Image Classification as a Software Platform

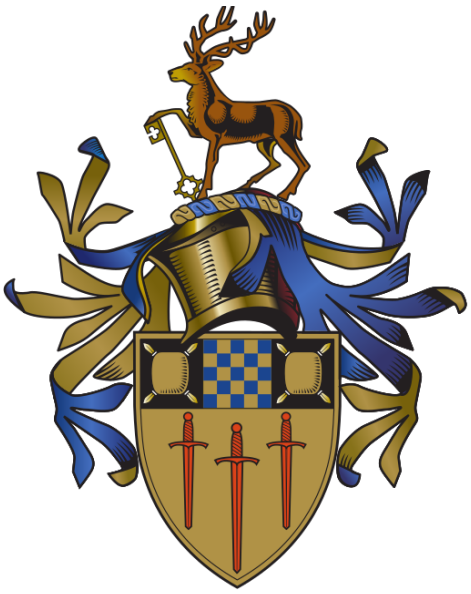
by

Andre Goncalves Henriques
URN: 6644818

A dissertation submitted in partial fulfilment of the
requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

May, 2024



Department of Computer Science
University of Surrey
Guildford GU2 7XH

Supervised by: Dr. Rizwan Asghar

Declaration of Originality

I confirm that the submitted work is my own work and that I have clearly identified and fully acknowledged all material that is entitled to be attributed to others (whether published or unpublished) using the referencing system set out in the programme handbook. I agree that the University may submit my work to means of checking this, such as the plagiarism detection service Turnitin® UK. I confirm that I understand that assessed work that has been shown to have been plagiarised will be penalised.

Andre Goncalves Henriques
May, 2024
University of Surrey
Guildford GU27XH

Acknowledgements

I would like to take this opportunity to thank my supervisor, Rizwan Asghar who helped me with this project from the start of the until the end. His help with the report was incredibly useful.
I would like to thank my family and friends for their support and encouragement.

Andre Goncalves Henriques
May, 2024
University of Surrey
Guildford GU27XH

Abstract

There are many automatable tasks that are currently being done manually. If those tasks can be done automatically, a lot of productivity could be gained by having the computers perform those tasks, allowing humans to perform tasks that only humans can do. One of this set of tasks are image classification tasks. Many image classification tasks are being performed by humans, when they could be performed by computers.

This project aims to develop an image classification platform where users can create image classification models with as few clicks as possible. Allowing for users that do not have any knowledge about image classification to use this system. Making it possible for more of the manually classified tasks to be done by machines, and not humans, increasing the productivity of possible users.

This dissertation evaluates the feasibility of such system, current similarly implemented systems, current techniques for image classification, and possible requirements, designs and implementations of such system. The dissertation focuses mainly on the implemented software, and the implementation choices that were made to achieve this project. The dissertation ends with a critical evaluation of the results of this project, to ensure that the goals set forth were achieved.

Contents

1 Introduction

The purpose of this dissertation is to design and implement an automated image classification service that will empower users to connect their existing services that required image classification with the one being implemented in this project. This report will detail what requirements such service might have. How those requirements can be turned into a design for such a service, and how that design can be implemented into software with limited time and resources.

This chapter will service as an introduction to the project, and will discuss the background, motivations, aims, goals, and success criteria for the project. This chapter will end with an overview of the project structure.

1.1 Project Background

There are many automatable tasks that are currently being done manually. If those tasks can be done automatically, a lot of productivity could be gained by having the computers perform those tasks, allowing humans to perform tasks that only humans can do. Moreover, recently, machine learning models have become as good, or even better than humans in tasks such image classification. This project will focus on image classification, as it is an area where automation is still required, and there are few other images automated systems that are easy to use. It is also an area that has not been saturated with new products such as the natural language processing space has.

1.2 Project Motivations

This project allows for the improvement of my learned skills, while being an interesting, complex piece of software to develop. The topics, skills, and knowledge required to build this project, cover all my years in the university, from the simple applications developed in the first year; the soft skills learned during placement; and the complexity of distributed systems and deep learning of the third year. I also wanted to use the opportunity that this project provides to gain experience in emerging technologies such as Go, and improve all my previous abilities and skills.

1.3 Project Aim

The project aims to create an easy-to-use software platform, where users can create image classification models without having prior knowledge about machine learning. The user should only need to upload the images and confirm, and the system should be able to perform all the steps necessary to create and manage the machine learning model.

1.4 Project Objectives

This project will have two different objectives. Primary objectives are objectives that are required for the project to be considered a success. Secondary objectives are objectives that are not required for the project to be considered a success, but they would provide a better experience for the user of the service.

This project's primary objectives are to design and implement:

- a system to upload images that will be assigned to a model.
- a system to automatically create and train models.
- a platform where users can manage their models.
- a system to automatically expand models without fully retraining the models.
- an Application Programming Interface (API) that users can interact programmatically with the service.

This project's secondary objectives are to:

- Create a system to distribute the load of training the model among multiple servers.

1.5 Success Criteria

As it was mentioned before, the project can be considered a success when the primary objectives have been completed. Therefore, the success criteria of this project can be defined as:

- A user can upload images, train a model on those images, and evaluate images using a user interface.
- A user can perform the same tasks, via the API service.

1.6 Project Structure

The report on the project shows the development and designs stages of the project. With each chapter addressing a part of the design and development process.

Introduction	The introduction chapter will do a brief introduction of the project and its objectives.
Literature and Technical Review	The Literature and Technical Review chapter will introduce some current existing projects that are similar to this one, and introduce some technologies that can be used to implement this project.
Service Analysis and Requirements	This chapter will analyse the project requirements. The chapter will define design requirements that the service will need to implement to be able to achieve the goals that were set up.
Service Design	This chapter will discuss how a service could be designed that it matches the requirements of the service.
Service Implementation	Information on how the design of the system was turned into software is in this chapter.
Legal, Societal, Ethical, Professional Considerations	This chapter will cover potential legal, societal, ethical and professional, issues that might arise from the service and how they are mitigated.
Service Evaluation	In this chapter, the model will be tested and the results of the tests will be analysed.
Critical Review of Project Outcomes	In this chapter, will compare the project goals with what was achieved. Then, according to the results, the project will either be deemed successful or not.

2 Literature and Technical Review

This chapter reviews existing technologies in the market that do image classification. It also reviews current image classification technologies, which meet the requirements for the project. This review also analyses methods that are used to distribute the learning between various physical machines, and how to spread the load so minimum reloading of the models is required when running the model.

2.1 Existing Classification Platforms

There are currently some existing software as a service (SaaS) platforms that do provide similar services to the ones this will project will be providing.

Amazon provides an image classification service called "Rekognition" [1]. This service provides multiple services from face recognition, celebrity recognition, object recognition and others. One of these services is called custom labels [2] that provides the most similar service, to the one this project is about. The custom labels service allows the users to provide custom datasets and labels and using AutoML the Rekognition service would generate a model that allows the users to classify images according to the generated model.

The models generated using Amazon's Rekognition do not provide ways to update the number of labels that were created, without generating a new project. This will involve retraining a large part of the model, which would involve large downtime between being able to add new classes. Training models also could take 30 minutes to 24 hours, [3], which could result in up to 24 hours of lag between the need of creating a new label and being able to classify that label. A problem also arises when the uses need to add more than one label at the same time. For example, the user sees the need to create a new label and starts a new model training, but while the model is training a new label is also needed. The user now either stops the training of the new model and retrains a new one, or waits until the one currently running stops and trains a new one. If new classification classes are required with frequency, this might not be the best platform to choose.

Similarly, Google also has "Cloud Vision API" [4] which provides similar services to Amazon's Rekognition. But Google's Vision API appears to be more targeted at videos than images, as indicated by their price sheet [5]. They have tag and product identifiers, where every image only has one tag or product. The product identifier system seams to work differently than the Amazon's Rekognition and worked based on K neighbouring giving the user similar products on not classification labels [6].

This method is more effective at allowing users to add new types of products, but as it does not give defined classes as the output, the system does not give the target functionality that this project is aiming to achieve.

