POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION

Institute of Computing Science

Bachelor's thesis

REAL-TIME ANIMAL MONITORING SYSTEM

Ewaryst Ławecki, 149591
Piotr Polus, 125950

Supervisor
dr inż. Anna Grocholewska-Czuryło

POZNAŃ 2024

Karta
pracy dyplomowej

Uczelnia:	Politechnika Poznańska	Profil studiów:	Ogólnoakademicki
Kierunek:	Informatyka	Forma studiów:	Niestacjonarne
Studia w zakresie:		Poziom studiów:	Pierwszego stopnia

Zobowiązuję/zobowiązujemy się samodzielnie wykonać pracę w zakresie wyspecyfikowanym niżej. Wszystkie elementy (m.in. rysunki, tabele, cytaty, programy komputerowe, urządzenia itp.), które zostaną wykorzystane w pracy, a nie będą mojego/naszego autorstwa będą w odpowiedni sposób zaznaczone i będzie podane źródło ich pochodzenia.

Jeżeli w wyniku realizacji pracy zostanie dokonany wynalazek, wzór użytkowy, wzór przemysłowy, znak towarowy, prawa do rozwiązań przysługiwać będą Politechnice Poznańskiej. Prawo to zostanie uregulowane odrębną umową.
Oświadczam, iż o wyniku prac wskazanych powyżej, a także o innych, w tym tych, które mogą być przedmiotem tajemnicy Politechniki Poznańskiej, niezwłocznie powiadomię promotora pracy.
Zobowiązuję się ponadto do zachowania w tajemnicy wszystkich informacji technicznych, technologicznych, organizacyjnych, uzyskanych w Politechnice Poznańskiej w okresie od daty rozpoczęcia realizacji prac do 5 lat od daty zakończenia wykonania prac.

	Imię i nazwisko	Nr albumu	Data i podpis
Student:	Ewaryst Ławecki	149591	
Student:	Piotr Polus	125950	

Tytuł pracy:	System rozpoznawania zwierząt w czasie rzeczywistym
Wersja angielska tytułu:	*Real-time Animal Monitoring System*
Dane wejściowe:	E. Freeman, E. Freeman, K. Sierra, B. Bates, *Head First Design Patterns*, Helion, 2017. Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd Edition*, O'Reilly Media, 2019
Zakres pracy:	1. Projekt i implementacja interfejsu użytkownika (ang. frontend). 2. Projekt i implementacja części serwerowej (ang. backend). 3. Projekt i implementacja algorytmu rozpoznawania zwierząt. 4. Wytrenowanie sieci neuronowej.
Termin oddania pracy:	30.09.2024
Promotor:	dr inż. Anna Grocholewska-Czuryło
Jednostka organizacyjna promotora:	Instytut Informatyki

podpis dyrektora/kierownika jednostki organizacyjnej promotora

data i podpis Dziekana

Contents

Abstract

The presented work describes the design and implementation of a system dedicated to real-time observation of wild animals on safari. In addition, the study includes a comparison of the performance of several machine learning models in the task of image recognition, depending on the architecture of the neural network.

Keywords: machine learning, motion detection, image classification, API, microservice, cloud, Amazon Web Services

Streszczenie

Przedstawiona praca opisuje projekt i realizację systemu dedykowanego do obserwacji dzikich zwierząt znajdujących się na safari w czasie rzeczywistym. Ponadto w pracy zawarto porównanie działania kilku modeli uczenia maszynowego w zadaniu rozpoznawania obrazów, w zależności od architektury sieci neuronowej.

Słowa kluczowe: uczenie maszynowe, detekcja ruchu, klasyfikacja obrazów, API, mikroserwis, chmura, Amazon Web Services

1. Introduction

1.1. About the system

In this work, a Real-Time Animal Monitoring System was designed, implemented, and evaluated. The system was created with the intention of assisting national parks, safari parks, and similar entities in collecting data on the animals present in a given area.

The operation of the system is as follows: cameras installed in the animal habitat area send video streams to a local server, which is responsible for detecting motion in the video stream. When motion is detected, a frame is sent to the cloud API, where animal recognition is performed and the recognition data is stored in a database. The collected data can then be viewed by users through a web interface.

In the project the most popular current technologies were used. It is very important because it brings two significant benefits: the ease of engaging new engineers in the project and the ease of solving various problems. Both of these benefits stem from the fact that popular technologies have large communities.

One of the most important non-functional requirements of the system is high scalability. This is made possible by the cloud, which can scale "to the sky" automatically with a pay-as-you-go pricing model, meaning that costs are incurred only for the resources used.

The work consists of the following chapters:

- **Chapter 1: Introduction** - which presents the concept of the system, a comparison with existing solution on the market, and the scope of the work that was carried out.
- **Chapter 2: Design guidelines** - which outlines the division of tasks in system development and the writing of the thesis. Functional and non-functional requirements that the system meets are also listed.
- **Chapter 3: Project structure** - which presents the system's architecture.
- **Chapter 4: Selected technology stack and tools** - which describes the programming languages, libraries, and tools used in the project.
- **Chapter 5: Image classification - recognizing animals from frames** - this chapter contains the research part of the work. It presents the theory of machine learning and convolutional neural networks and compares different neural networks in the task of image classification.
- **Chapters 6, 7, 8** - these chapters describe the remaining components of the system responsible for motion detection in the video stream, displaying data to the user, and data distribution.
- **Chapter 9: Conclusions** - contains a summary of the entire project, ideas for further system development, and general conclusions about the completed project.

1.2. Review of existing solutions

While devising the functionalities of the system and determining its operation, no inspiration was drawn from any existing systems of this type. The design relied entirely on original creativity. However, during internet research, one commercial solution was found that operates very similarly.

This solution is CVEDIA-RT [4]. It is an AI software platform designed for real-time video and image analysis. CVEDIA-RT leverages advanced neural networks to perform tasks such as object detection, classification, and tracking. The platform can process video feeds in real-time, identify and categorize various objects, and provide detailed analytics based on the detected data. CVEDIA-RT is highly scalable, allowing it to be deployed in a variety of environments, from small-scale systems to large enterprise solutions. It also integrates seamlessly with other systems, offering APIs for easy communication and data exchange.

CVEDIA-RT is, therefore, a very advanced solution with numerous functionalities. One of its modules is specifically designed for recognizing and classifying animals from videos or images [5]. The operating principle of this module is exactly the same as in the system described in this thesis, which is the utilization of deep neural networks.

1.3. Objective and scope of work

The aim of the project was to design and implement a system that possesses the minimum functionalities required to support national parks and nature reserves. The basic functions that meet these minimum requirements are:

- **Animal Recognition:** Using a trained neural network, the system performs animal recognition from video streams.
- **Data Analysis**: The system can analyze data such as recognized species, date and time of recognition, frame with the recognized animal, the number of recognitions of a given species, etc.
- **Displaying data to users in a Web application**: Logging into the application using a username and password.

By dividing the system into components, it was very easy for us to split the work between two people. This made the work proceed very smoothly, leaving plenty of time to test various solutions and choose the best ones.

2. Design guidelines

2.1. Division of tasks

The thesis was divided into the tasks shown in the table [Table 2.1.1] between the two authors. Green in the table means assignment to a specific author, blue one indicates participation. The main division included the implementation parts of the application (4 parts: API, Animal recognition algorithm, Motion detector, User interface) and the essay part of the engineering work (divided into chapters) [Table 2.1.2].

Tasks were split between the two authors of the overall work. The jira system for project management was used during the whole work (figure 2.1.1).

Table 2.1.1: Division of labour

TASK	Ewaryst	Piotr
Implementation of an application programming interface (API)		
Design and implementation of a neural network that recognizes images (Animal recognition algorithm)		
Implementation of motion recognition algorithm (Motion detector)		
Build the frontend of the application (User interface)		
Organizing the work of the team using Jira		
Testing the proper performance of the application		
Researching popular technologies and selecting the most suitable ones		
Researching and expanding knowledge in the field of artificial intelligence		
Gain and extend knowledge of HTML, CSS, JavaScript, React library		

Table 2.1.2: Division of labour for chapters

CHAPTER	Ewaryst	Piotr
1. Introduction	x	
2. Design guidelines		x
3. Project structure	x	
4. Selected technology stack and tools		x
5. Image classification - recognizing animals from frames	x	
6. Motion detection in the video stream		x
7. User Interface		x
8. API	x	
9. Conclusions	x	x

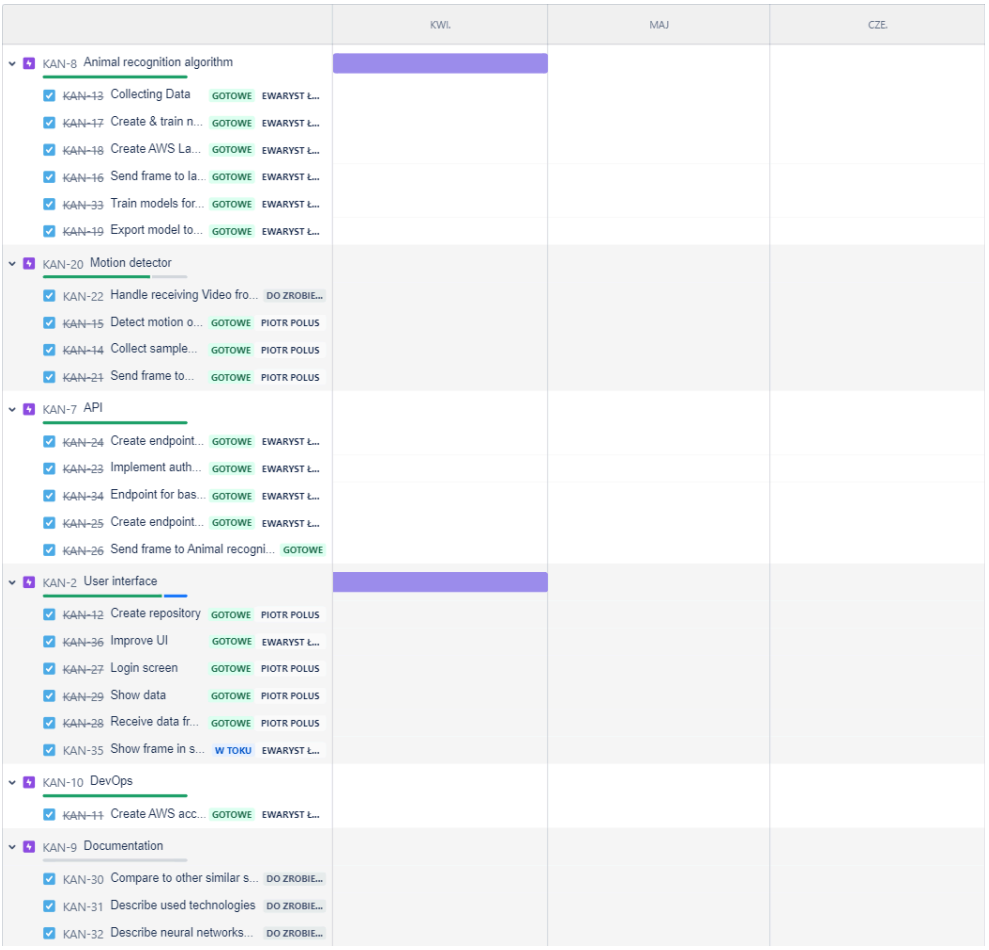


Figure 2.1.1: The screenshot from the Jira application. Project timeline

2.2. Functional requirements

The following functional requirements were distinguished in the thesis:

- Administrator is able to create a user account in the system.
- The application allows the user to log in using unique credentials: user email and password specified by admin.
- The user must be logged in to use the functionalities of the application.
- The user will not be logged out when the page is reloaded.
- The user can log out from the application.
- After logging into the application, the user is redirected to the home page, where the Dashboard is displayed.
- The user can view the data stored on the server by the application like the date and time of the animal spotted, the species of the animal identified, the degree of confidence in the model (accuracy) of the animal identified and the photo itself on which the model was operated.

2.3. Non functional requirements

The following non-functional requirements were distinguished in the thesis:

- System should be highly available therefore AWS services will be used to serve application and perform all backend computing.
- The user authentication function will be implemented by using the AWS Cognito.
- The data transferred between the application, the server and the neural network model will be in Base64 format.
- Application interface will be written using a TypeScript programming language and React library.
- The machine learning related parts of the system will be written using the Python language, using the well-known TensorFlow and Keras libraries.
- The following tools and services should be used to communicate between the application interface and the server: TypeScript, Serverless Framework, AWS Lambda, AWS API Gateway, AWS SQS, AWS S3.
- Data will be stored in AWS databases using Amazon S3 and DynamoDB services.

- The system correctly recognises all animal groups for which the neural network has been trained, at least 75% of accuracy for supported animal species.
- The application uses the best of the AI models created (evaluation based on testing).

3. Project structure

3.1. Microservices architecture

In software engineering, a microservice architecture is an architectural pattern that arranges an application as a collection of loosely coupled, fine-grained services, communicating through lightweight protocols [3].

This type of architecture has many advantages that make the system significantly better. But the most important ones are that it is possible to use more than one programming languages (Flexibility in Technology Stack) and engineering teams are able to work on their components separately (Team Autonomy) when a common data contract between components is defined.

In more details, microservices offer benefits like:

- **Flexibility in Technology Stack**: Different services can be built using different technologies, which allows teams to choose the best tool for the job,
- **Fault Isolation**: If one microservice fails, it does not necessarily bring down the entire system, which enhances overall system reliability,
- **Easier Maintenance and Updates**: Smaller, modular services are easier to understand, test, and update compared to a monolithic system,
- **Faster Time to Market**: Independent development and deployment enable faster iterations and quicker delivery of new features,
- **Enhanced Scalability**: Each service can be scaled independently to meet demand, which optimizes resource usage and performance,
- **Team Autonomy**: Smaller teams can own and manage individual microservices, leading to increased productivity and ownership,
- **Reuse and Composability**: Services can be reused across different parts of the application or even in different projects, promoting code reuse and composability,
- **Continuous Delivery and Deployment**: The ability to deploy changes to individual services without affecting the entire system supports continuous integration and delivery practices.

Obviously, it is not ideal architecture. It also has some disadvantages. The biggest drawback of using microservices architecture is the additional complexity of code and infrastructure. In practice, it significantly slows down the development process compared to the most simple architecture type like monolith. Because of that smaller and less complex projects should be designed rather in monolith / modular monolith architecture instead of microservices to save time and money.

The true benefits of microservices architecture start to emerge when a system gains more users, and with that, we need more computing power and engineers to maintain it.

3.2. Architecture diagram

The system is built in microservices architecture which means that it contains several independently operating components to increase the scalability of the solution and facilitate the easy division of work.

