | 1 |
| | | grey feathers, orange head, tree | no | 1 |
| | monk parakeet | sky, snow, tree, wing patches | no | 1 |
| | pine grosbeak | wing patches, snow, grey feathers | no | 1 |
| | | wing patches, pink feathers, sky | no | 1 |
| | | grey feathers, heavy chest | no | 1 |
| | monk parakeet | grey feathers, tree | no | 1 |

Continued on next page

continued from previous page

| True label | Predicted label | Mechanism | Req? | Count |
|---|---|---|---|---|
| | pine grosbeak | snow, pink feathers | no | 1 |
| | | sky, pink feathers | no | 1 |
| | | wing patches, tree, green background | no | 1 |
| | | grey feathers, orange head, snow | no | 1 |
| | | heavy chest, tree, wing patches | no | 1 |
| | | grey wings, wing patches, pink feathers | no | 1 |
| | | tree, pink feathers, eye, flower | no | 1 |
| | mandarin duck | grey feathers, orange head | yes | 1 |
| | pine grosbeak | sky, heavy chest | no | 1 |
| | lesser goldfinch | grey feathers, grey wings | no | 1 |
| | pine grosbeak | grey wings, tree | no | 1 |
| | monk parakeet | sky, tree | no | 1 |
| | american goldfinch | tree, heavy chest, grey wings | no | 1 |
| | | yellow feathers, grey feathers, green background | no | 1 |
| | bufflehead | grey feathers, orange head, sky | no | 1 |
| | american goldfinch | grey feathers, sky, tree | no | 1 |
| | lesser goldfinch | orange head, pink feathers | no | 1 |
| | monk parakeet | tree, orange head, grey feathers, sky | no | 1 |
| | pine grosbeak | snow, pink feathers, wing patches | no | 1 |
| | | snow, tree, pink feathers, grey wings | no | 1 |
| | | grey wings, pink feathers, tree | no | 1 |
| | | sky, grey feathers, orange head | no | 1 |
| | downy woodpecker | snow, tree, orange head, grey feathers | no | 1 |
| | pine grosbeak | snow, orange head, grey feathers | no | 1 |
| | lesser goldfinch | grey feathers, orange head | yes | 1 |

Table B.8: Mechanisms overview table after four rounds for *birds*

# C
# Proposed Improvements for Brickroutine

Suggested improvement on existing functionalities:

- Make MongoDB save the annotated images in one operation asynchronously instead of doing a write per image.
- When image annotations are reset, either on an individual level or all at once, also undo the ’entire concepts fields.
- Clicking on a row with a remark now shows the heatmap, while it makes more sense to show the original image.

# D

# Readme from Code Repository

Birckroutine is a system to debug and explain computer vision models by iteratively(routinely) finding the human-comprehensible concepts (bricks) that helped an AI-algorithm make a certain classification.

The system uses RabbitMQ as a message broker and is build with docker containers

## Running

Prerequisites: Docker Docker Compose (For linux, comes included with Docker Desktop for Mac and Windows)

- For the first time, simply run in rootfolder:

```
docker-compose -f ./docker-compose.yml up
```

- To rebuild the images

```
docker-compose -f ./docker-compose.yml up --build
```

- To select a specific set of services

```
docker-compose -f ./docker-compose.yml up brickroutine-api brickroutine-ui rabbitmq --build
```

- To shut down the docker containers

```
docker-compose -f ./docker-compose.yml down
```

- To remove all docker images when you don't use it anymore, simply run

```
docker system prune -a
```

## Developing

```
docker-compose -f ./docker-compose.debug.yml up
```

Or, as recommended, install the docker extension in vscode. In order to easily debug python code in brickroutine-heatmaps and the API in brickroutine-webapi, I stronly recommend using the development container files and setting up a development container. In that way, you don't have to install all the dotnet and python dependencies on your system and can easily remove the containers when done. When in the development container environment, just debug as you would usally and the containers are attached to the docker network (brickroutine_network). The Frontend (brickroutine-ui) runs inside a node container and refreshes automatically when you change a .tsx file.

Unix/Windows
For Unix (mac/linux) Just use docker and docker-compose from command line or vscode extension. For windows, do use Docker desktop with WSL 2 backend and run clone the project folder inisde a mounted WSL directory and not in a Windows directory to leverage file system compatibility (when developing the UI) as per best practices described `https://code.visualstudio.com/docs/containers/overview` here.

## Using the system

There is a folder Test_Dataset in the root of this repo, which can be uploaded on the pane 'Dataset'. It contains of 300 images and a accompanying csv file

## Credentials

RabbitMQ
guest/guest ### Azure blob storage This system uses azure blob storage to upload and retrieve images Read more. For just experimental use, the free tier is sufuccient. TU Delft students and employees are eligible for cloud credits. In order to use this, make an azure account (TU delft email can be used) and create a blob store resource. The connection string can be entered in the appsettings.json of Brickroutine.WebAPI. Update the docker-compose.yml file with the appropriate link.