2.2 Requirements of Image Classification Models

One of the main objectives of this project are to be able to create models that can give a class given an image for any dataset. Which means that there will be no "one solution fits all to the problem". While the most complex way to solve a problem would most likely result in success, it might not be the most efficient way to achieve the results.

This section will analyse possible models that would obtain the best results. The models for this project have to be the most efficient as possible while resulting in the best accuracy as possible.

A classical example is the MNIST Dataset [7]. Models for the classification of the MNIST dataset can be both simple or extremely complex and achieve different levels of complexity. For example, in [8] an accuracy 99.91%, by combining 3 Convolutional Neural Networks (CNNs), with different kernel sizes and by changing hyperparameters, augmenting the data, and in [9] an accuracy of 95% was achieved using a 2 layer neural network with 300 hidden nodes. Both these models achieve the accuracy that is required for this project, but [8] are more computational intensive to run. When deciding when to choose what models they create, the system should choose to create the model that can achieve the required accuracy while taking the leas amount of effort to train.

The models for this system to work as indented should be as small as possible while obtaining the required accuracy required to achieve the task of classification of the classes.

As the service might need to handle many requests, it needs to be able to handle as many requests as possible. This would require that the models are easy to run, and smaller models are easier to run; therefore the system requires a balance between size and accuracy.

2.3 Method of Image Classification Models

There are all multiple ways of achieving image classification, the requirements of the system are that the system should return the class that an image that belongs to. Which means that we will be using supervised classification methods, as these are the ones that meet the requirements of the system.

The system will use supervised models to classify images, using a combination of different types of models, using neural networks, convolution neural networks, deed neural networks and deep convolution neural networks. These types were decided as they have had a large success in the past in other image classification challenges, for example in the ImageNet challenges [10], which has ranked different models in classifying a 14 million images. The contest has been running since 2010 to 2017.

The models that participated in the contest tended to use more and more Deep convolution neural networks, out of the various models that were generated there are a few landmark models that were able to achieve high accuracies, including AlexNet [11], ResNet-152 [12], EfficientNet [13].

These models can be used in two ways in the system, they can be used to generate the models via transfer learning and by using the model structure as a basis to generate a complete new model.

2.4 Well-known models

This section will compare the different models that did well in the image net challenge.

AlexNet [11] is a deep convolution neural network that participated in the ImageNet ILSVRC-2010 contest, it achieved a top-1 error rate of 37.5%, and a top-5 error rate of 37.5%. A variant of this model participated in the ImageNet LSVRC-2012 contest and achieved a top-5 error rate of 15.3%. The architecture of AlexNet consists of 5 convolution layers that are run separately followed by 3 dense layers, some layers are followed by Max pooling. The training the that was done using multiple GPUs, one GPU would run the part of each layer, and some layers are connected between GPUs. The model during training also contained data argumentation techniques such as label preserving data augmentation and dropout. While using AlexNet would probably yield desired results, it would complicate the other parts of the service. As a platform as a service, the system needs to manage the number of resources available, and requiring to use 2 GPUs to train a model would limit the number of resources available to the system by 2-fold.

ResNet [14] is a deep convolution neural network that participated in the ImageNet ILSVRC-2015 contest, it achieved a top-1 error rate of 21.43% and a top-5 error rate of 5.71%. ResNet was created to solve a problem, the problem of degradation of training accuracy when using deeper models. Close to the release of the ResNet paper, there was evidence that deeper networks result in higher accuracy results, [15, 16]. but the increasing the depth of the network resulted in training accuracy degradation. ResNet works by creating shortcuts between sets of layers, the shortcuts allow residual values from previous layers to be used on the upper layers. The hypothesis being that it is easier to optimize the residual mappings than the linear mappings. The results proved that the using the residual values improved training of the model, as the results of the challenge prove. It's important to note that using residual networks tends to give better results, the more layers the model has. While this could have a negative impact on performance, the number of parameters per layer does not grow that steeply in ResNet when comparing it with other architectures as it uses other optimisations such as $1x1$ kernel sizes, which are more space efficient. Even with these optimisations, it can still achieve incredible results. Which might make it a good contender to be used in the service as one of the predefined models to use to try to create the machine learning models.

EfficientNet [17] is a deep convolution neural network that was able to achieve 84.3% top-1 accuracy while "$8.4x$ smaller and $6.1x$ faster on inference than the best existing ConvNet". EfficientNets [1] are models that instead of the of just increasing the depth or the width of the model, we increase all the parameters at the same time by a constant value. By not scaling only depth, EfficientNets can

[1] the family of models that use the thecniques that described in [17]

acquire more information about the images, specially the image size is considered. To test their results, the EfficientNet team created a baseline model which as a building block used the mobile inverted bottleneck MBConv [18]. The baseline model was then scaled using the compound method, which resulted in better top-1 and top-5 accuracy. While EfficientNets are smaller than their non-EfficientNet counterparts, they are more computational intensive, a ResNet-50 scaled using the EfficientNet compound scaling method is 3% more computational intensive than a ResNet-50 scaled using only depth while improving the top-1 accuracy by 0.7%. And as the model will be trained and run multiple times decreasing the computational cost might be a better overall target for sustainability then being able to offer higher accuracies. Even though scaling using the EfficientNet compound method might not yield the best results using some EfficientNets what were optimised by the team to would be optimal, for example, EfficientNet-B1 is both small and efficient while still obtaining 79.1% top-1 accuracy in ImageNet, and realistically the datasets that this system will process will be smaller and more scope specific than ImageNet.

2.5 Machine learning libraries

While there are various machine learning libraries, the two bigger ones are Tensorflow and PyTorch. This section will compare the two different libraries. TensorFlow [19] is an open-source machine learning platform created by Google to develop their production and research systems. PyTorch [20] is an open-source machine learning library developed by Meta to power their systems.

While both libraries can achieve the same tasks with similar level of accuracy [21], PyTorch is mostly used in research oriented applications rather than applications that might require deployment [21, 22]. This is generally attributed to the maturity of TensorFlow and TensorFlow's ability to create static graphs, which are optimised for inference.

More important for the project is compatibility with other technologies that the project will use. In this case, TensorFlow has native support for Go while PyTorch does not. Which due to Tensorflow's advanced in deployment and compatibility the clear choice for the project.

2.6 Summary

The technical review of current systems, shows that there are current systems that exist that can perform image classification tasks, but they are not friendly in ways to easily expand currently existing models.

The current methods that exist for image classification seem to have reached a classification accuracy and efficiency that make a project like this feasible. Model architectures such as ResNet, and EfficientNet have been able to perform image classification on large sets of models and achieve higher than human performances. Taking these architectures in mind the system should be able to create machine learning models that perform equally well.

As for what technologies to use to build such models TensorFlow seams to be the correct choice as it has better performance when deploying to production, and can more easily integrate with the chosen web technologies.

3 Service Analysis and Requirements

Understanding the project that is being built is a critical step in the software deployment process. This section will discuss what are the requirements that a service needs to implement for the project to be considered a success.

As a software as a service project, there are some required parts that the project needs to have:

- A way for the user to interact with the system
- A way for programs to interact with the system
- Management of images
- Management of models
- Management of compute resources

3.1 Service Structure

The service has to be structured so that users can interact with in two ways.

The first way is for the user to directly interface with the system using a user interface. This interface does not have any strict form requirements, it could either be a desktop application, web application or even a command line application. The main objective of this interface is for the user to quickly understand what the system is doing with their data, and if they can use the model they created to evaluate images.

The second way for the user to interface with the system needs to be an API. This is required as it would not make sense for users to be able to quickly generate image classification models if they still had to evaluate all the images manually. Therefore, there needs to be away for the user product to connect with the system, the API provides exactly that.

The system should also be structured in a way that allows easy scalability. So that it can handle many requests at the same time. The system should be able to scale, this could be achieved in many ways. One way is by allowing the service to act as a cluster, where the same application is running multiple times and a load balancer, balances the load between the systems. Another way is for the service to behave as a distributed system, where the services are split into smaller modules and those modules can be replicated. Independently of how the system scales, it requires the ability to handle the fact that the data that the system uses not be available everywhere.