Determining the number of microservices and their responsibilities was done at the very beginning during the domain modeling stage. At that time, based on domain analysis, a decision was made that it would be optimal to operate in four different domains: motion detection, analysis of individual frames to detect animals, communication with database and presenting data to the user.

Figure 3.1.1 presents how microservices communicate with each other using protocols (like HTTP and RTSP) or message queue (SQS).

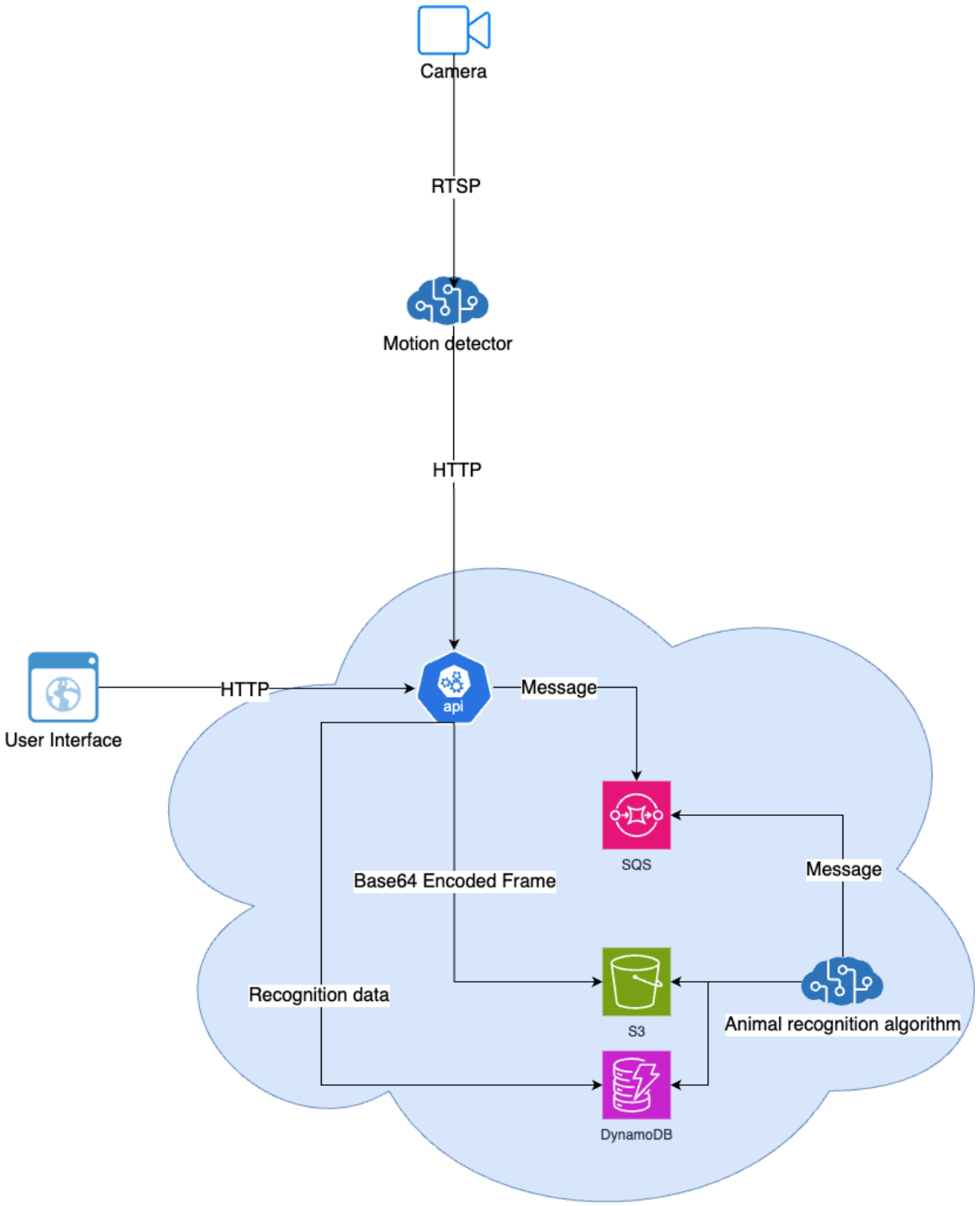


Figure 3.1.1: Architecture diagram of Real-time Animal Monitoring System

First element on the architecture diagram is the Camera. It is the main source of data for our system which produces video stream.

First microservice in the system is Motion Detector which is responsible for receiving data from cameras via RTSP protocol and analyzing video in order to detect motion. When motion is detected then Motion Detector sends frames to the API within HTTP protocol.

API is the central microservice of our system. It is responsible for communicating with other microservices, which involves delivering data to them and receiving data from them. To ensure the reliability of our system and ensure that each frame sent to the API is processed at least once between API and the Animal Recognition Algorithm, a queue was used (AWS SQS). When the API receives a frame from Motion Detector then it puts a message to the queue. The API stores the frames encoded in base64 in a bucket on AWS S3.

The Animal Recognition Algorithm is a microservice responsible for analyzing stored frames and assigning them to one of the classes in other words performing the task of image classification. This microservice continuously pools the queue for new messages to process. After assigning a frame to a class, it saves information such as the class name, confidence level, and timestamp to the database (DynamoDB).

The final microservice is the User Interface, which allows the user to login and view the data collected by the system, such as the saved frame, the animal that was recognized, the confidence level, the timestamp, and other relevant information.

4. Selected technology stack and tools

4.1. Programming languages

4.1.1. Python

As written on the official **Python** documentation page, this programming language is very easy to learn because of its simple syntax. It has efficient high-level data structures, dynamic typing and an effective approach to object-oriented programming despite its simplicity. This makes Python "*an ideal language for scripting and rapid application development in many areas on most platforms*" [21].

In addition, the libraries used in the thesis (4.2.1) are based on this language. It is possible to find full documentation and a variety of other sources to help with projects like this one because Python is a very popular programming language and it has a huge community. These are some of the multiple reasons for choosing this technology.

4.1.2. TypeScript

TypeScript is an open-source programming language created by Microsoft as a superset of JavaScript. This means that any program written using this language is a valid JavaScript program. Applications written in TypeScript compile directly into JavaScript code. It is a relatively modern programming language, having been made available to the public in 2012.

TypeScript, unlike the previously mentioned language, is static and strongly typed [24]. It means that TypeScript allows developers to catch most of the errors at compile time rather than at run time. It is really helpful with reducing bugs and improving code quality [19][20].

The advantages of this language and the fact that one of the co-authors uses this tool in his daily work made it a perfect fit for this project.

4.2. Libraries

4.2.1. Tensorflow and Keras

TensorFlow is an open source machine learning and artificial intelligence library developed by Google Brain Team [32]. It is used to create and train neural network models, deep learning and other machine learning algorithms. The library is used in a variety of applications such as image recognition, natural language processing and predictive analytics. TensorFlow is available for a variety of programming languages, including Python, making it easy to integrate into existing projects. Due to its flexibility and scalability, it has become a more and more popular tool among data scientists and engineers [15][16].

Keras is an open source library for creating and training machine learning models, acting as a high-level interface to libraries such as TensorFlow [33]. These two tools are strongly linked in ML applications, as they are constantly updated and supported together. Keras is designed to be easy to use, flexible and modular,

enabling rapid prototyping of models. Keras allows users to easily build and train deep neural networks using an intuitive and concise API [35]. It supports both convolutional and recurrent networks, as well as combining them into hybrid models. Thanks to its simplicity and performance, Keras is widely used by beginners and currently, by the rapid development of Artificial Intelligence, often also by experienced machine learning professionals [17][18].

As stated on the official website, keras provides:

- Simplicity - to be easy to understand and use, so that it can be used by everyone and focus on the important problems rather than the base,
- Flexibility and efficiency - very efficient and well scalable tool, validated by deployment in large organizations and big projects [17].

4.2.2. React

React is a popular JavaScript library developed by Facebook, used to build user interfaces, especially for single-page applications (SPA). It works on a component basis, allowing complex interfaces to be created from smaller, self-contained components. Components in React can be functional or class-based, and each can manage its own state. This language uses the Virtual DOM (**DOM** - *Document Object Model* - a way of representing an HTML or XML document as a tree structure wherein each node is an object representing a part of the document. Cross-platform and programming language independent [36]), which improves performance by minimizing direct operations on the real DOM. JSX, or JavaScript XML, allows code to be written that looks like HTML, making it easier to create interfaces. React has extensive community support and a wide ecosystem of tools and libraries, making it powerful and flexible [13][14].

The advantages of using the React library are:

- Performance: Thanks to the virtual DOM, React minimizes operations on the real DOM, which increases rendering speed,
- Component-ability: Enables modular and reusable components, which speeds up development and makes code easier to maintain,
- JSX: Makes it easier to write user interface code by enabling the use of HTML-like syntax in JavaScript,
- Hooks: Makes it easier to manage state and side effects in function components, simplifies code,
- Community support: A large developer community provides support, rich documentation and many additional libraries,
- Tool environment: A tool, such as Create React App (used in the work), makes it easy to set up and get started with React.

4.3. AWS (Amazon Web Services)

Cloud computing is a model for delivering computing services such as data processing, storage, networking, and software over the internet. These services are provided by external vendors, known as cloud providers, who offer their resources on a pay-as-you-go basis, allowing for flexible scaling of resources based on user needs [30].

One of the main advantages of cloud computing is its flexibility – users can quickly increase or decrease the amount of resources they use in response to changing demands. This eliminates the need for costly investments in IT infrastructure and allows companies to focus on their core business activities. Additionally, cloud services offer high levels of availability and reliability, as cloud providers ensure redundant systems and disaster recovery mechanisms.

Cloud computing can be divided into three main service models [30]:

- **Infrastructure as a Service** (*IaaS*) - provides basic IT resources such as servers, storage, and networks, on which users can build and manage their applications.
- **Platform as a Service** (*PaaS*) - offers ready-to-use platforms and development environments that simplify the development and deployment of applications.
- **Software as a Service** (*SaaS*) - delivers ready-to-use software applications, such as email, CRM, or content management systems, accessible through a web browser.

Using the cloud can also enhance data security, as cloud providers invest in advanced protection mechanisms and regular system updates. Companies can also benefit from the global availability of cloud services, enabling them to quickly enter new markets and serve customers worldwide.

Cloud computing supports innovation by allowing companies to experiment with new technologies and solutions without large capital investments. By accessing a wide range of advanced tools and services, businesses can quickly deploy new products and services, gaining a competitive edge in the market.

Amazon Web Services (AWS) is a comprehensive and widely adopted cloud platform offered by Amazon, providing over 200 fully featured services from data centers globally [28]. AWS offers a broad range of services, including computing power, storage options, and networking, which enable businesses to scale and innovate quickly and cost-effectively [31].

With its pay-as-you-go pricing model, AWS allows organizations to only pay for the resources they use, reducing overall IT costs. AWS is renowned for its high

levels of security, offering robust protection measures and compliance certifications to safeguard data and applications. Additionally, AWS provides extensive support and documentation, making it accessible for businesses of all sizes to leverage cloud computing for their operations [29] [31].

In the project a number of services provided by AWS have been used, and these include:

- Amazon API Gateway,
- AWS Lambda,
- Amazon Simple Storage Service (Amazon S3),
- Amazon Simple Queue Service (Amazon SQS) [27].

For a more detailed description see section 8.2.

4.4. GitHub

Git is a distributed version control system that allows follow changes in project's source code. All the files of a particular project and the full history of changes are stored in the corresponding repository. Git is based on the GNU (General Public License) [8].

The main features of git include [6]:

- **Distributed version control** - each repository clone contains the full history of the project, including all changes. This ensures that, even if the central server fails, data is not lost because copies exist on multiple computers. In addition, it is possible to work on the project offline, without a current connection to the server.
- **Tracking of changes** - all changes made to files in the project are tracked continuously and give developers the ability to monitor the history of modifications. Each change is saved as a commit with a unique identifier, allowing for easy review and restoration of previous versions of the code.
- **Branching and merging code versions** - it is possible to create branches, which allow different functionalities to be developed in parallel without affecting the main version of the code. The merge process (merging) allows changes from different branches to be combined, allowing new features or fixes to be integrated into the main code.

GitHub is a Git-based online platform for managing software developer projects. It allows developers to share, store and manage their code. They can

collaborate on code remotely. GitHub offers features such as pull requests, code reviews and issues to help manage and develop projects. In addition, this hosting platform offers integrations with various CI/CD tools and process automation to help manage the software development life cycle efficiently [7].

The study uses this software with all its advantages so that the authors do not have to work together from one place and at one time, and each developer works on the assigned tasks at a convenient time.

4.5. Jira

Jira is a tool for project management used to plan and track work across every team. Software allows track bugs, issues and using an agile approach to project management [25].

Originally, Jira was an open source tool available for anyone created by the Atlassian company. Application designed for use in small teams or individually have grown in popularity and number of users and have been adopted by many companies (not only those working on code). As a result, the company decided to create paid versions [26].

In the thesis, Jira was used in a free, basic version. Nevertheless, it helped the authors to:

- plan and organise tasks,
- align work to authors,
- mutual tracking of work progress.

5. Image classification - recognizing animals from frames

5.1. Machine learning

Machine learning (ML) is a branch of artificial intelligence (AI) and computer science that focuses on using data and algorithms to enable AI to imitate the way that humans learn, gradually improving its accuracy [34][38].

Generally speaking, machine learning consists of three main parts:

- Decision process - machine learning algorithms are primarily employed to predict outcomes or classify data. They analyze input data, whether labeled (with known outputs) or unlabeled (without predefined outcomes), to identify patterns and produce an estimate based on that data.
- Error function - to gauge the model's accuracy, an error function is utilized. This function compares the model's predictions against known results (if available) to measure how far off the predictions are from the actual values.
- Model optimization process (learning process) - the optimization process involves adjusting the model to improve its accuracy. By iteratively tweaking parameters (e.g., weights) and minimizing the error between predicted and actual outcomes, the algorithm refines itself until it reaches a satisfactory accuracy level. This cycle of evaluation and optimization continues until the model converges on an optimal solution.

Machine learning methods are divided into two main categories:

- Supervised learning - this method relies on labeled data. It means that for each training example the correct answer is known. Then the model’s goal is to discover and learn the rules to predict these answers for new, unseen previously data.
- Unsupervised learning - this method uses data without labels in the learning process. The model’s goal is to discover hidden patterns or structures in the data.

The most popular machine learning algorithms:

- neural networks,
- linear regression,
- logistic regression,
- clustering,
- decision trees,

- random forests.