# Bibliography


[1] Yali Amit, Pedro Felzenszwalb, and Ross Girshick. “Object detection”. In: *Computer Vision: A Reference Guide* (2020), pp. 1–9.

[2] Agathe Balayn et al. “What do You Mean? Interpreting Image Classification with Crowdsourced Concept Extraction and Analysis”. In: *The World Wide Web Conference*. ACM. 2021, to appear.

[3] Solon Barocas and Andrew D Selbst. “Big data’s disparate impact”. In: *Calif. L. Rev.* 104 (2016), p. 671.

[4] Kent Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[5] Eric Breck et al. “The ML test score: A rubric for ML production readiness and technical debt reduction”. In: *2017 IEEE International Conference on Big Data (Big Data)*. IEEE. 2017, pp. 1123–1132.

[6] Joy Buolamwini and Timnit Gebru. “Gender shades: Intersectional accuracy disparities in commercial gender classification”. In: *Conference on fairness, accountability and transparency*. PMLR. 2018, pp. 77–91.

[7] Aylin Caliskan, Joanna J Bryson, and Arvind Narayanan. “Semantics derived automatically from language corpora contain human-like biases”. In: *Science* 356.6334 (2017), pp. 183–186.

[8] Rick Cattell. “Scalable SQL and NoSQL data stores”. In: *Acm Sigmod Record* 39.4 (2011), pp. 12–27.

[9] Alexandra Chouldechova and Aaron Roth. “The frontiers of fairness in machine learning”. In: *arXiv preprint arXiv:1810.08810* (2018).

[10] IEEE Standards Coordinating Committee et al. “IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990). Los Alamitos”. In: *CA: IEEE Computer Society* 169 (1990), p. 132.

[11] Lorenzo De Lauretis. “From monolithic architecture to microservices architecture”. In: *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2019, pp. 93–96.

[12] Finale Doshi-Velez and Been Kim. “Towards a rigorous science of interpretable machine learning”. In: *arXiv preprint arXiv:1702.08608* (2017).

[13] Warren J von Eschenbach. “Transparency and the Black Box Problem: Why We Do Not Trust AI”. In: *Philosophy & Technology* (2021), pp. 1–16.

[14] Amirata Ghorbani et al. “Towards automatic concept-based explanations”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[15] Shivakumar R Goniwada. “Event-Driven Architecture”. In: *Cloud Native Architecture and Design*. Springer, 2022, pp. 241–294.

[16] Kinnary Jangla. *Accelerating Development Velocity Using Docker: Docker Across Microservices*. Apress, 2018.

[17] David Jaramillo, Duy V Nguyen, and Robert Smart. “Leveraging microservices architecture by using Docker technology”. In: *SoutheastCon 2016*. IEEE. 2016, pp. 1–5.

[18] Daniel Kang et al. “Model assertions for debugging machine learning”. In: *NeurIPS MLSys Workshop*. 2018.

[19] Josua Krause, Adam Perer, and Kenney Ng. “Interacting with predictions: Visual inspection of black-box machine learning models”. In: *Proceedings of the 2016 CHI conference on human factors in computing systems*. 2016, pp. 5686–5697.

[20] Gunnar Kudrjavets, Nachiappan Nagappan, and Thomas Ball. “Assessing the relationship between software assertions and faults: An empirical investigation”. In: *2006 17th International Symposium on Software Reliability Engineering*. IEEE. 2006, pp. 204–212.

[21] Robert C Martin, James Newkirk, and Robert S Koss. *Agile software development: principles, patterns, and practices*. Vol. 2. Prentice Hall Upper Saddle River, NJ, 2003.

[22] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[23] Robert Munro Monarch. *Human-in-the-Loop Machine Learning: Active learning and annotation for human-centered AI*. Simon and Schuster, 2021.

[24] Hugh Powell, Charles Ripper, and Cornell Lab. *Bird ID skills: Field marks*. Aug. 2015. URL: `https://www.allaboutbirds.org/news/bird-id-skills-field-marks/`.

[25] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why should i trust you?” Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[26] Matthew Richardson and Pedro Domingos. “Markov logic networks”. In: *Machine learning* 62.1-2 (2006), pp. 107–136.

[27] David S. Rosenblum. “A practical approach to programming with assertions”. In: *IEEE transactions on Software Engineering* 21.1 (1995), pp. 19–31.

[28] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps”. In: *arXiv preprint arXiv:1312.6034* (2013).

[29] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[30] Xingjiao Wu et al. “A Survey of Human-in-the-loop for Machine Learning”. In: *arXiv preprint arXiv:2108.00941* (2021).

[31] Rashid Zafar et al. “Big data: the NoSQL and RDBMS review”. In: *2016 International Conference on Information and Communication Technology (ICICTM)*. IEEE. 2016, pp. 120–126.

[32] Jie M Zhang et al. “Machine learning testing: Survey, landscapes and horizons”. In: *IEEE Transactions on Software Engineering* (2020).

[33] Yu Zhang et al. “A survey on neural network interpretability”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* (2021).