As a machine learning solution, the service requires the necessary computational power to handle the training of the models. This means that the system needs to be structured in an away that it can decouple the training process from the main process. Which guarantees that the compute requirements for training the model do not affect the main server running. Ideally, the service should be able to divide the tasks from tasks that would require the GPU, and tasks that would require the CPU.

3.2 Resources

As a machine learning image classification service, the service has to manage various types of resources.

3.2.1 Compute Resources

As mentioned before, the service needs to be able to manage its compute resources. This is required because, for example, if the system starts training a new model and that training uses all the GPU resources, it would impact the ability of the service to be able to evaluate images for other users. As this example demonstrated, the system needs to keep track of the amount of GPU power available, so it can manage the actions it has to take accordingly. Therefore, for optimal functionality, the service requires the management of various compute resources.

There should be a separation of the different kinds of compute power. The two types of compute power are: CPU and GPU. The CPU is needed to handle the multiple requests that the API might

answer at the same time. And the GPU resources are required to train models and evaluate the images.

As a result, the service needs a system to distribute these compute tasks. The tasks have to be distributed between the application that is running the API and the various other places where that compute can happen.

An ideal system would distribute the tasks intelligently, to allow the maximisation of resources. An example of this would be running image classification, on the same model, on the same place twice, this would allow the model to stay in memory and not need to be reloaded again from disk. These kinds of optimisations would help the system to be more efficient and less wasteful.

Another way to reduce the load that the system goes through is to allow users to add their own compute power to the system. That compute power would only use images and models that are owned by the user. While allowing the compute power to run any image or model in the system would allow for an even more scalable system, it would be an incredible violation of privacy and security. As it allows outsiders access to possible sensitive information. Which makes the idea of a complete distributed network of user provided compute power not viable.

3.2.2 Storage

Another resource that it has to handle is storage. As the service accepts user uploaded images, the service has to monitor how much storage those images take. The service will need systems to handle when the user uploaded images take too much space. There are many ways of handling this, such as allowing the user to store their images, compacting the images, deleting images that the system might no longer need, or allowing dynamic storage services such as Object Buckets.

If there is not enough space to store all the images from all the models, and the service needs to delete images. There should be a system that removes the images in a manner that causes the less harm. An example of this would be deleting images in a way that keeps the dataset balanced.

If there is a discrepancy of where compute and storage happen, the system needs to be able to handle that. This can be accomplished in various methods. The most aggressive one is not allowing to compute resources to access data that is far away. The less aggressive and smarter way is to allow the system to move data to the optimal place.

3.3 User interface

A service such as this requires a way for the users to quickly get an understating of how their data is being used and how they can perform the actions they want.

As previously mentions, this application can take multiple forms, from web apps, to command line applications. As long as the application is easy to use, and allows the user to perform the required tasks:

- Configure model
- Upload images
- Manage images
- Request model training
- Request image evaluation
- Configure access

The way that the application communicates with the service should be done, via the API. If there was a requirement to physical access the computer that the service is running on, it would defeat the purpose of this project. Therefore, being able to control the service via the API makes the most reasonable sense. A second system could be developed that allows the application to control the service, but that would be terribly inefficient. Allowing the application to control the system via the API, also improves the API, as the API now gets more features.

The application should also allow administrators of the service to control the resources that are available to the system, to see if there is any requirement to add more resources.

3.4 API

As a SaaS platform, most of the requests made to the service would be made via the API, not the user interface. This is the case because the users that would need this service would set up the model using the web interface and then do the image classifications requests via the API.

While there are no hard requirements for the user interface, that is not the case for the API. The API must be implemented as an HTTPS REST API, this is because the most of the APIs that currently exist online are HTTPS REST APIs [23]. If the service wants to be easy to use, it needs to be implemented in away such that it has the lowest barrier to entry. Making the type of the API a requirement would guarantee that the application would be the most compatible with other systems that already exist. The API would also need to be able to do all the tasks that the application can do. As it would allow a user who wants to interact with the service via the API the ability to do so. The API also requires authentication because without authentication it would allow users who might have malicious intent to:

- Modifying systems settings
- Accessing other users' data

Allowing such actions would be incredibly damaging for the system. Therefore, the API must implement authentication methods to prevent those kinds of actions from happening.

3.5 Data Management

The service will store a large amount of user data. This includes: user information, user images, user models.

User Information

There are no hard requirements on how the user information needs to be stored, as long as it is done securely. User information includes personal identifiable information such as username and email, and secret information such as passwords, and access tokens.

Future versions of the service could possible also store more sensitive information about the user, such, as payment information and addresses. Such information is required if the user needs to be charged, but payment for the services provided is outside the scope of this project.

User Images

Images are another kind of information that has to be stored. As it was mentioned before, the system has to keep track of the images and the space they use. The system should also guarantee that there is some level of security in accessing the images that were uploaded to the service.

Models

The last kind of data that the service has to keep track of is model data. Once the model is trained, it has to be saved on disk. The service should implement a system that manages where the models are stored. This is similar to the image situation, where the model should be as close as possible to the compute resource that is going to utilise it, even if this requires copying the model.

3.6 Summary

This section shows that there are requirements that need to be met for the system to work as indented. These requirements range from usability requirements, implementation details, to system-level resource management requirements. The most important requirement is for the system to be easy to use by

the user. As if it is difficult to use, then the service already fails in one of its objectives. The other requirements are significant as well, as without them, the quality of the service would be very degraded. And even if the service was effortless to use, it is as bad as being difficult to use if it could not process the images quickly in a reasonable amount of time. The next chapter will describe a design that matches a subset of the requirements.

4 Service Design

This chapter presents an idealised design, such design is open-ended to allow for multiple possible implementations that still meet the project requirements. This idealised design is also envisioned to not be limited by time or engineering constraints. The chapter 5 will discuss in more details how this design was further scoped to be able to be implemented in the timeframe available. This chapter will transform the requirements discussed in the previous chapter into a more specialized technical design that can be used as a guide to implement such a service.

4.1 Structure of the Service

The service is designed to be a 4 tier structure:

- Presentation Layer
- API Layer
- Worker Layer
- Data Layer

This structure was selected because it allows separation of concerns. The layers were separated based on what resources are required by that layer.

The presentation layer requires interactivity of the user, and therefore it needs to be accessible from the outside, and be simple to use. The presentation layer was limited from being any interaction method to be a web page. The web page can a separate server, or as part of the main API application, if it is in the same. The API layer, is one of the most important parts of the service. As it will be the most used way to interact with the service. The user can use the API to control their entire model process from importing, to classification of images. The Worker layer, consists of a set of servers available to perform GPU loads. The Data layer, consists of stored images, models, and user data.

4.2 Interacting with the service

As a software platform, this project requires a way for users to interact with the service. This interface is mainly intended to be as a control and monitoring interface. The user would use the interface to set up and manage the models, then most of the interactions would happen via the API.

While there were no specific restrictions on what the interface can be, it makes most sense for it to be a web page. This is because most software as a service applications are controlled with web pages, and the API is already a web-based application.

Independently of the kind of the application is, it needs to allow users to fully control their data in an easy to use and understand way. The application should allow users to:

- Manage access tokens.
- Upload images for training.
- Delete images.
- Request training model.
- Delete models.
- Classify Images.
- See previous classification results
- Keep track of model accuracy

Aside from being able to perform the above tasks, there are no restrictions on how the application needs to be architected.

4.3 API

As a SaaS, one of the main requirements is to be able to communicate with other services. The API provides the simplest way for other services to interact with this service.

The API needs to be able to perform all the tasks that the application can do, which include:

- Manage access tokens.
- Upload images for training.
- Delete images.
- Request training model.
- Delete models.
- Classify Images.
- See previous classification results
- Keep track of model accuracy

While implementing all the features that mentioned above, the API has to handle multiple simultaneous requests. Ideally, those requests should be handled as fast as possible. The API should be implemented such that it can be easily expandable and maintainable, so that future improvements can happen. It should be consistent and easy to use, information on how to use the API should also be available to possible users. The API should be structured as a REST JSON API, per the requirements. The API should only accept inputs via the URL parameters of GET requests or via JSON on POST requests. Binary formats can also be used to handle file upload and downloads, as transferring files via JSON extremely inefficient.