Machine learning is widely used today. Here are a few examples of its applications:

- computer vision,
- speech recognition,
- recommendation engines,
- language models (LLM),
- fraud detection,
- autonomous vehicles.

5.2. Artificial neural network

A neural network is a machine learning program, or model, that makes decisions in a manner similar to the human brain, by using processes that mimic the way biological neurons work together to identify phenomena, weigh options and arrive at conclusions [39].

The functioning principle of an artificial neural network (ANN) is inspired by the way the human brain operates. Similar to how biological neurons in the brain process information by receiving inputs, processing them, and then transmitting outputs, artificial neurons in a neural network perform similar tasks.

As it is presented in figure 5.2.1 an artificial neural network consists of a layers of nodes (artificial neurons). First layer is called the input layer. Next there is one or more hidden layers and at the end output layer.

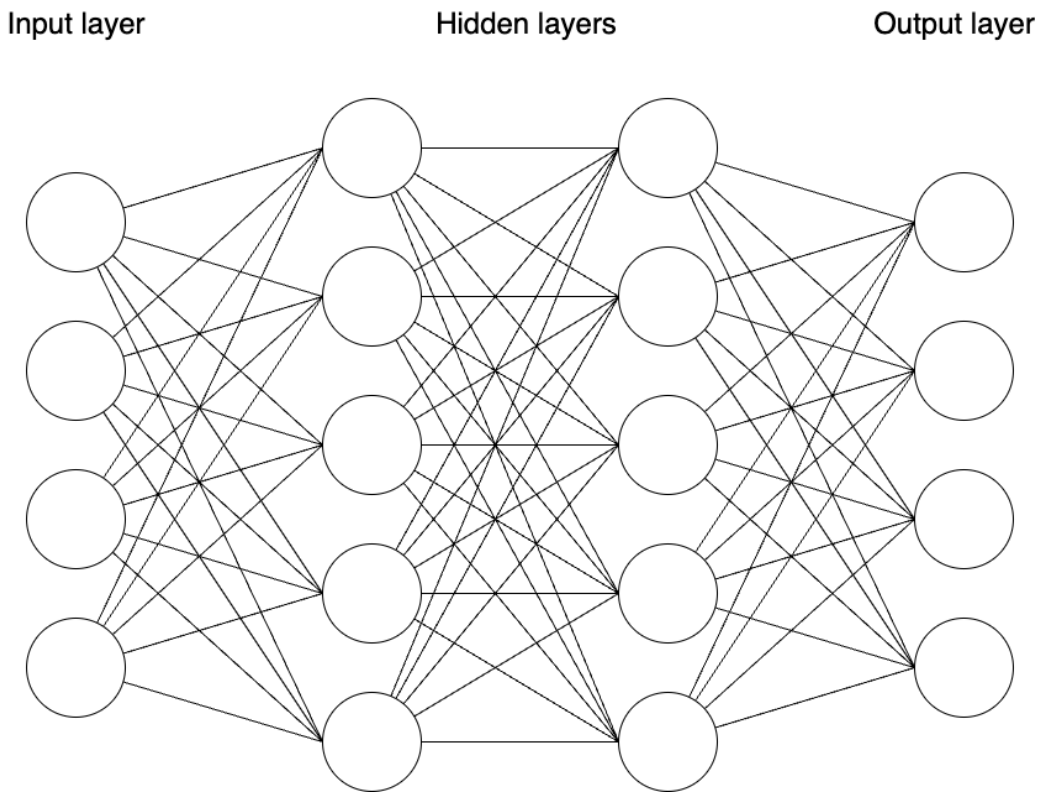


Figure 5.2.1: Neural networks architecture [40]

An artificial neuron consists of three main components:

- Inputs - these receive signals, which are typically the outputs from other neurons in previous layers. Each input is associated with a weight that determines its influence on the neuron's output.
- Neuron body (processing unit) - the body of the neuron computes the output based on the input values, their respective weights and bias value. The weighted inputs are summed, bias is added to the sum and then an activation function is applied to determine the neuron's final output.
- Output - the processed value (output) of the neuron is passed on to neurons in the next layer of the network.

The structure of an artificial neuron is illustrated in figure 5.2.2.

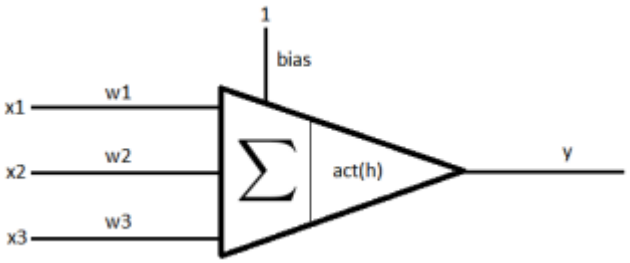


Figure 5.2.2: Structure of artificial neuron

Thus, the formula for the output of a single neuron is as follows:

$$y = f\left(b + \sum_{i=1}^{n} x_i \cdot w_i\right)$$

where:

y - output of neuron

f - activation function

b - bias

x_i - value of input indexed by i

w_i - weight of input indexed by i

n - amount of inputs

As an activation function vary functions might be used but the most popular are:

- Sigmoid,
- Hyperbolic Tangent (Tanh),
- Rectified Linear Unit (ReLU),
- Leaky ReLU,
- Softmax Function.

As mentioned above, neural networks are built from neurons arranged in layers. Therefore, once an appropriate neural network architecture is selected, the training process can begin.

To begin this process, it is required to have a training set D, consisting of n pairs (x_i, t_i), where x represents the network inputs and t represents the desired outputs. As can be observed, this is a type of supervised machine learning because the desired output of the model for the training example is known.

Learning process consists of a few steps, the first of them is **initialization**. In this process weights and biases are initialized with random values. These are the

parameters that will be adjusted during training. Also one constant parameter which needs to be initialized at the very beginning is learning rate(α).

Next step is **forward propagation**. Now neurons in input layer receive values x_1, x_2, x_3 … x_n (often called “features”). Each neuron in the input layer computes its output according to the formula described above and forwards it to the neurons in the next layer. When this process is done, error of the neural network might be computed.

So **error calculation** is the next step in the learning process. Formula for the error of a single neuron will be used, which is presented as follows:

$$E = \frac{1}{2}(t - y)^2 = \frac{1}{2}\delta^2$$

where y is network output value for given input and t is desired output. Difference between t and y is marking as a δ (delta).
When the error is calculated and it is greater than zero, the *gradient descent algorithm* is applied to update the weights in the network, thereby minimizing the network's error. According to that algorithm each weight must be modified by:

$$\Delta w_i = -\alpha\frac{\partial E}{\partial w_i}$$

where α the learning rate and is an important parameter set at the beginning of the algorithm. It dictates how much the weights will be modified in a single learning step. Next part of the formula is partial derivative which is used to estimating the contribution of the weight w_i to the error E which is a function of all weights of the neuron.
After appropriate transformations, the formula can be written as:

$$\Delta w_i = \alpha \cdot \delta \cdot x_i \cdot \frac{\partial y}{\partial h}$$

where h is the sum of input values multiplied by weights. This formula is called the **delta rule.**

As can be observed, these formulas allow us to calculate the errors only for the neurons in the output layer. For neurons in the hidden layers, the **backpropagation algorithm** is used. To update weight in hidden layers δ needs to be computed in a different way. The error δ of neuron N in the hidden layer should be taken as the weighted sum of the errors of the neurons in the next layer, to which neuron N transmitted its activation. This sum is weighted by the connection weights between the neurons. It means that if neuron N transmitted activation to a neuron M in the next layer with a large weight, it had a significant contribution to the potential error made by that neuron.

The process of forward propagation, error calculation, and weight updating in the network using the backpropagation algorithm is repeated until a sufficiently small error is achieved.

5.3. Deep learning

The terms "deep learning" and "neural networks" are often used interchangeably, which can lead to confusion. Therefore, it is important to clarify that the "deep" in "deep learning" simply refers to the number of layers in a neural network. A neural network that consists of more than three layers (including the input and output layers) can be considered a deep learning algorithm. A neural network with only two or three layers is merely a basic neural network.

5.4. Convolutional neural network

There are various types of neural networks which are used for different purposes. The perceptron is the oldest one, created by Frank Rosenblatt in 1958 [39]. Besides perceptron and multilayer perceptron neural network there are other types and two the most popular of them are:

- Convolutional neural networks (CNNs),
- Recurrent neural networks (RNNs).

This chapter will be focused on CNN which is a great choice when it comes to image recognition, pattern recognition and computer vision problems. Therefore, in this project, a convolutional neural network was used.

Convolutional neural networks mainly consists of three types of layers:

- Convolutional layer,
- Pooling layer,
- Fully-connected layer.

The convolutional layer, as the name implies, carries out a mathematical operation known as convolution on a grid of pixels that make up an image. The sliding window that is applied to this grid is referred to as a kernel or filter. Typically there are several filters of equal size which are applied and each filter is responsible for recognizing a specific pattern from the image. For example the curving of the objects, the edges or even the whole shape of the objects.
Image f convolution with filter ω can be described as follows(it is discrete operation):

$$g(x,\ y) = \omega * f(x,\ y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,\ dy) f(x + dx,\ y + dy)$$

where:

- $g(x, y)$ is filtered image
- $f(x, y)$ is original image
- ω is the filter kernel

Every element of the kernel is considered by $-a \leq dx \leq a$ and $-b \leq dy \leq b$.

Convolution operation can be described also as a linear combination:

$$w^T x = \sum_{i-1}^{n} w_i x_i$$

where:

- x is a vector of pixels
- w is a vector of filter values

The convolution operation involves sliding the filter (kernel) across the image and computing the dot product between the filter and the corresponding section of the image.

The result of the convolution operation across the entire image is stored in a feature map. Each value in the feature map represents how much the specific feature detected by the filter is present at that location in the image. The dimensions of the feature map are smaller than the original image because the filter slides over the image, reducing its size.

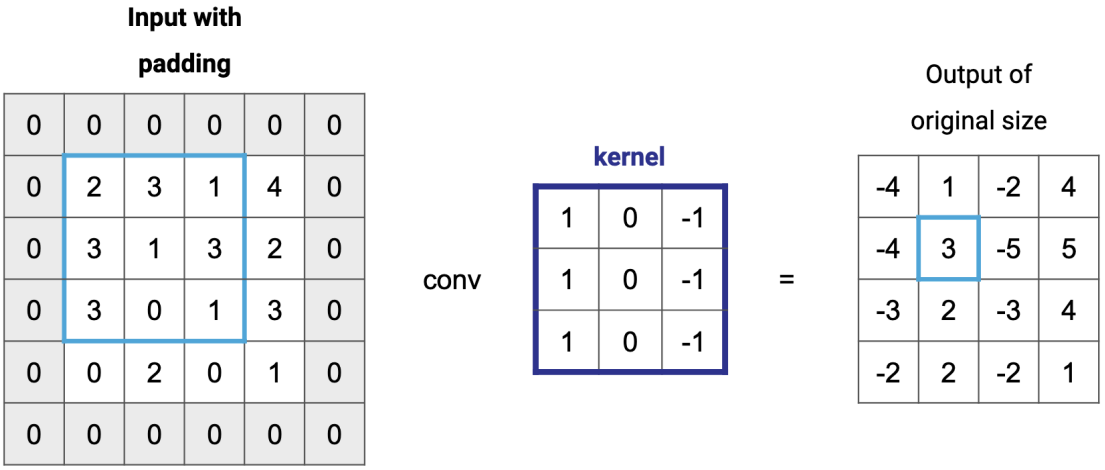


Figure 5.4.1: Convolution performed on matrix of pixels [41]

In the example one the figure 5.4.1 there is also padding added to the original matrix in order to keep the processed matrix in the same size. It is an example of 2D convolution, it might be used when the image is in grayscale. In the convolutional neural networks image is usually represented as a matrix of pixels in shape n x m x 3 because there are 3 RGB canals therefore to process it the filter must be in shape like p x r x 3.

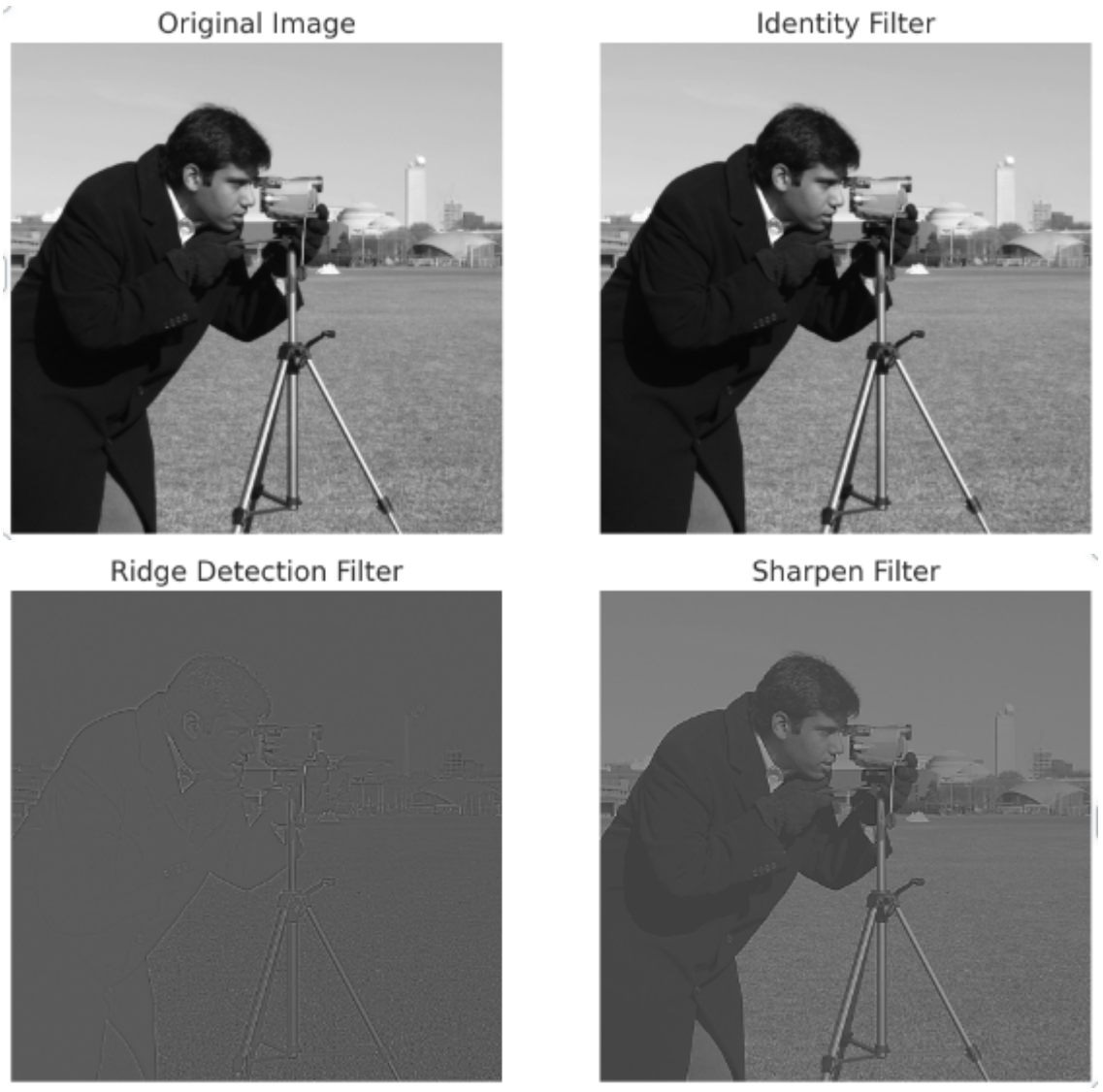


Figure 5.4.2: Example of applying filters to an image

In above example(figure 5.4.2) following filters have been applied to original image:

Identity

0	0	0
0	1	0
0	0	0

Ridge detection

0	-1	0
-1	4	-1
0	-1	0

Sharpen

0	-1	0
-1	5	-1
0	-1	0

A CNN typically uses multiple filters to detect different features in the input data and then each filter creates its own activation map. For example if the image is in shape 32x32x3 and CNN uses 6 filters 5x5x3 then output will be in shape 28x28x6. As in common neural networks described in previous section activation function is also applied for each value in the output matrix. A very important point is that the **values of the filters are determined during the training process of the network**. We treat them as parameters (weights) of the convolutional network. As a loss function in CNN *cross entropy function* is used very often:

$$L = -\sum_{i=1}^{K} y_i ln(\widehat{y}_i)$$

where:

K - number of classes

y_i - desired response for the class i, $y_i \in \{0, 1\}$

$\widehat{y}_i$ - received response for class i, $\widehat{y}_i \in [0, 1]$

Each filter will learn to recognize a different aspect of the image (e.g., vertical edges, horizontal edges, corners, textures). Filters are learned during the training process through backpropagation.

In CNNs, in addition to convolutional layers, following layers are also very often uses:

- Max / Min Pooling layer - a window of size n x m is slid over the image, and the maximum / minimum value is extracted. This operation reduces the size of an image. It is usually used on activation maps.
- Fully connected (Dense) layer - it is used in the final stages of the network to combine features extracted by convolutional and pooling layers into a final classification or regression output.
- Flatten layer - it is used to convert the multi-dimensional input (e.g., a feature map) into a one-dimensional vector, which is then fed into fully connected layers.

- Dropout layer - it helps prevent overfitting by randomly setting a fraction of input units to zero during training, which forces the network to learn more robust features.

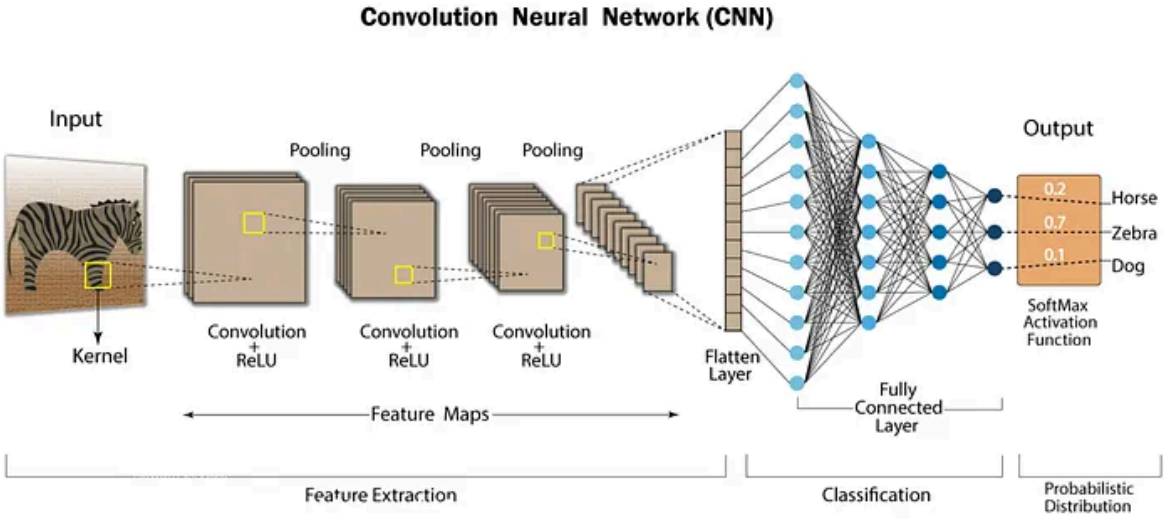


Figure 5.4.3: Example CNN architecture [42]

In the example in figure 5.4.3, we see the architecture of a CNN. As can be observed, the first part of the network utilizes Convolution and Pooling layers to extract features from the image by creating feature maps. Then, all the resulting feature maps are flattened into a vector in the Flatten layer.

The second part of the network is responsible for classifying the image into one of the classes. This is the previously discussed multilayer perceptron in its Fully Connected variant.

The third and final part of the network converts the classifier's output into probabilities of the image belonging to each class. Softmax function is used to achieve it.

5.5. Architecture of used neural network

The architecture of neural network used in the system is presented in the following screenshot showing the code in Python (figure 5.5.1):

```
64  # Creating a model
65  model = Sequential([
66    data_augmentation,
67    layers.Rescaling(1./255, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
68    layers.Conv2D(32, 3, padding='same', activation='relu'),
69    layers.MaxPooling2D(),
70    layers.Conv2D(64, 3, padding='same', activation='relu'),
71    layers.MaxPooling2D(),
72    layers.Conv2D(128, 3, padding='same', activation='relu'),
73    layers.MaxPooling2D(),
74    layers.Dropout(0.2),
75    layers.Flatten(),
76    layers.Dense(128, activation='relu'),
77    layers.Dense(num_classes, name="outputs")
78  ])
```

Figure 5.5.1: CNN architecture used in project

Model is defined using the *Sequential* class in Keras. Here is its composition and a description of the all used layers:

- As we can see, a first layer *data_augmentation* variable has been used (figure 5.5.2).

```
52  # For preventing overfitting
53  data_augmentation = keras.Sequential(
54    [
55      layers.RandomFlip("horizontal",
56                        input_shape=(IMG_HEIGHT,
57                                     IMG_WIDTH,
58                                     3)),
59      layers.RandomRotation(0.1),
60      layers.RandomZoom(0.1),
61    ]
62  )
```

Figure 5.5.2: *data_augmentation* variable in the code

Data augmentation is the operation of increasing the amount of training data by generating new examples based on existing training data, such as rotating, shifting, or scaling images. This is useful for preventing overfitting.

- *Rescalling* layer normalizes the pixel values of the image to a range from 0 to 1. RGB images have pixels in the range of 0 to 255, so the rescaling divides

each pixel by 255. Argument *input_shape* expects images with dimensions (IMG_HEIGHT, IMG_WIDTH, 3), where 3 refers to the number of channels (RGB).

- *Conv2D* is a convolutional layer when the first argument is a number of filters, the second argument is a size of filter. Argument named *padding* with value *'same'* means that image will keep its original size after convolution. In each convolutional layer ReLu has been used as an activation function.
- *MaxPooling2D* is a pooling layer which decreases the size of an image. It performs max pooling operations in the 2x2 region.
- *Dropout* is a layer which randomly "turns off" 20% of the neurons during training (as 0.2 value as an argument indicates). It helps to prevent overfitting because the network does not become too dependent on specific neurons.
- *Flatten* layer as name indicates flattens the output data from previous layer into one-dimensional vector. So then it can be passed to the fully connected layer.
- *Dense* is a fully connected layer where the first argument is a number of neurons. In the first *Dense* layer ReLu has been used as an activation function. In the second one no activation function has been used because it is the last layer which has the number of neurons equal to the number of classes. It is used to classify output data by using the Softmax function.

After compiling and learning the model in the console appears a summary of the trained network(figure 5.5.3). In this case the neural network has 8 024 006 trainable parameters and 16 048 014 optimizer parameters which gives 24 072 020 parameters in total. Note that *IMAGE_WIDTH* and *IMAGE_HEIGHT* for that training were set to 180.

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 180, 180, 3)	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 32)	896
max_pooling2d (MaxPooling2D)	(None, 90, 90, 32)	0
conv2d_1 (Conv2D)	(None, 90, 90, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 64)	0
conv2d_2 (Conv2D)	(None, 45, 45, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 128)	0
dropout (Dropout)	(None, 22, 22, 128)	0
flatten (Flatten)	(None, 61952)	0
dense (Dense)	(None, 128)	7,929,984
outputs (Dense)	(None, 6)	774

Total params: 24,072,020 (91.83 MB)
Trainable params: 8,024,006 (30.61 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 16,048,014 (61.22 MB)

Figure 5.5.3: CNN neural network layer with number of parameters

5.6. Learning process and data

The entire process of training a Convolutional Neural Network (CNN) from the code level consists of several parts:

- Creating datasets - training and validation(figure 5.6.1).

```
15  # Load data
16  train_ds = tf.keras.utils.image_dataset_from_directory(
17    DATA_PATH,
18    validation_split=0.2,
19    subset="training",
20    seed=123,
21    image_size=(IMG_HEIGHT, IMG_WIDTH),
22    batch_size=BATCH_SIZE)
23
24  val_ds = tf.keras.utils.image_dataset_from_directory(
25    DATA_PATH,
26    validation_split=0.2,
27    subset="validation",
28    seed=123,
29    image_size=(IMG_HEIGHT, IMG_WIDTH),
30    batch_size=BATCH_SIZE)
```

Figure 5.6.1: Creating training and validation data sets

Training and validation sets are subsets of the data used for the learning process.

Training set is used to train models. It means that the model adjusts its weights based on the errors it makes on the training set. This process is guided by the optimizer and loss function. In this case the training set is 80% of the all images in DATA_PATH localization.

On the other hand, a validation set is used to evaluate the performance of the model during the training. It is very important that the validation set and training set should not include the same images. Data from the validation set provides an unbiased assessment of how well the model generalizes to new, unseen data. It also helps in tuning hyperparameters and prevents overfitting. In this case the validation set is 20% of the all images in DATA_PATH localization.

- Building and compiling the model.

After building the model shown in figure 5.5.1, the next step which needs to be executed is compiling the model (figure 5.6.2).

```
80    # Compile model
81    model.compile(optimizer='adam',
82                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
83                  metrics=['accuracy'])
84
```

Figure 5.6.2: Compiling the model

As an optimizer for loss function Adam has been used which is one of the most popular and efficient optimization algorithm. Adam is an acronym for Adaptive Moment Estimation. In short, the principle of this algorithm's operation is customization of each parameter's learning rate based on its gradient history, and this adjustment helps the neural network learn efficiently as a whole [43].

Next parameter is *loss* which indicates which loss function should be used. In this case Sparse Categorical Crossentropy has been used. It is a loss function used in multiclass classification problems and when labels are encoded as integers rather than one-hot encoded. It is more efficient to use integers when large numbers of classes are in data. Parameter *from_logits* means that network output is not normalized (not probabilities) so the loss function will apply softmax function internally before calculating cross-entropy loss. Accuracy is a metric that measures how often the model's predictions match the actual labels and it will be reported in console during training.

- Creating an instance of the EarlyStopping class (figure 5.6.3).

EarlyStopping is a technique used to stop training a neural network once the performance stops improving. It is used in model training to avoid overfitting.

```
85    # Early Stopping callback
86    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

Figure 5.6.3: Creating *early_stopping* callback

Following parameters have been passed to EarlyStopping class constructor:

- *monitor='val_loss'* - specifies what metric the early stopping will monitor. In this case training will be finished when validation loss stops decreasing,
- *patience=5* - specifies how many epoch the model will wait for improvement before stopping the training process,
- *restore_best_weights=True* - this flag tells the model that after stopping the best weights observed during training should be restored(weights when validation loss value was the lowest).

- Training the model (figure 5.6.4).

```
88    # Training
89    history = model.fit(
90      train_ds,
91      validation_data=val_ds,
92      epochs=EPOCHS,
93      callbacks=[early_stopping]
```

Figure 5.6.4: Starting training process of the model

Next step is to simply start the training process using the *model.fit function*. As a parameters training data set and validation data have been passed. Also it is required to specify how many epochs of learning will function perform. Lastly, an *early_stopping* callback has been passed to function to use EarlyStopping technique in the learning process.

- Displaying summary plots of the training process (figure 5.6.5).

```
96    # Show model summary
97    model.summary()
```

Figure 5.6.5: Showing summary of the trained model

Model summary is a table which was shown in the previous section in figure 5.5.3.

- Saving the model to disk.