4.4 Generation of Models

The service should use any means available to generate models, such means can be:

- Templating.
- Transfer Learning.
- Database Search.
- Pretrained Models with classification heads.

4.5 Models Training

Model Training should be independent of image classification. A model training should not affect any current classification. The system could use multiple ways to achieve this, such as:

- Separating the training to different machines.
- Control the number of resources that training machine can utilise
- Control the time when the shared training and inference machine can be used for training.
- Allow users to have their own "Runners" where the training tasks can happen.

4.6 Summary

This chapter introduced multiple possible designs options for a service, that intends to achieve automated image classification, can follow to implement a robust system. The next chapter will be discussing how the system was implemented and which of the possible design options were chosen when implementing the system.

5 Service Implementation

This chapter will discuss how the service followed some possible designs to achieve a working system. The design path that was decided matches what made sense for the scale and needs of the project.

5.1 Structure of the Service

The structure of the service matches the designed structure, as it can be seen in Figure 1.

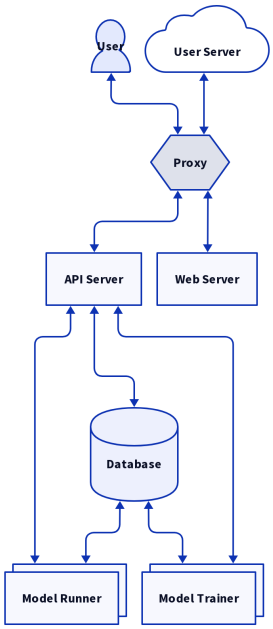


Figure 1: Simplified diagram of the service

The implementation contains: Web App; Web server, which serves the Web App; API; Training Runners; Model Runners. This differs from the designed solution, as it contains an extra nginx reverse proxy server [24]. The reverse proxy is requireed as it allows for the API and the webpage to be accessible from them same domain.

The selected database was PostgresSQL [25] as it is one of the most advanced open source databases available. The database stores all data required for the system to work, with the exeption of uploaded images and model files.

The rest of this chapter will discuss how each individual part of the system was implemented.

5.2 Web Application

The web application (WEB App) is the chosen user interface to control the service.

This subsection discusses details of the user flows and implementation of the application.

Implementation Details

The Web APP is a single-page application (SPA). The SPA architecture is one of the most prevalent architectures that exists nowadays. It allows for the fast transitions between pages without having a full reload of the website.

Since this implementation separated the API and the Web App, it makes the use of server-side rendering more complex and less efficient. As, the server would have to first request the API for information to build the web page and then send it to the users' device. Therefore, the system will

use client-side rendering only, allowing for the users' device to request the API directly for more information.

There exist currently many frameworks to create SPAs. I selected Svelte [26] for this project. I selected Svelte because it is been one of the most liked frameworks to work with in the last years, accordingly to the State of JS survey [27]. It is also one of the best performant frameworks that is currently available that has extremity good performance [28]. I also already have experience with Svelte.

I will be using Svelte with the SvelteKit framework [29] which greatly improves the developer experience. SvelteKit allows for the easy creation of SPAs with a good default web router. When deploying into a production enviroment the static adapter can be used to generate a static HTML and JavaScript files. This static files can be then hosted in a more efficient http server, other than the one running on NodeJs.

The web application uses the API to control the functionality of the service. This implementation allows users of the application to do everything that the application does with the API, which is ideal in a SaaS project. The communication with the API, when correcly consigured, uses HTTPS to make this communication encrypted and safe.

Service authentication

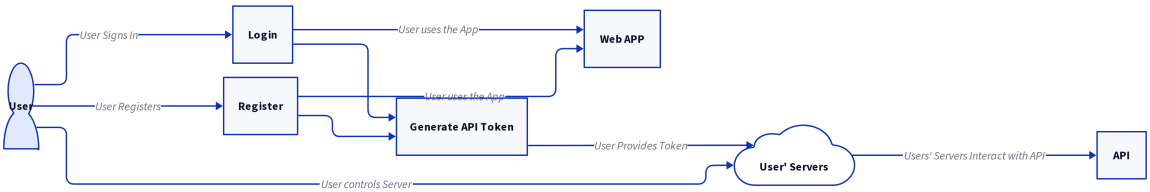


Figure 2: Simplified Diagram of User Authentication

The user uses an email and password to Sign In or Register with the application. This is sent to the server and stored in a user account. The Password is stored hashed using bcrypt [30]. In the future, other methods of authentication might be provided; like using Google's OAuth. Once logged In, the user will be able to use the application and manage tokens that were emitted for this application. This allows the user to manage what services have access to the account. On the web app, the user can manage existing tokens. Guaranteeing that only the clients that should be accessing the information are. In the management screen, which can be seen in Figure 3, the user can remove, and create tokens.

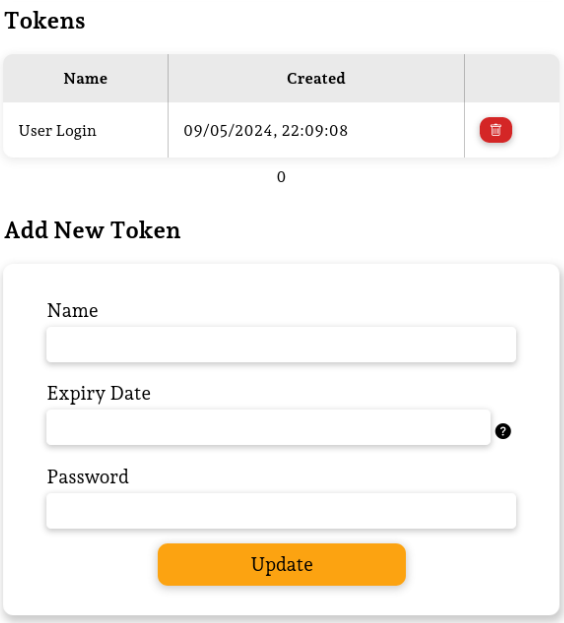


Figure 3: Screenshot of web application on the token control section.

Model Management

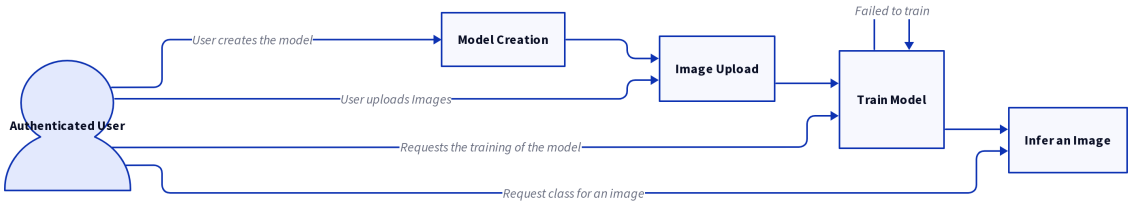


Figure 4: Simplified Diagram of Model management.

Figure 4 shows the steps that the user takes to use a model.

First, the user creates the model. In this step, the user uploads a sample image of what the model will be handling. This image is used to define what the kinds of images the model will be able to intake. This is done in the page shown in Figure 5, the user provides a name for the model and an image and then presses the button create.

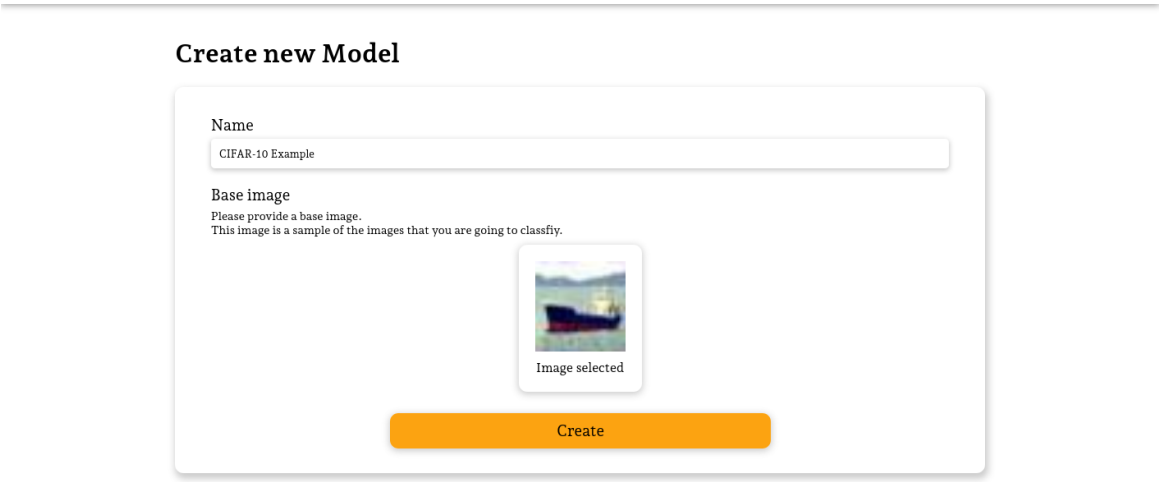


Figure 5: Screenshot of web application on the page that allows the creation of a new model.

The user is then shown the model page, which contains all the information about a model, which can be seen in Figure 6.

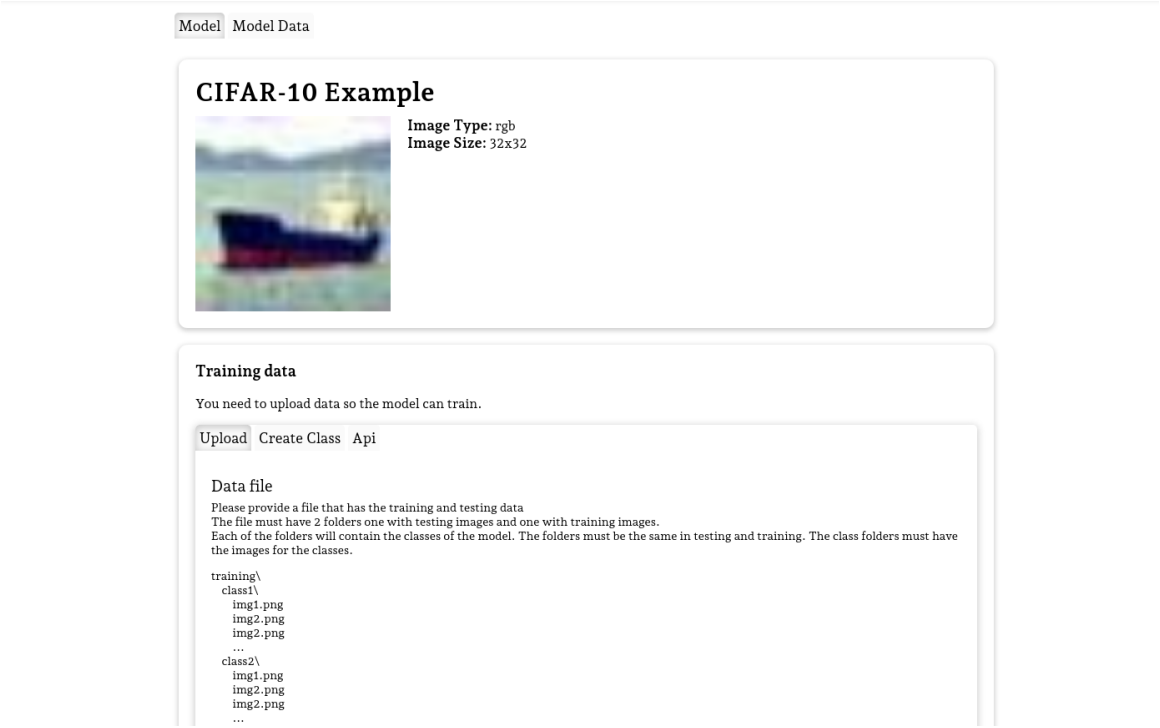


Figure 6: Screenshot of the web application on the page shows basic information about the model.

This page contains a set of tabs a top. Each tab gives different insight about the model. The "Model" tab, contains only relevant actions that the user can take. In Figure 6, the user has created a model but has not added training data, so the page shows a section where the user can input training data. The "Model Data" tab contains a more detailed view about data that has been updated.

Currently, the system does not support resizing of images that are different from the one uploaded at the creation step. This was done to guarantee that the image that the user wants to classify is unmodified. Moving the responsibility of cropping and resizing to the user. In the future, systems could be implemented that allow the user to select how an image can be cropped.

The second step is uploading the rest of the dataset. This can be done via the "Model" tab or via the "Model Data" tab that becomes available when the data of the model is first uploaded. In this tab, the user can add and remove images, as well as create new classes for the model. The page also

shows some useful information, such as the distribution of the dataset, which can be seen in 7.

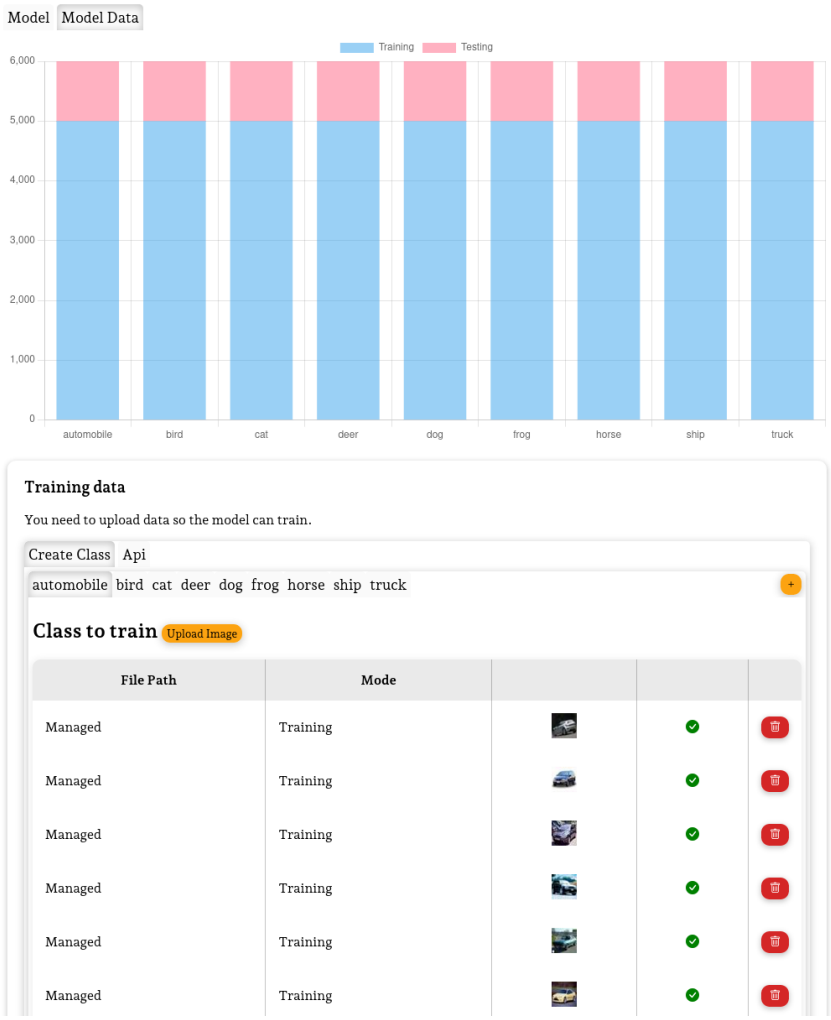


Figure 7: Screenshot of web application part of the "Model Data" tab.

This information can be useful to more advanced users that might decide to gather more data to balance the dataset. To upload the reset of the data set, the user can upload a zip file that contains a set of classes and images corresponding to that class. That zip file is processed and images and classes are created. The user is given instruction on how to create the zip file so that the system can easily process the data, the upload set can be seen in Figure 9. This process was original slow as the system did not have the capabilities to parallelize the process of importing the images, but this was implemented, and the import process was improved. The improved process now takes a few seconds to process and verify the entirety of the dataset, making the experience for the end user better. Alternatively, the user can use the API to create new classes and upload images.