```
122   # Save model
123   model.save(os.path.join(MODELS_PATH,'model.keras'))
```

Figure 5.6.6: Saving trained model to the disk

To use a model in the future, it is possible to save it to disk. In Keras *save* method might be used to achieve that.

5.7. Testing trained model

To select the appropriate neural network parameters, tests were conducted on various variants, and the best-performing model was chosen based on the results. The variables in the tested variants included the number of filters in the convolutional layers, the use of data augmentation, and the application of the dropout layer.

For the training process, we used data obtained from Kaggle. The amount of data for each class was as follows:

- elephant: 2995,
- giraffe: 695,
- lion: 1410,
- parrot: 2554,
- rhino: 1779,
- tiger: 1391.

Before starting model training, a significant challenge was filtering out images that did not fit our task. For example, images showing only parts of animal bodies instead of whole animals, animal figurines, animated representations of animals.

For testing the trained models, a test set was created by selecting 100 images for each class. Then, tests were conducted on the test set for each model.

Learning processes and test results for various variant of neural network are presented below:

- **Variant 1**: the number of filters in successive convolutional layers: 32 / 64 / 128, data augmentation and dropout was used:

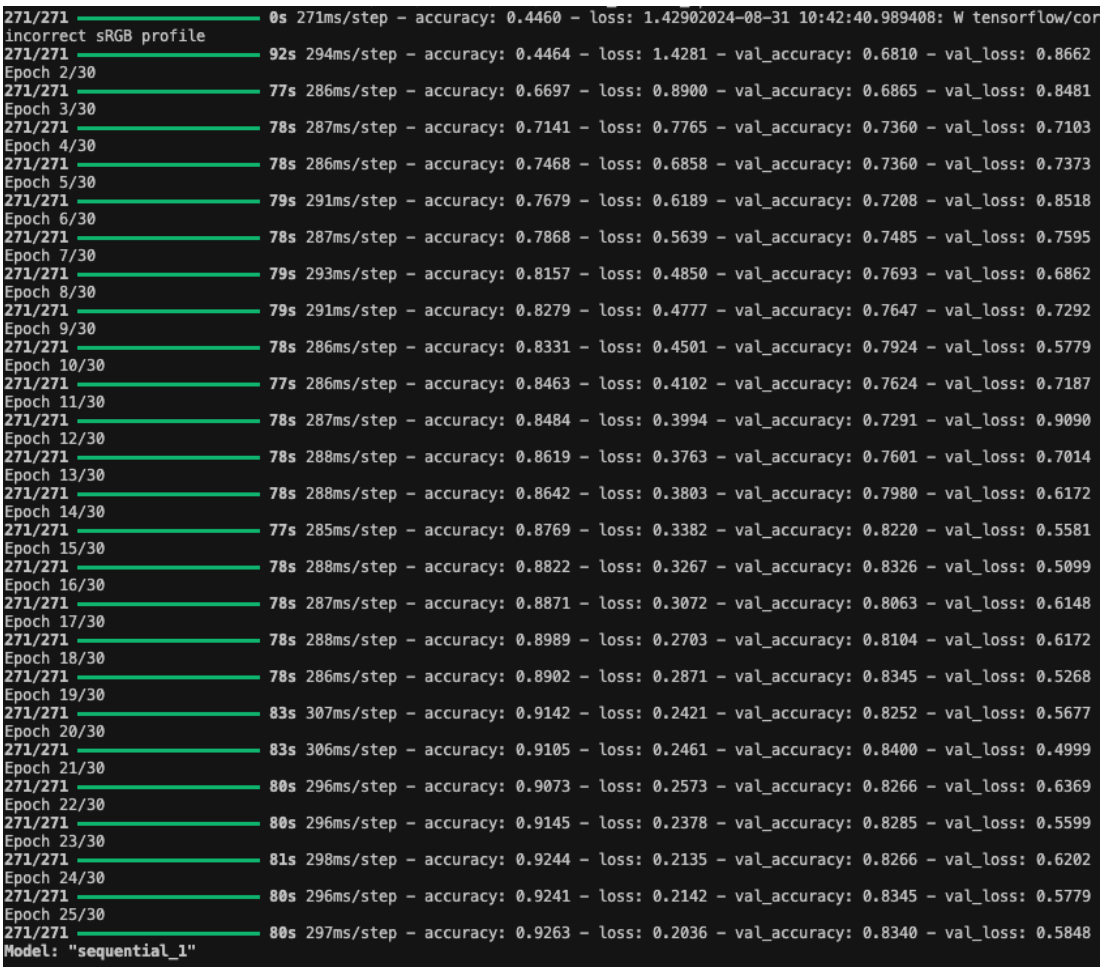


Figure 5.7.1: Training model process for Variant 1

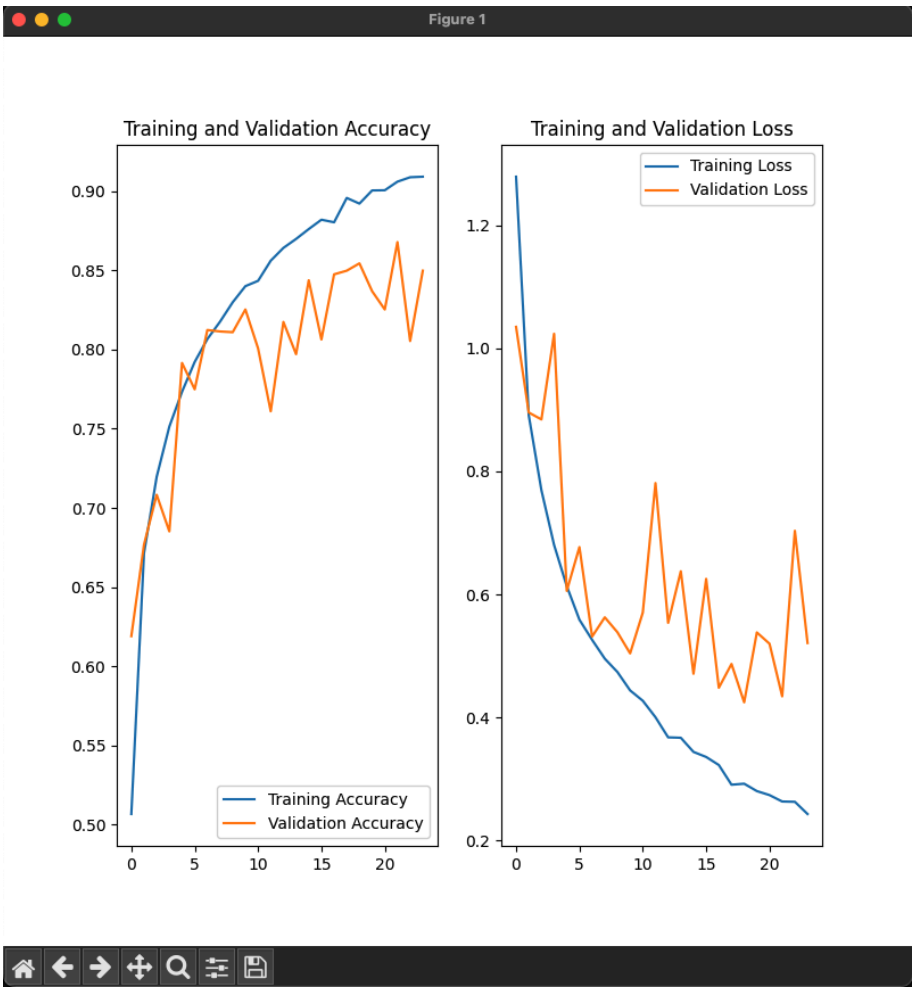


Figure 5.7.2: Training model results for Variant 1

Table 5.7.1: Correctly classified images for Variant 1

Class name	Correctly classified images [%]
elephant	75
giraffe	77
lion	73
parrot	90
rhino	68
tiger	80

- **Variant 2**: the number of filters in successive convolutional layers: 128 / 64 / 32, data augmentation and dropout was used.

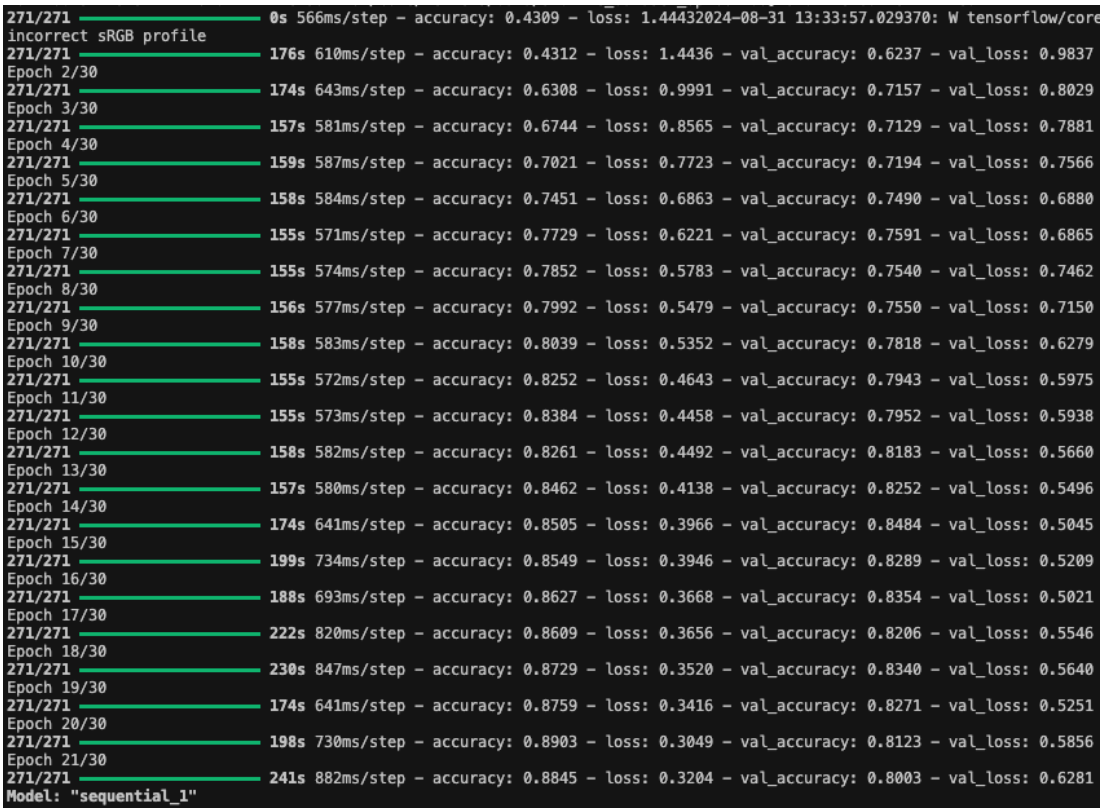


Figure 5.7.3: Training model process for Variant 2

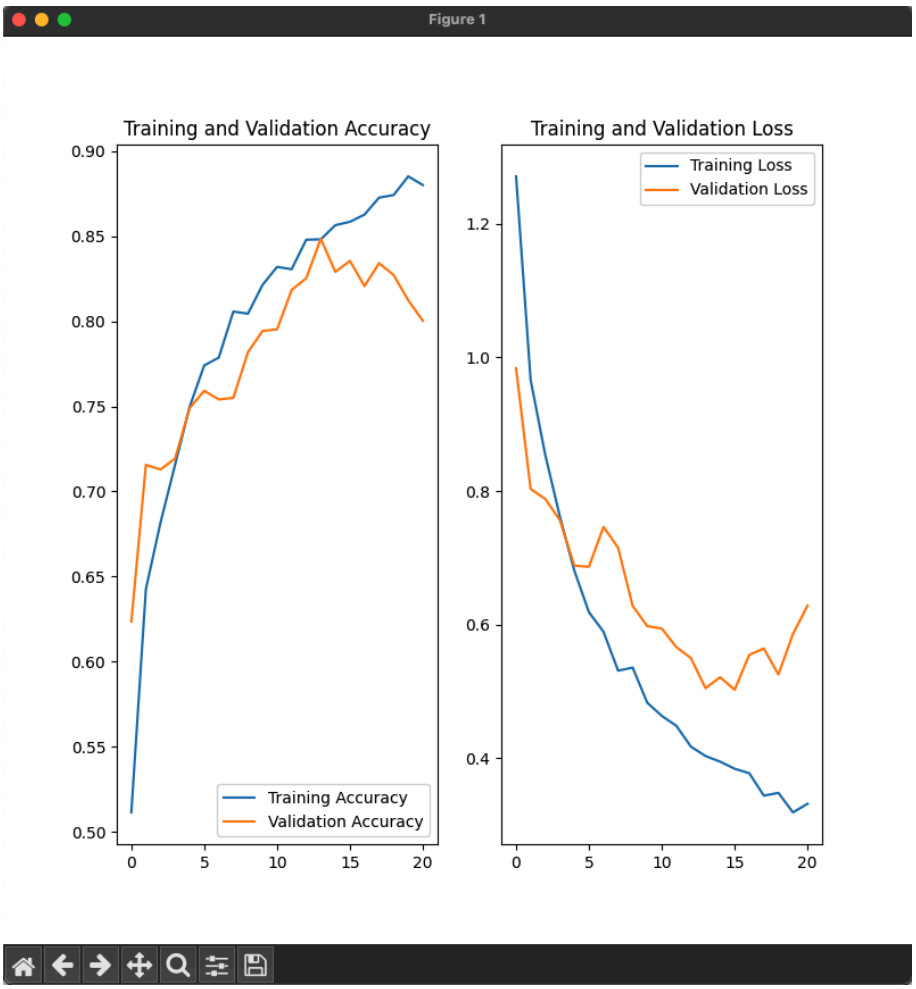


Figure 5.7.4: Training model results for Variant 2

Table 5.7.2: Correctly classified images for Variant 2

Class name	Correctly classified images [%]
elephant	79
giraffe	86
lion	79
parrot	81
rhino	54
tiger	74

- **Variant 3**: the number of filters in successive convolutional layers: 128 / 64 / 32, dropout was used.

```
271/271 ━━━━━━━━━━━━━━━━━━━━ 0s 538ms/step - accuracy: 0.4645 - loss: 1.36442024-08-31 14:50:40.729668: W tensorflow/core
incorrect sRGB profile
271/271 ━━━━━━━━━━━━━━━━━━━━ 167s 579ms/step - accuracy: 0.4649 - loss: 1.3635 - val_accuracy: 0.7161 - val_loss: 0.8150
Epoch 2/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 155s 571ms/step - accuracy: 0.7292 - loss: 0.7470 - val_accuracy: 0.7152 - val_loss: 0.7986
Epoch 3/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 155s 573ms/step - accuracy: 0.8008 - loss: 0.5470 - val_accuracy: 0.7684 - val_loss: 0.6715
Epoch 4/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 156s 577ms/step - accuracy: 0.8537 - loss: 0.4062 - val_accuracy: 0.7439 - val_loss: 0.7966
Epoch 5/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 157s 578ms/step - accuracy: 0.9076 - loss: 0.2707 - val_accuracy: 0.7776 - val_loss: 0.7673
Epoch 6/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 157s 579ms/step - accuracy: 0.9381 - loss: 0.1971 - val_accuracy: 0.8054 - val_loss: 0.7848
Epoch 7/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 155s 571ms/step - accuracy: 0.9542 - loss: 0.1235 - val_accuracy: 0.8044 - val_loss: 0.8430
Epoch 8/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 154s 570ms/step - accuracy: 0.9709 - loss: 0.0906 - val_accuracy: 0.8132 - val_loss: 0.9084
Model: "sequential_1"
```

Figure 5.7.5: Training model process for Variant 3

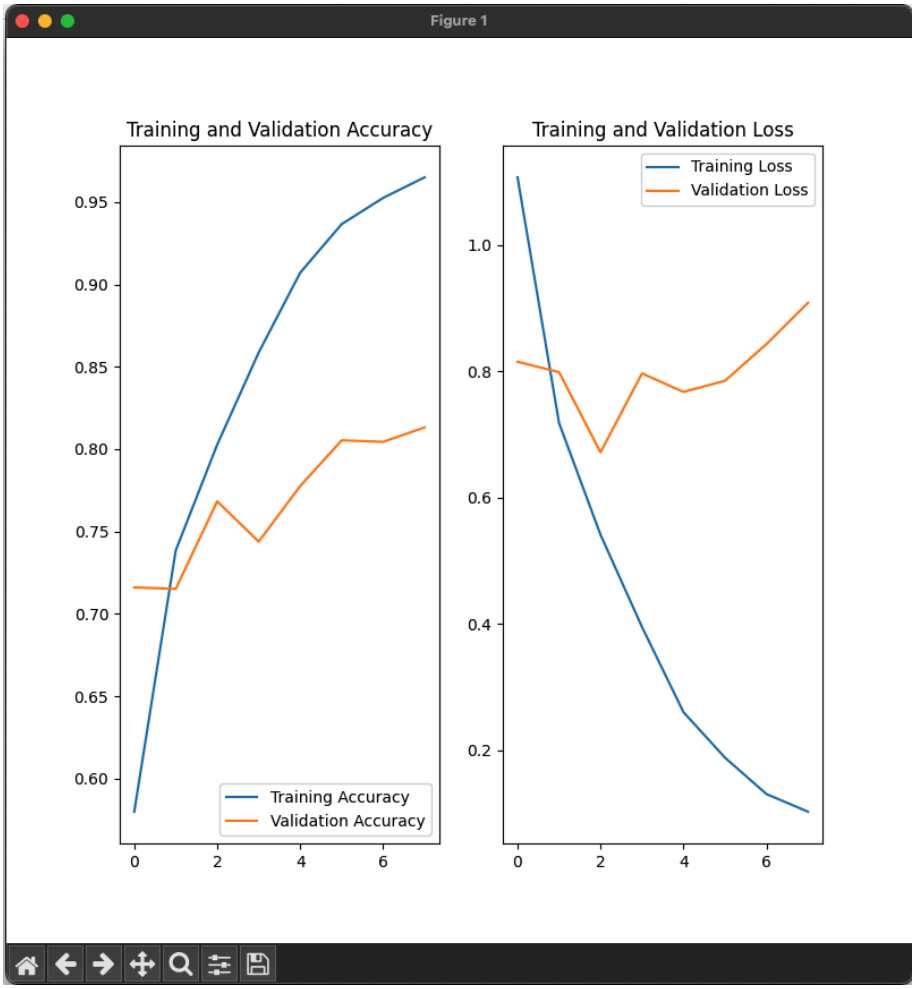


Figure 5.7.6: Training model results for Variant 3

Table 5.7.3: Correctly classified images for Variant 3

Class name	Correctly classified images [%]
elephant	82
giraffe	73
lion	67
parrot	76
rhino	43
tiger	62

- **Variant 4**: the number of filters in successive convolutional layers: 128 / 64 / 32, data augmentation was used.

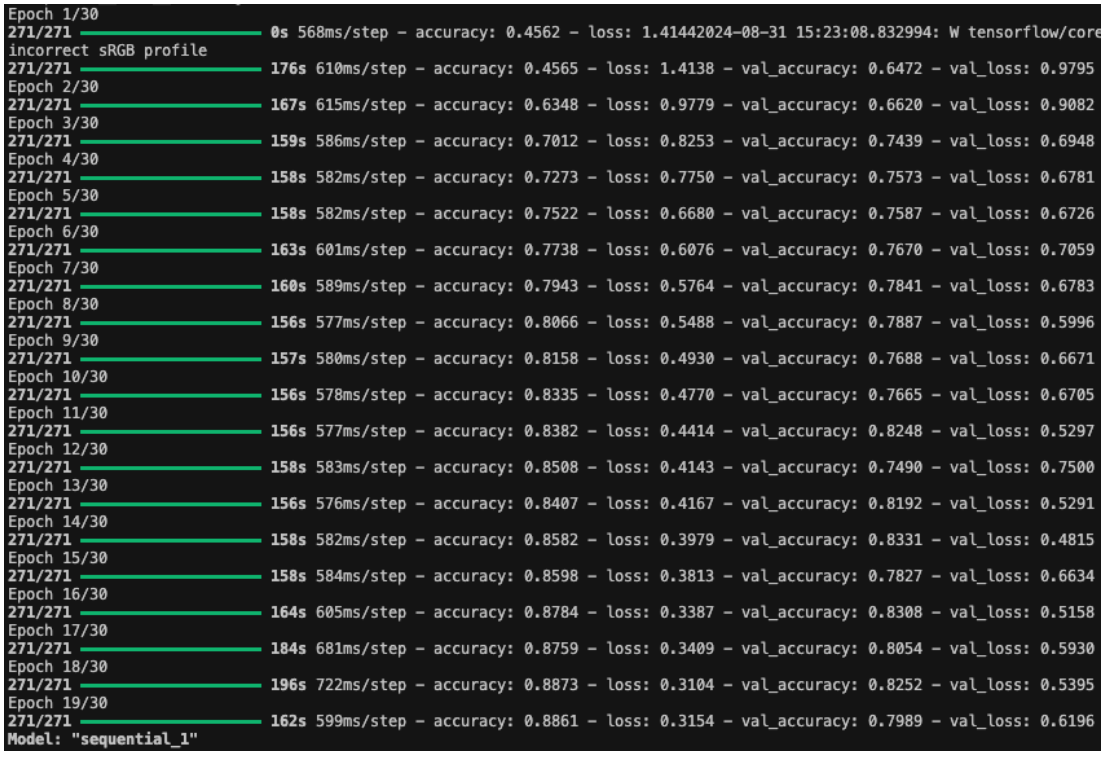


Figure 5.7.7: Training model process for Variant 4

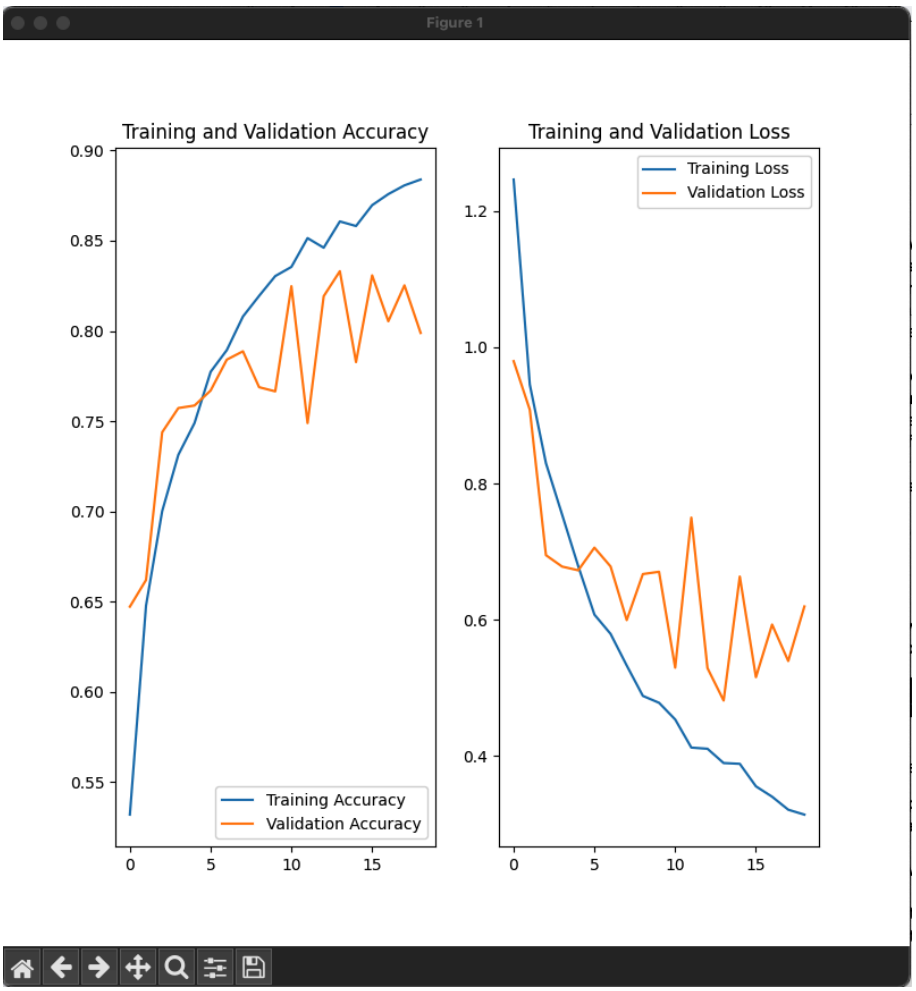


Figure 5.7.8: Training model results for Variant 4

Table 5.7.4: Correctly classified images for Variant 4

Class name	Correctly classified images [%]
elephant	82
giraffe	64
lion	75
parrot	79
rhino	69
tiger	78

5.8. Conclusions

As can be observed from the above tests performed on different model variants, Variant 1 turned out to be the best, correctly classifying an average of 77% of the images (table 5.7.1). Similarly, but slightly worse, was Variant 2 (table 5.7.2), which classified the images from the test set with an accuracy of 76%. It is worth noting that the only difference between Variant 1 and Variant 2 was the number of filters in the successive convolutional layers.

Variant 3 performed significantly worse, achieving an accuracy of only 67% (table 5.7.3). This was due to the lack of data augmentation, and as can be seen in figure 5.7.6, during the training of this model, overfitting occurred, meaning the model became excessively tailored to the training data. As a result, the model lost its ability to generalise to new, unseen data. This can be observed by the fact that the loss function value for the training data decreased rapidly, while for the validation data, it started to increase.

Variant 4 achieved 74% (table 5.7.4) accuracy which is a result similar to Variant 1 and Variant 2, indicating that the lack of the dropout method did not have as significant an impact on the model's performance as the absence of the data augmentation method.

In the figures 5.7.1 - 5.7.8 the learning process is shown. It can be seen how the loss function decreased during training (val_loss).

6. Motion detection in the video stream

6.1. Technologies

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library [22]. Originally developed by Intel around 2000 [23]. Today, it is used for various functions and tools for motion detection. The library has more than two and a half computer vision and machine learning algorithms. These can be used for many applications, such as: track moving objects like traffic monitoring, detect and recognize faces, identify objects, to remove red eyes from images, etc. The library nowadays plays a major role in image recognition tasks.

6.2. Motion detection methodology

Motion detection using OpenCV is a process of detecting and analyzing changes in the position of objects in a video stream. The method involves capturing video frames, checking them for variations in pixel intensity, and identifying movement in the scene [44][46].

The initial step in detecting moving objects is background subtraction. This engineering work uses such a solution to recognise movement by using OpenCV library. Background subtraction is commonly used with static camera applications to generate a foreground mask (a binary image containing the pixels belonging to moving objects in the scene).

Background subtraction works by calculating the foreground mask. It does this by calculating the difference between the current (moving) frame from and a background model that represents the static elements of the scene (anything that can be classified as the background based on the characteristics of the scene being observed, like street, forest, landscape, etc.). The working logic can be seen in the example of figure 6.2.1.

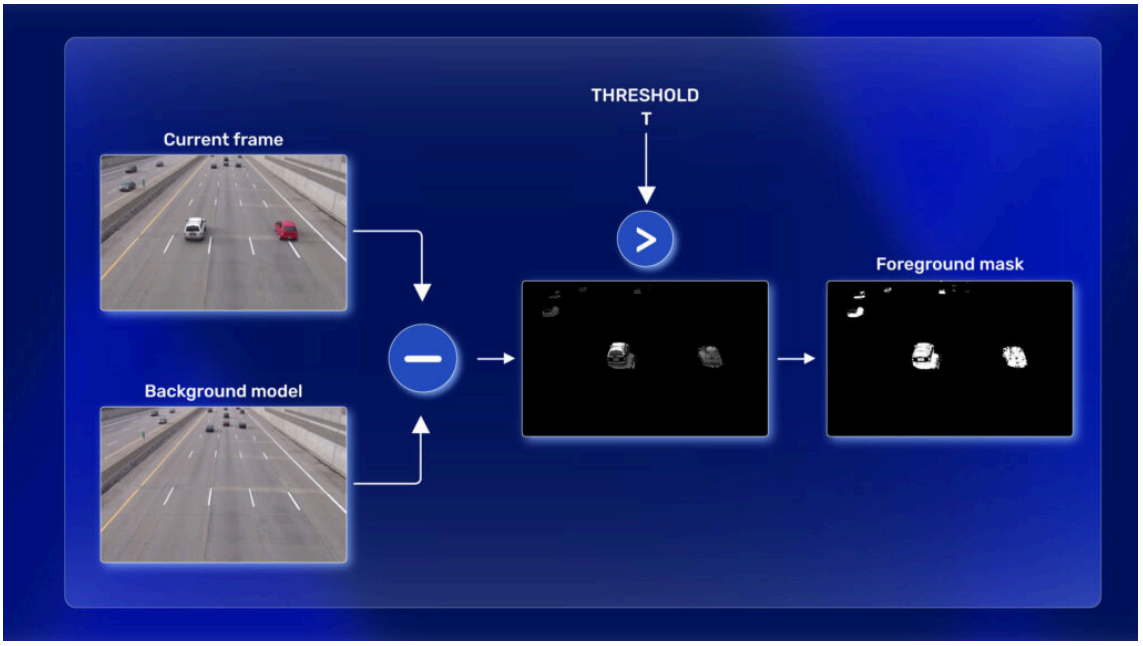


Figure 6.2.1: Logic Behind Background Subtraction [45]

There are two key steps in modelling a background: taking the first background and updating the background. During the first stage, an initial background model is created, while the second stage focuses on updating this model to take into account any changes in the scene over time. Then the process is repeated until image processing is complete.

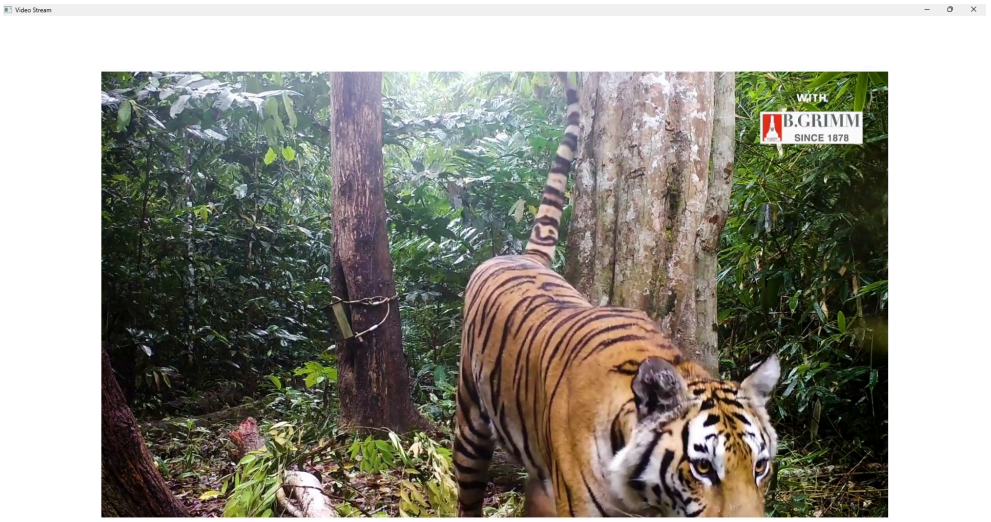


Figure 6.2.2: Current frame from hidden animal camera.

Figure 6.2.3: Frame processed by the app, after cutting out the background image (jungle)

Image 6.2.2 shows the processed video frame, while in image 6.2.3 we can see the differences detected. The white pixels show where the movement was noticed after the calculation.

If the number of white pixels exceeds a preset threshold, the program detects movement. Then the frame is sent to the API using the ***on_movement_detected()*** function.

6.3. Code

Nowadays, creating a motion detection application is relatively simple, using suitable methods from the OpenCV library [48]. Program logic can be implemented in several steps.
Steps to create motion detector:

1. Capture Video from Camera or File
   Start by capturing video from a camera or loading a pre-recorded video file using ***cv2.VideoCapture()*** (figure 6.3.1). RTP stream processing is also implemented in the application, which allows live image analysis, e.g. from a webcam.