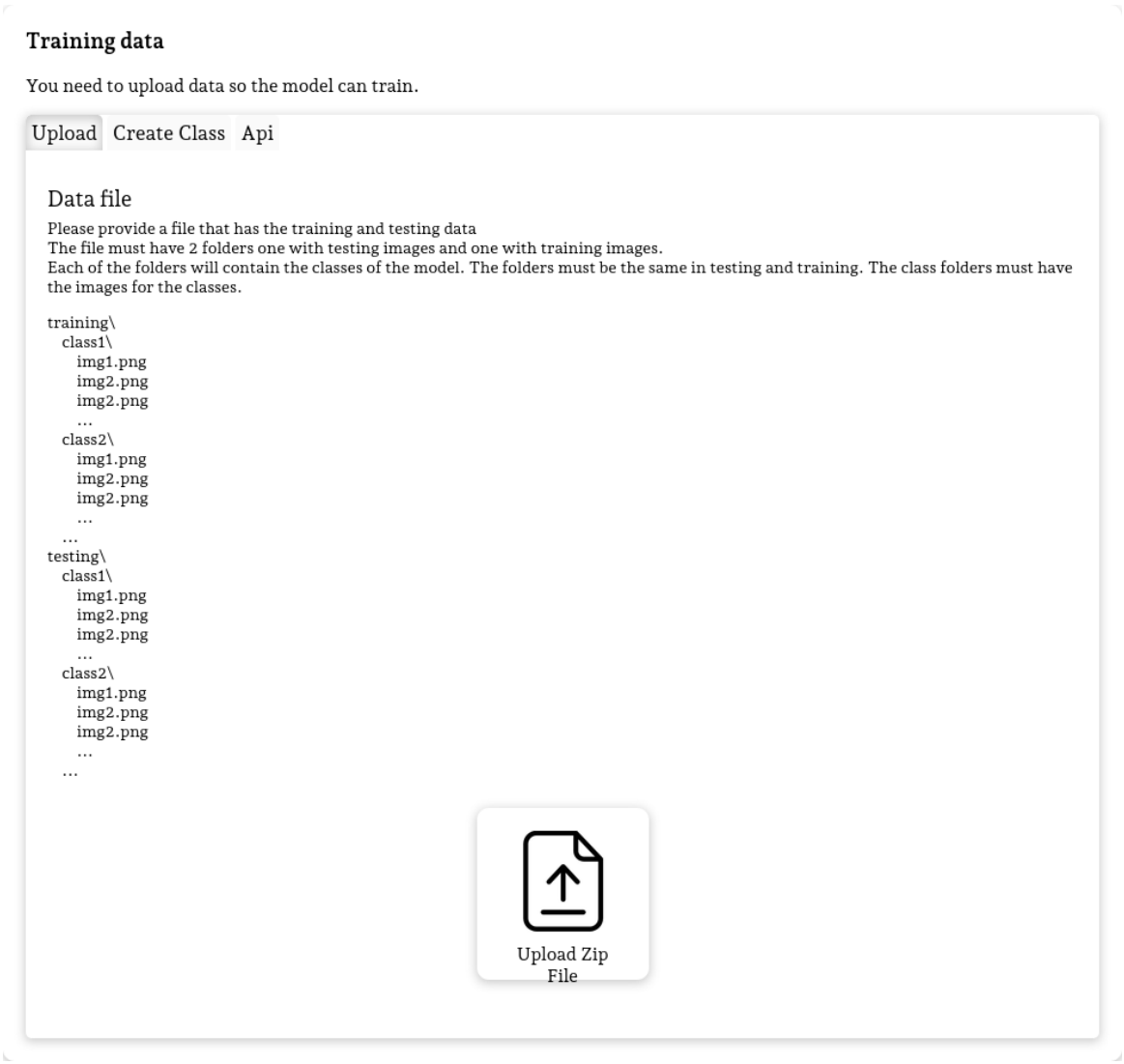


Figure 8: Screenshot of web application upload zip step of the model page.

After all the images that are required for training are uploaded, the user can go to the training step. This step will appear both in the main tab of the model page and in the dataset tab. Once the user instructs the system to start training, the model page will become the training page, and it will show the progress of the training of the model. During this step, the system automatically trains the model. After the system trains a model that meets the specifications set by the user, the service will make the model available for the user to use.

When the model is finished training, the user can use the model to run inference tasks on images. To achieve this, the user can either use the API to submit a classification task or use the tasks tab in the web platform.

In the tasks tab, which can be seen in Figure 9, the user can see current and previous tasks. The users can see what tasks were performed and their results. The user can also inform the service if the task that was performed did return the correct results. This information can be used to keep track of the real accuracy of the model. The information can be used to see if the model needs refinement. The system can add the classes that failed to return the correct result to a list of the original data, to be used in case of retraining the model.

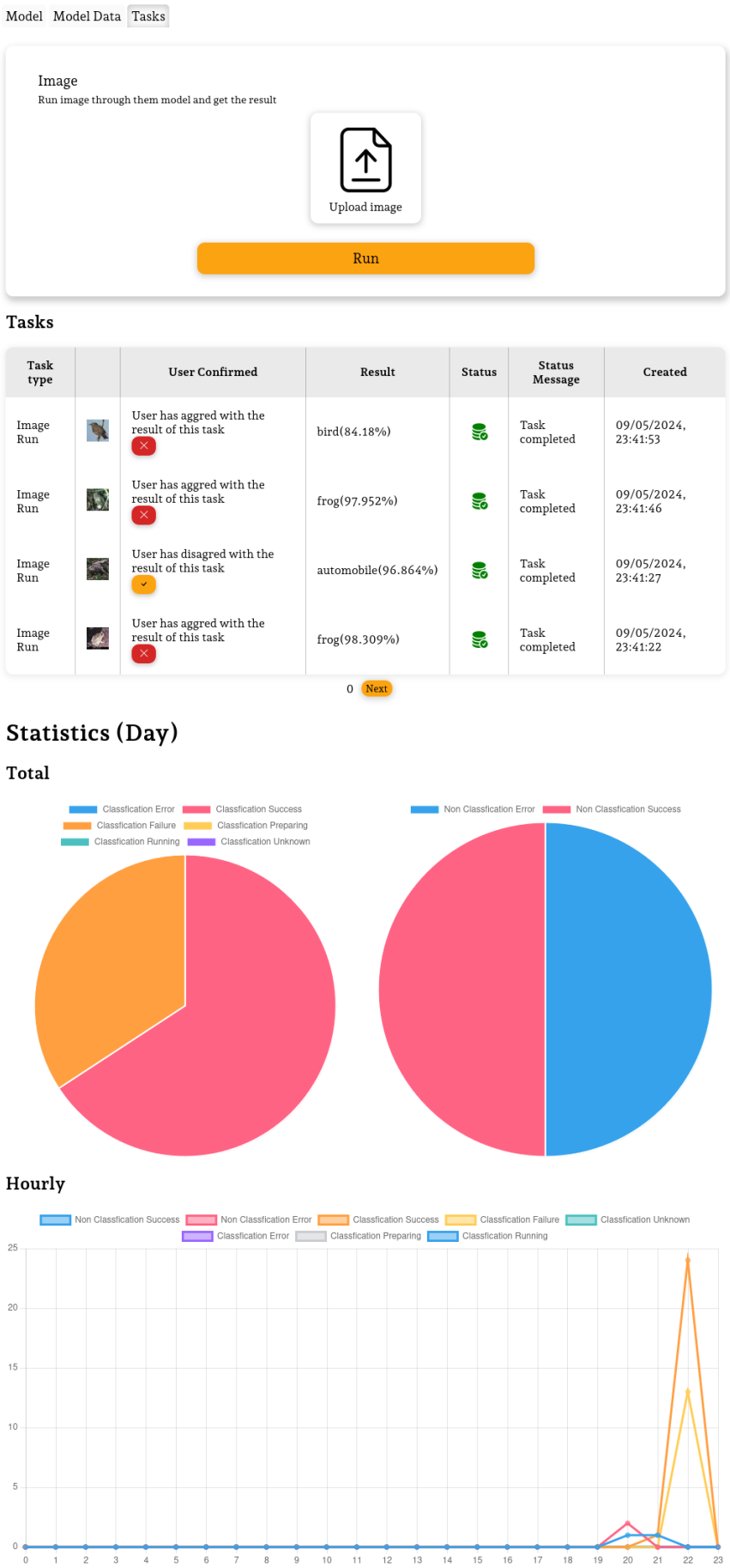


Figure 9: Screenshot of web application on the tasks tab.

Advanced Model Management

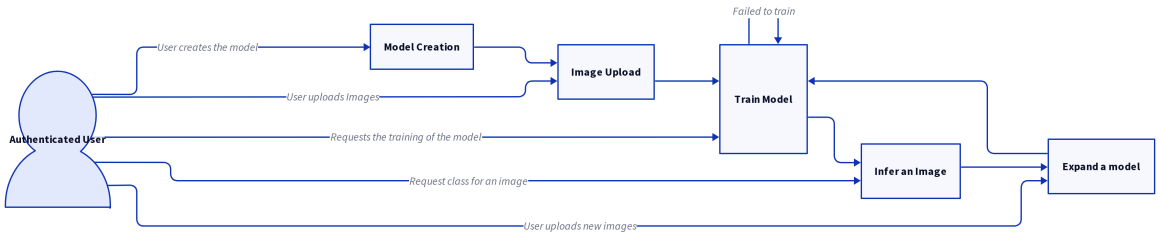


Figure 10: Simplified Diagram of Advanced Model management.

Figure 10 shows the steps that the user takes to use a model.

The steps are very similar to the normal model management. The user would follow all the steps that are required for normal model creation and training.

At the end of the process, the user will be able to add new data to the model and retrain it. To achieve that, the user would simply go to the data tab and create a new class, which the Figure 11 shows. Once a new class is added, the webpage will inform the user that the model can be retrained. The user might choose to retrain the model now or more new classes and retrain later.

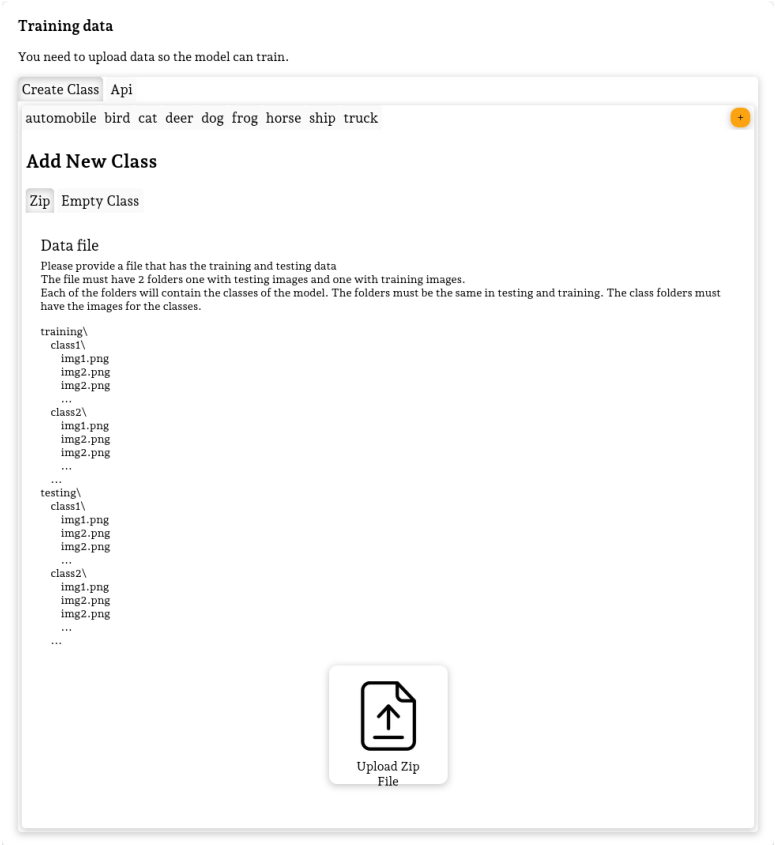


Figure 11: Screenshot of web application on the expand part of the “Model Data” tab.

During the entire process of creating new classes in the model and retraining the model, the user can still perform all the classifications tasks they desire.

Task Management

Task management is the section of the website is where uses can manage their tasks. This includes training and classification tasks.

Users in this tab can see what is the progress, and results of their tasks. The webpage also provides nice, easy to see statistics on the task results, allowing the user to see how the model is performing. Which is shown on Figure 9

On the administrator, users should be able to change the status of tasks as well as see a more comprehensive view on how the tasks are being performed. Administrator users can see the current status of runners, as well as which task the runners are doing, the Figure 12 shows the runner visualisation page.

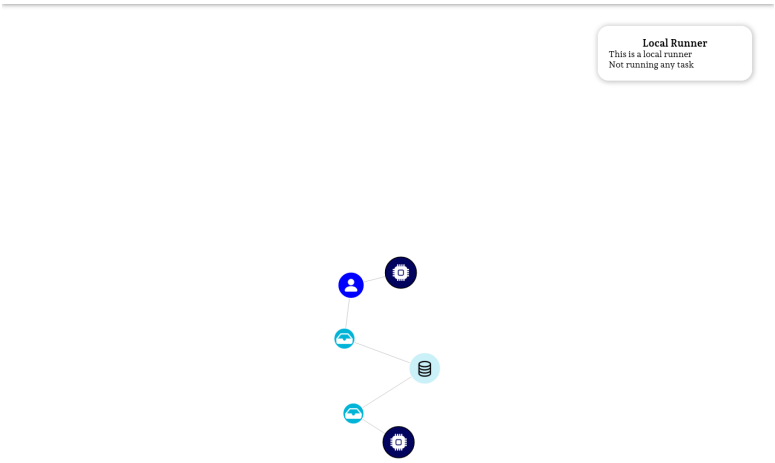


Figure 12: Screenshot of web application on the runner administration page.

5.3 API

The API was implemented as a multithreaded Go [31] server. The application, on launch, loads a configuration file and connects to the database. After connecting to the database, the application performs pre-startup checks to make sure no tasks that were interrupted via a server restart and were not left in an unrecoverable state. Once the checks are done, the application creates workers, which will be explained in section 5.6, which when completed the API server is finally started up.

Information about the API is shown around the web page so that the user can see information about the API right next to where the user would normally do the action, providing a good user interface. As, the user can get information about right where they would normally do the action, as it can be seen in Figure 13.

Figure 13: Screenshot of the web application that shows the explanation of the API call.

This server will take JSON and multipart form data requests, the requests are processed, and answered with a JSON response. The multipart requests are required due to JSON's inability to transmit binary data, which will make the uploading of images extremely inefficient. Those images would have to be transformed into binary data and then uploaded as a byte array or encoded as base64 and uploaded. Either of those options is extremely inefficient. Therefore, there is a need to use multipart form requests are required to allow the easy uploading of binary files.

Go was selected as the language to implement the backend due to various of its advantages. Go has extremely fast compilations which allows for rapid development, and iteration. It has a very minimal runtime which allows it to be faster, than heavy runtime languages such as JavaScript. It is also a simple language, which helps maintain the codebase.

The Go language integrates well with C libraries, which allows it access to machine learning libraries like TensorFlow or Lib Torch.

Authentication

The API allows users to login, which emits a token, and once logged in, manually create tokens. While using the web application, this is done transparently, but it can also be manually done via the respective API calls.

During the login process, the service checks to see if the user is registered and if the password provided during the login matches the stored hash. Upon verifying the user, a token is emitted.

Once a user is logged in they can then create more tokens as seen in section 5.2. While using the API the user should only use created tokens in the settings page as those tokens are named, and have controllable expiration dates. This is advantageous from a security perspective, as the user can manage who has access to the API. If the token gets leaked, the user can then delete the named token, to guarantee the safety of his access.

The token can then be used in the "token" header as proof to the API that the user is authenticated.