```
cap = cv2.VideoCapture(video_path)
```

Figure 6.3.1: Launching video capture in code

2. Read and Preprocess Frames
   Each frame is read from the video stream, and it undergoes preprocessing steps to make motion detection more efficient. These steps include:
   - Converting to Grayscale (6.3.2) - makes the data more simply, colour is not needed for motion detection,

```
frame_bw = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Figure 6.3.2: Converting colour by using OpenCV function

- Blurring (6.3.3) - Reduces noise and small irregularities in the image and make it smooth to avoid false positives during motion detection.

```
frame_bw = cv2.GaussianBlur(frame_bw, (5,5), 0)
```

Figure 6.3.3: Simple Implementation of blurring

3. Background Subtraction (Compute Difference Between Frames)

   Compute the absolute difference between the current frame converted to grayscale *(frame_bw)* and the background frame. This helps to identify the areas where motion has occurred.
   To complete this task, an existing function from the OpenCV library was finally used (figure 6.3.4)

```
difference = cv2.absdiff(frame_bw, start_frame)
```

Figure 6.3.4: Implementation of background subtraction

Cv2.Absdiff Method calculates the per-element absolute difference between two arrays or between an array and a scalar [47].
For certain data types, like *uint8* (unsigned 8-bit integers), negative values cannot be represented since these types only allow non-negative numbers. When a negative value is encountered, it wraps around and results in an incorrect value. Signed types (such as floats or signed integers) can correctly represent negative values. That is why the cv.absdiff function was used here, it computes the absolute difference between values, ensuring that the result is always non-negative, making it safe to use with unsigned types like uint8.

4. Thresholding

```
threshold = cv2.threshold(difference, 25, 255, cv2.THRESH_BINARY)[1]
```

Figure 6.3.5: Implementation of threshold

For thresholding, the program used cv2.threshold function (figure 6.3.5). This is applied to turn this difference image into a binary image (black and white). The threshold value was set to 25 and the max value to 255, and in that way the program will simply replace all the values under 25 with 0 - pixels are turned to black (indicating no movement) and this removes all the shadows. Pixels with values higher than 25 are turned white (indicating movement). This helps isolate only significant changes in the scene.

```
85              # value sets the "sensitivity" of motion detection - the lower it is, the more sensitive it is
86              if threshold.sum() > sensitivity:
87                  alarm_counter += 1
88              else:
89                  if alarm_counter > 0:
90                      alarm_counter -= 1
91              cv2.imshow('Video Stream', threshold)
92          else:
93              cv2.imshow('Video Stream', frame)
94
95          if alarm_counter > 20:
96              if not alarm:      Piotr, 4 months ago • handle exceptions, send frame to api -> done
97                  alarm = True
98                  threading.Thread(target=on_movement_detected).start()
```

Figure 6.3.6: Implementation of motion recognition sensitivity detection

Conditional statement if with threshold.sum() computes the sum of all pixel values in the binary image. In a binary image, pixels can either be 0 (black) or 1 (white), so the sum reflects how many white pixels (indicating detected motion) are present. In the condition program checks whether the sum of the white pixels exceeds the value of the **sensitivity** variable. If the value is higher than the **sensitivity** variable value, it interprets this as significant movement.

Sensitivity variable refers then to how much change in the image is required to be considered as a "motion." It is a threshold for motion to be detected, the sum of white pixels (areas of change) must be greater than **sensitivity** value. It helps to avoid smaller changes in the image that will be detected as motion.

The **sensitivity** value should be chosen experimentally. It is worth noting that motion detection is reported only when the alarm counter reaches a value of 20 (each time the sensitivity threshold is exceeded, the counter is incremented, otherwise the counter is decremented).

5. Post motion detection tasks

All the tasks that the application performs when the motion detection threshold is exceeded are contained in the function shown in 6.3.7. A helper function (6.3.8) was also created for converting images.

```
def on_movement_detected():
```

Figure 6.3.7: Function that performs all tasks when movement has been detected

```
def save_snapshot_as_base64(image, directory):
```

Figure 6.3.8: Function that encodes the image to Base64

6. Send results

   Sending of results is contained in the conditional instruction if (6.3.9). This is done by using the POST request method supported by HTTP (6.3.10). Frame is sent directly to the API.

```
if frame_to_send is not None:
```

Figure 6.3.9: Condition checking if frame exists

```
response = requests.post(api_url, json=body, headers=headers)
```

Figure 6.3.10: Function invocation which sends the encoded image to the API

7. Loop and Release Resources

   All necessary image processing is included in the main function of this part of the application (6.3.11) which processes frames received from RTP stream one by one. Program ends with the release of resources (6.3.12). These lines are executed when the application is shut down.

```
# MAIN LOOP
while True:
    try:
```

Figure 6.3.11: Main loop initiation

```
        break

cap.release()
cv2.destroyAllWindows()
```

Figure 6.3.12: Resource release functions

7. User Interface

The front-end application was written by using the React library. This solution allowed the project to be kept very organised (figure 7.0.1)

User Interface is a simple web application used to display frames sent by the Motion Detector to the API along with the animal classes assigned to them. The application communicates only with the API microservice, and the communication is possible only after the user has been successfully authenticated.

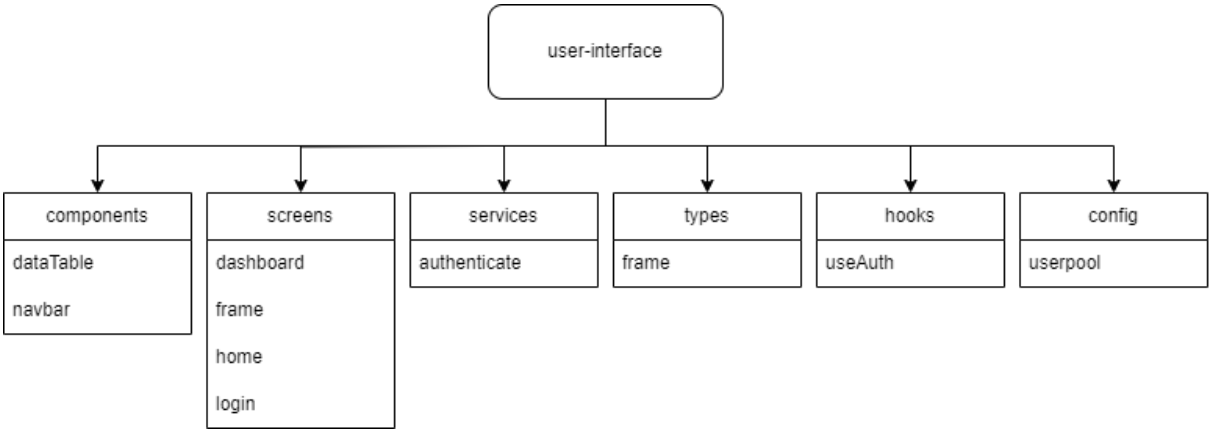


Figure 7.0.1: Front-end structure

The application offers the following three views: login, dashboard, frame.

7.1. Login view

Login view (figure 7.1.1) - it is used to log into the system. The authentication process, after submitting the username and password, is carried out by the AWS Cognito service.

AWS Cognito is a service that helps you manage user sign-up, sign-in, and access control for web and mobile apps. It simplifies the process of authenticating users by allowing them to log in using their email, phone number, or third-party services. Cognito allows users to securely manage user accounts, handle password resets and multi-factor authentication (MFA) and control user permissions by using access tokens.

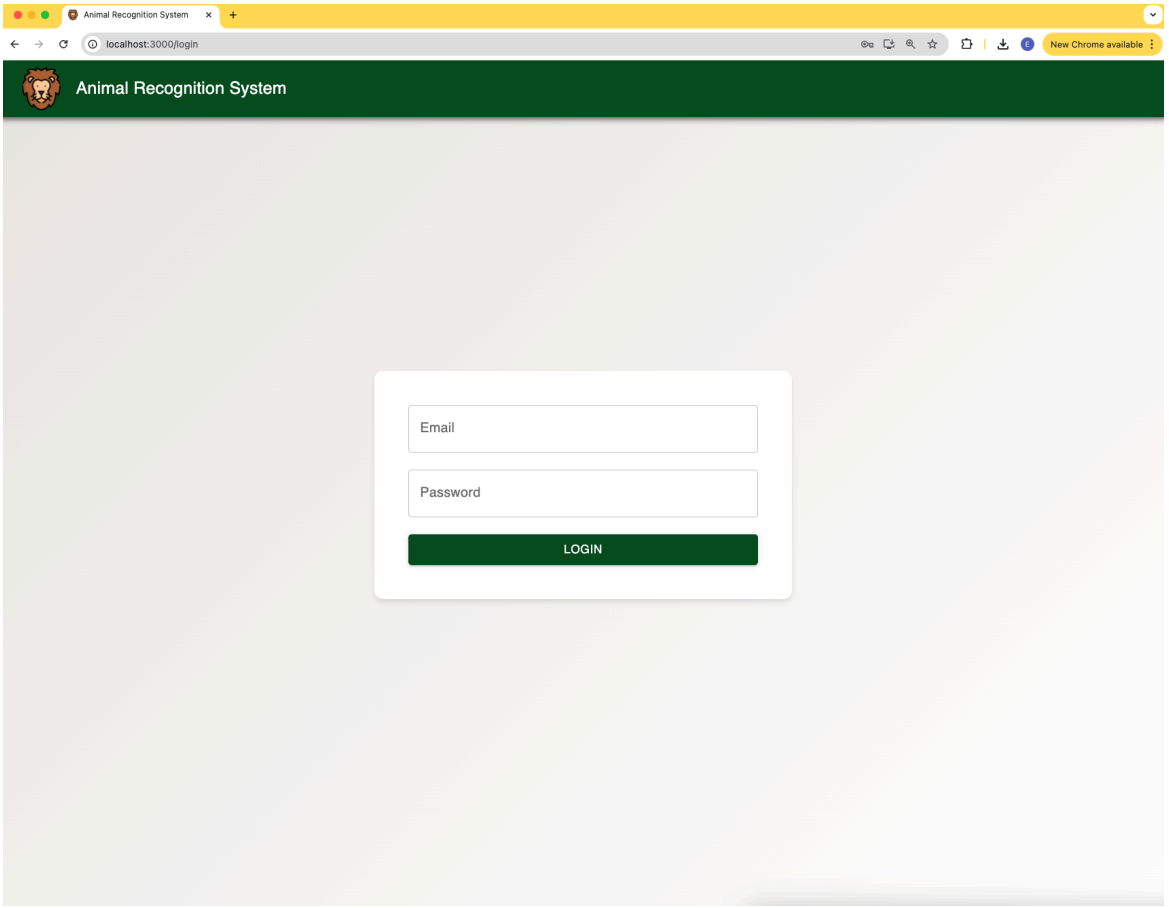
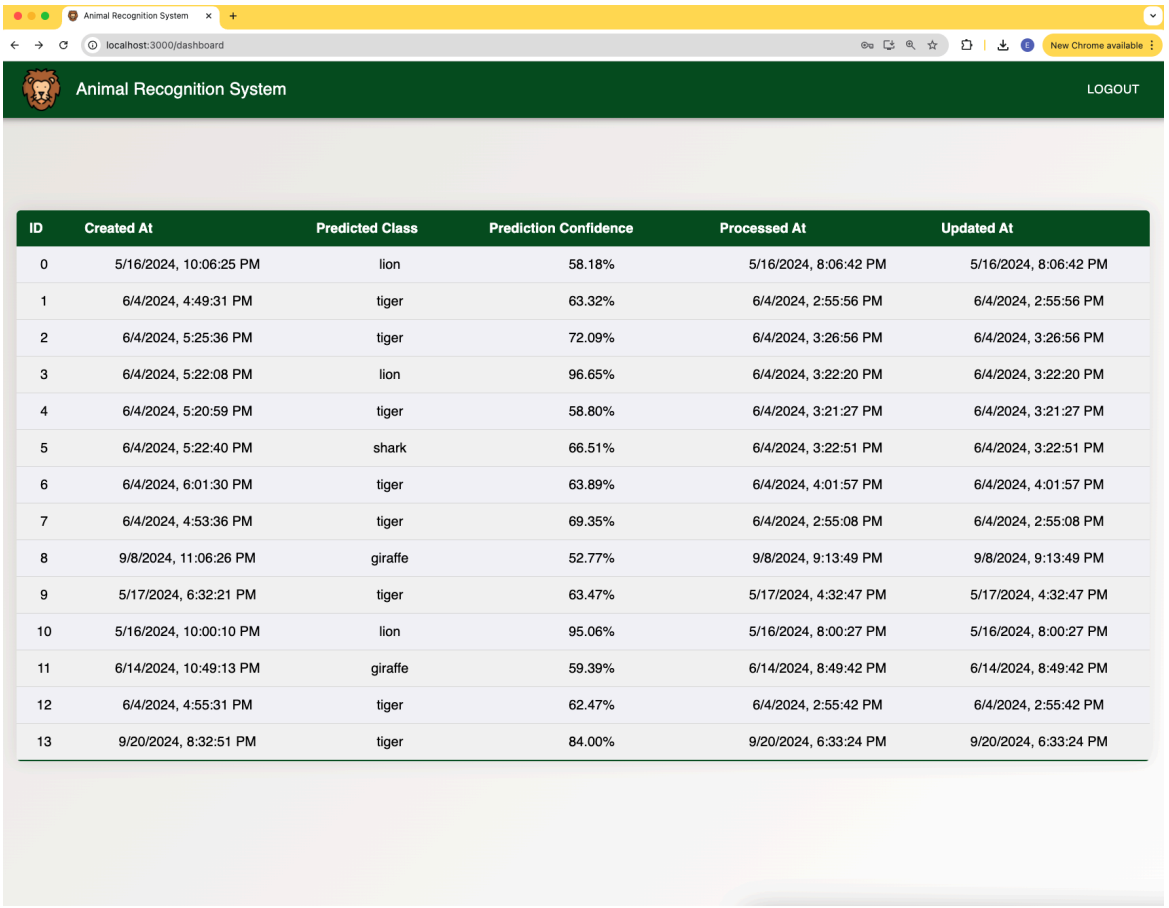


Figure 7.1.1: Login view

7.2. Dashboard view

Dashboard view (figure 7.1.2) - it displays a list of frames collected in the database along with the animal species (classes) assigned to them.



ID	Created At	Predicted Class	Prediction Confidence	Processed At	Updated At
0	5/16/2024, 10:06:25 PM	lion	58.18%	5/16/2024, 8:06:42 PM	5/16/2024, 8:06:42 PM
1	6/4/2024, 4:49:31 PM	tiger	63.32%	6/4/2024, 2:55:56 PM	6/4/2024, 2:55:56 PM
2	6/4/2024, 5:25:36 PM	tiger	72.09%	6/4/2024, 3:26:56 PM	6/4/2024, 3:26:56 PM
3	6/4/2024, 5:22:08 PM	lion	96.65%	6/4/2024, 3:22:20 PM	6/4/2024, 3:22:20 PM
4	6/4/2024, 5:20:59 PM	tiger	58.80%	6/4/2024, 3:21:27 PM	6/4/2024, 3:21:27 PM
5	6/4/2024, 5:22:40 PM	shark	66.51%	6/4/2024, 3:22:51 PM	6/4/2024, 3:22:51 PM
6	6/4/2024, 6:01:30 PM	tiger	63.89%	6/4/2024, 4:01:57 PM	6/4/2024, 4:01:57 PM
7	6/4/2024, 4:53:36 PM	tiger	69.35%	6/4/2024, 2:55:08 PM	6/4/2024, 2:55:08 PM
8	9/8/2024, 11:06:26 PM	giraffe	52.77%	9/8/2024, 9:13:49 PM	9/8/2024, 9:13:49 PM
9	5/17/2024, 6:32:21 PM	tiger	63.47%	5/17/2024, 4:32:47 PM	5/17/2024, 4:32:47 PM
10	5/16/2024, 10:00:10 PM	lion	95.06%	5/16/2024, 8:00:27 PM	5/16/2024, 8:00:27 PM
11	6/14/2024, 10:49:13 PM	giraffe	59.39%	6/14/2024, 8:49:42 PM	6/14/2024, 8:49:42 PM
12	6/4/2024, 4:55:31 PM	tiger	62.47%	6/4/2024, 2:55:42 PM	6/4/2024, 2:55:42 PM
13	9/20/2024, 8:32:51 PM	tiger	84.00%	9/20/2024, 6:33:24 PM	9/20/2024, 6:33:24 PM

Figure 7.1.2: Dashboard view

7.3. Frame view

Frame view (figure 7.1.3) - it is a view that appears after clicking on an item in the list on the Dashboard view. It allows you to see a specific frame from the camera.

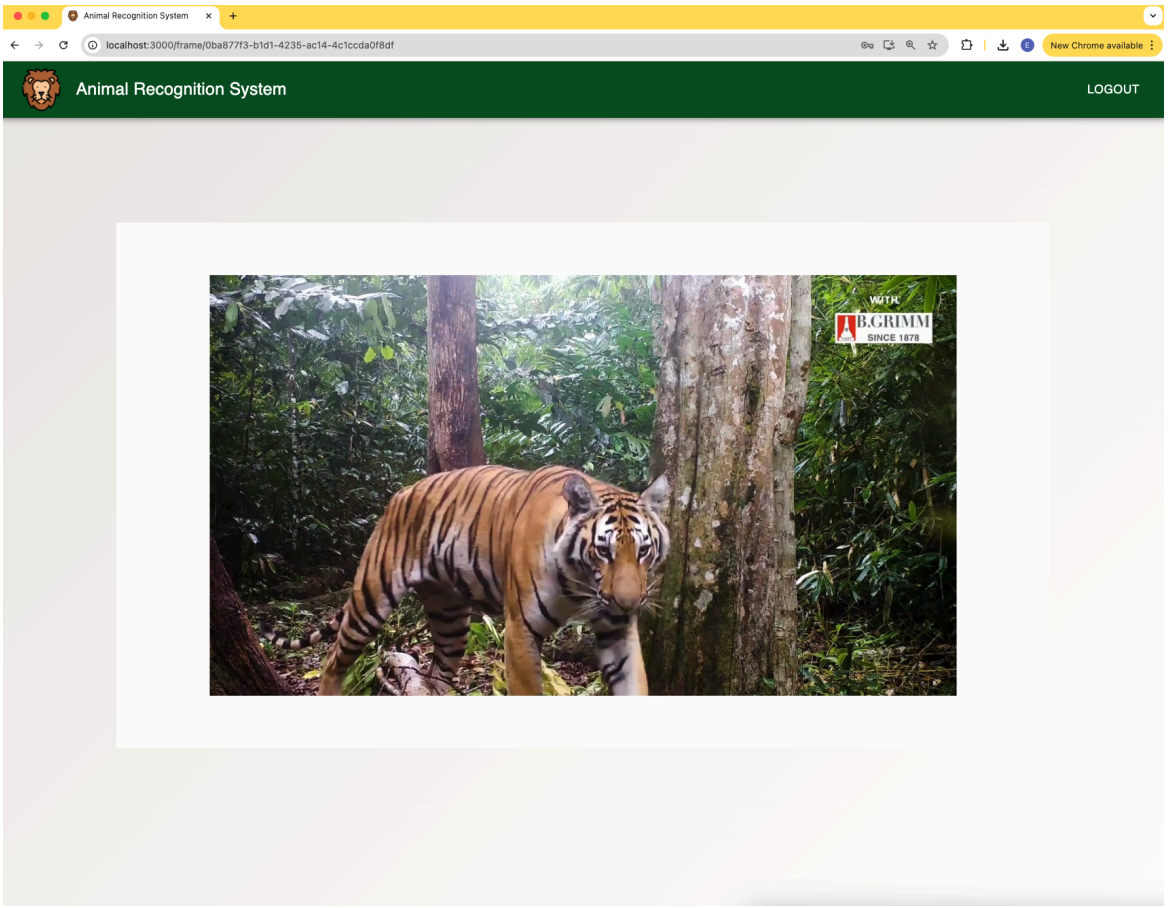


Figure 7.1.3: Frame view

8. API

8.1. Motivation

In order to enable the User Interface and Motion Detector components to communicate with the part of the system that is located in the cloud, there was a need to create an API component that provides both public and private endpoints.

All endpoints use the HTTP protocol and its various methods such as POST, GET, PUT, etc. In the system architecture, the API serves as the central microservice that communicates with all other microservices.

8.2. Technologies

Programming language - API microservice has been written using TypeScript which is a modern and very popular language nowadays. Creating API with TypeScript is very convenient and safe as TypeScript is a static and strongly typed language.

Infrastructure - the API is running in the cloud. Cloud provider which has been chosen for the system is AWS therefore following AWS services have been used:

- API Gateway - for creation, deployment and management of HTTP API. It acts as a front door to API microservice. [12].
- Cognito - provide authentication and authorization for API microservice [37].
- Lambda - serverless compute service where code can be run without provisioning and managing any servers. Code is executed on demand and it scales automatically [11].
- S3 - provide scalable object storage where frames received from Motion Detector microservice are stored [10].
- DynamoDB - fully managed NoSQL database service where all the data about recognitions are stored [9].

Using those AWS services collectively our API microservice is highly scalable, highly available and has security on a very high level.

In order to define and create the infrastructure for the API microservice, the Serverless Framework was used. The Serverless Framework is an open-source tool that simplifies the deployment and management of serverless applications.

8.3. Endpoints

The API has only one user group - there is no division between admins and regular users as it was not necessary in the basic version of the project. However, the API does have different authorization methods for various endpoints:

- API key - used by the endpoints that serve the Motion Detector microservice,
- JWT - used by the endpoints that serve the User Interface web application.

In API microservice following HTTP endpoints are available to deliver data to User Interface web application and show all the data to user:

Table 8.1.1: Endpoints available in API microservice

Method	Endpoint	Description
GET	/frames	Obtain list of all informations about analyzed frame which system performed
GET	/frames/{id}	Obtain all information about analyzed frame with actual image data encoded in base64
POST	/frames	Create new frame which will be analyzed asynchronously

9. Conclusions

The goal of the project was to design and implement a Real-Time Animal Monitoring system, which could be used by organizations involved in observing animals in their natural habitat. The main non-functional requirements that the system had to meet were simplicity of installation and high scalability. Both of these requirements were achieved thanks to the use of cloud technology, which means that the user, in addition to installing cameras in the area where the animals are located, only needs to have one server with relatively low computing power, as this server will only be responsible for motion detection. The most computationally expensive task, namely the classification of frames received from the cameras, will take place in the cloud.

When it comes to effectiveness of animal recognition from the frames which is done by Animal Recognition Algorithm microservice, based on the tests conducted, it is approximately 77%. The testing process of the trained models is described in detail in section 5.8. In the future, this accuracy can certainly be improved by collecting more training data and further training the model. Particularly good results can be achieved by using the **transfer learning** [49] method with data from cameras, where the classes were assigned by the system's user.

The second factor, besides the model's performance, that affects the overall efficiency of the system is proper motion detection. We should aim to minimize the number of frames sent to the API that do not contain any animals (so-called false-positive results). This can be controlled using the **sensitivity** environment variable in the Motion Detector microservice. In this way, we can adjust this parameter according to the environment where the camera is installed. This is important because, in some environments, there may be moving elements in the camera's frame that are not animals, such as moving tree branches.

The other two implemented microservices, namely the API and User Interface, do not require special configuration. The User Interface is responsible for displaying data to the end user after logging in. The API, on the other hand, is responsible for collecting data from the Motion Detector and Animal Recognition Algorithm microservices and passing it to the User Interface (see the architecture diagram in section 3.2).

From the end-user’s perspective, the system may seem to lack many features; however, in our work, we have built the backbone of the entire system, and adding additional features to improve the user experience will be very easy. The system's extensibility and modifiability are also facilitated by the fact that a microservices architecture was applied, along with good practices of writing clean code.

Some ideas for additional features that could significantly enhance its usefulness include:

- Geotagging – displaying a map of the area and showing the real-time location of specific animal species,
- Training models to recognize animal species from environments other than safaris, such as jungles, deserts, etc.,
- Alerting for the detection of rare species,
- Alerting when a predator is spotted near areas where people are present.

Dictionary

AI - Artificial Intelligence

ANN - Artificial Neural Network

API - Application Programming Interface

AWS - Amazon Web Services

CI/CD - Continuous Integration (CI) and Continuous Delivery (CD)

CNN - Convolutional Neural Networks

DOM - Document Object Model

GNU - General Public License

HTTP - Hypertext Transfer Protocol

JWT - Json Web Token

LLM - Large Language Model

MFA - Multi-Factor Authentication

ML - Machine Learning

RNN - Recurrent Neural Networks

RTP - Real-Time Transport Protocol

RTSP - Real-Time Streaming Protocol

SPA - Single-Page Application

SQS - Amazon Simple Queue Service

Bibliography

1. E. Freeman, K. Sierra, B. Bates, *Head First Design Patterns*, Helion, 2017.
2. Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd Edition*, O'Reilly Media, 2019
3. https://en.wikipedia.org/wiki/Microservices
4. https://www.cvedia.com
5. https://www.cvedia.com/animal-detection
6. https://en.wikipedia.org/wiki/Git
7. https://en.wikipedia.org/wiki/GitHub
8. https://docs.github.com/en/get-started/start-your-journey/about-github-and-git
9. https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html
10. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
11. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
12. https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html
13. https://legacy.reactjs.org/
14. https://legacy.reactjs.org/docs/getting-started.html
15. https://www.tensorflow.org/about
16. https://www.tensorflow.org/api_docs/python/tf#tensorflow
17. https://keras.io/about/
18. https://keras.io/getting_started/intro_to_keras_for_engineers/
19. https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html
20. https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-func.html
21. https://docs.python.org/3/tutorial/index.html
22. https://opencv.org/about/
23. https://en.wikipedia.org/wiki/OpenCV
24. https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html#typescript-a-static-type-checker
25. https://www.atlassian.com/software/jira
26. https://en.wikipedia.org/wiki/Jira_(software)
27. https://aws.amazon.com/
28. https://en.wikipedia.org/wiki/Amazon_Web_Services
29. https://aws.amazon.com/about-aws/
30. https://en.wikipedia.org/wiki/Cloud_computing
31. https://www.frogriot.com/blog/pl/aws-czym-sa-amazon-web-services-qa/ (dostęp: 31.07.2024)
32. https://en.wikipedia.org/wiki/TensorFlow
33. https://en.wikipedia.org/wiki/Keras
34. https://en.wikipedia.org/wiki/Neural_network_(machine_learning)
35. https://en.wikipedia.org/wiki/Keras
36. https://en.wikipedia.org/wiki/Document_Object_Model
37. https://docs.aws.amazon.com/cognito/
38. https://www.ibm.com/topics/machine-learning (Access: 25.09.2024)
39. https://www.ibm.com/topics/neural-networks (Access: 25.09.2024)

40. https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/ (Access: 25.09.2024)
41. https://8thlight.com/insights/what-is-a-convolution-how-to-teach-machines-to-see-images (Access: 25.09.2024)
42. https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5 (Access: 25.09.2024)
43. https://builtin.com/machine-learning/adam-optimization (Access: 25.09.2024)
44. https://hackmd.io/@lKuOpplzSUWLhLim2Z7ZJw/BkGj66aW3 (Access: 20.09.2024)
45. https://learnopencv.com/moving-object-detection-with-opencv/ (Access: 20.09.2024)
46. https://www.geeksforgeeks.org/opencv-overview/ (Access: 21.09.2024)
47. https://shimat.github.io/opencvsharp_docs/html/6c7cefd1-59bc-a595-fe2a-0c8a709f8d16.htm (Access: 21.09.2024)
48. https://shimat.github.io/opencvsharp_docs/html/6121915d-1174-7345-bdca-789ee1373642.htm (Access: 21.09.2024)
49. https://www.tensorflow.org/tutorials/images/transfer_learning

Index of tables

Index of figures