5.4 Generation and Training of Models

Model generation happens on the API server, the API server analyses what the image that was provided and generates several model candidates accordingly. The number of model candidates is user defined. The model generation subsystem decides the structure of the model candidates based on the image size, it prioritises the smaller models for smaller images and convolution networks with bigger images. The depth is controlled both by the image size and number of outputs, models candidates that need to be expanded are generated with bigger values to account for possible new values. It tries to generate the optimal size if only one model is requested. If more than one is requested then the generator tries to generate models of various types and sizes, so if there is possible smaller model it will also be tested.

Model training happens in a runner, more information about runners will be explained in section 5.6.

Model training was implemented using TensorFlow. Normally, when using go with machine learning, only the prediction is run in go and the training happens in python. The training system was implemented that way.

The runner, when it needs to perform training it generates a python script tailored to the model candidate that needs to be trained, then runs the that python script, and monitors the result of the python script. While the python script is running, it takes use of the API to inform the runner of epoch and accuracy changes.

The during train, the runner takes a round-robin approach. It trains every model candidate for a few epochs, then compares the different models candidates. If there is too much difference in accuracy, from the best model to the worst model, then the system might decide not to continue training a certain candidate and focus the training resources on candidates that are performing better. Once one candidate archives the target accuracy, which is user defined, the training system stops training the models candidates. The model candidate that achieved the target accuracy is then promoted to the model, and the other candidates are removed. The model now can be used to predict the labels for any image that the user decides to upload.

Expandable Models

Expandable models follow mostly the same process as the normal models. First, bigger model candidates are generated. Then the models are training using the same technic. At the end, after the model candidate has been promoted to the full model, the system starts another python process that loads the just generated model and splits into a base model and a head model.

With these two separate models, the system is now ready to start classifying new images.

Expanding Expandable Models

During the expanding process, the generation system creates a new head candidate that matches the newly added classes. The base model, that was created in the original training process, is used with all available data to train the head candidate to perform the classification tasks. The training process is similar to the normal training system, but this uses a different training script. Once the model has finished training and the system meets the accuracy requirements, then makes the new head available for classification.

5.5 Model Inference

Model inference also runs inside a runner. However, inference runs internally on go, instead of creating tailored scripts for training using python.

Once a classification request is received by the API, the uploaded image is checked to see if the model will accept it. If the model is capable of accepting the image, it is temporarily saved to disk and then a classification task is created.

Eventually, a runner will pick up the classification task. Upon pickup, the runner will load the model and run the image through it, the results are then matched with the stored classes in the

database. The system then stores the class with the highest probability of matching the image, according to the model, in the results of the task.

The user then can finally use the API to obtain the results of the model.

Expandable Models

For expandable models, the inference step is very similar to the normal models. The runner first loads the base model and runs the image through the model, the resultant features are then stored. Once the features are obtained, the system then runs those features to the various possible heads, and the results are then matched with the stored classes, and the one with the height probability is then selected.

5.6 Runner

Runners are the name used to reference to the software that can perform CPU or GPU intensive tasks without halting the main API server.

Architecturally, they were implemented as a controller and worker pattern. When the application that runs the main application starts, the system creates an orchestrator, this orchestrator is a piece of software that decides what work each runner is doing. The orchestrator runs on go routine created at startup. During the startup, the orchestrator by obtaining values from the configuration file. Those values define the number of local runners.

These runners, which are started up by the orchestrator, act as local runners, runners that are running on the same machine as the main server. Local runners are useful when running the main server on a machine that also has GPU power available to it, and in testing.

Local runners, run inside a go routine, this allows the runners and the orchestrator to communicate using go channels, which are the easiest way to communicate between two go routines. The orchestrator is constantly waiting to receive either for a timer-based event or a runner-based event.

Timer-based events happen when the orchestrator internal clock informs it of needing to check if tasks are available to run. The time at which this clock ticks in configured in the settings of the app. Upon receiving a timer-based event, the orchestrator then checks if there is a new task available for it to run and if there are any runners available for the task to run on. If there are tasks available, then the orchestrator instructors the runner to pick up the task and run it.

Runner-based events happen when a runner finishes running a task or crashes while trying to do it. Upon receiving a runner event, the orchestrator checks if it is a success or a failure message.

If it is a failure message and the runner is a local runner, then the orchestrator just restarts the runner. Upon restart, it adds the runner to the list of available runners. If the runner is a remote runner, the orchestrator marks the runner as failed and stops sending messages to the runner until the runner informs the service again that is available.

If the message is of success, then the orchestrator just adds the runner to the list of viable runners, independently if the runner is remote or not.

5.7 Summary

This chapter went into the details of how the designed was implemented. The design was envisioned to be the best possible version of this service, but scope was restrained to the necessities of the system while it was being developed. And possible features that would make the implemented application closer to the ideal design could have been implemented if there was higher need during the development timeline. This will be more discussed in chapter 8.

6 Legal, Societal, Ethical and Professional Considerations

This section will address possible legal, societal, ethical and professional issues that might arise from the deployment of the software being designed.

The Self-Assessment for Governance and Ethics (SAGE) form has addressed, and it is submitted along with the report.

6.1 Legal Issues

Legal issues can occur due to the data being stored by the service.

The service collects, the least amount of sensitive information, from the users who directly use the service. That data that is collected while being sensitive is required to be able to authenticate the user, such as name, email, and password. To safeguard that information, the system will be using industry standards to guarantee data security of that data.

Legal issues might occur due to image uploaded images. For example, those images could be copyrighted, or the images could be confidential. The service is designed to provide ways to allow users to host their images without having to host the images itself, moving the legal requirement to the management of the data to the user of the system.

GDPR

The General Data Protection Regulation (GDPR) (GDPR, 2018) is a data protection and privacy law in the European Union and the European Economic Area, that has also been implemented into British law. The main objective of the GDPR is to minimise the data collected by the application for purposes that are not the used in the application, as well as giving users the right to be forgotten.

The application collects only personal data needed to authenticate the user, and data that is generated during the normal usage of the application.

All the data that is related to the user can be deleted. The system will prevent any new work that is related to the data, that was requested to be deleted. Once the there is no more work that requires the data being done, the system will remove all relevant identifiable references to that data.

6.2 Social Issues

The web application was designed to be easy to use and there tries to consider all accessibility requirements.

6.3 Ethical Issues

While the service itself does not raise any ethical concerns. The data that the service will process could raise ethical complications.

For example, if the service gets acquired by a company that also wants to use the data provided to the system for other reasons.

6.4 Professional Issues

As a member of the British Computer Society (BCS), it is important to follow the Code of Conduct practices. The code of conduit contains 4 key principles.

Public interest

This project tries to consider the public health, privacy, and security of third parties and therefore follows the principle of public interest.

Professional Competence and Integrity

This project has been an enormous undertaking that pushed the limits of my capabilities. I am glad that I was able to use this opportunity to learn about distributed systems, image classification, go, and Svelte. During this project, I also followed the best practices of software development, such as using source control software and having an audit to tasks and issues.

Duty to Relevant Authority

For the duration of the project, all the guidelines provided by the University of Surrey were followed.

Duty to the Profession

During the research, design, and implementation, and report state all interactions with the supervisor of the project have been professional, respectful, and honest. To the best of my ability, I tried to design a system that would contribute to the field.

7 Service Evaluation

This section will discuss how the service can be evaluated from a technical standpoint and its results.

With the goals of the project, there are two kinds of tests that need to be accounted for. User testing tests that relate to the experience of the user while using the project and tests that quantitive test the project.

Such as accuracy of the generated models, response time to queries.

7.1 Testing the model creation

To test the system, a few datasets were selected. The datasets were selected to represent different possible sizes of models, and sizes of output labels.

The ImageNet[10] was not selected as one of the datasets that will be tested, as it does not represent the target problem that this project is trying to tackle.

The tests will measure:

- Time to process and validate the entire dataset upon upload
- Time to train the dataset
- Time to classify the image once the dataset has been trained
- Time to extend the model
- Accuracy of the newly created model

The results will be placed in the results table.

MNIST

The MNIST [7] is a large dataset of handwritten digits, that is commonly used to trains and test machine learning systems. This dataset was selected due to its size. It is a small dataset that can be trained quickly and can be used to verify other internal systems of the service. During testing, only the 9 out of 10 classes are trained and the 10th is added during the retraining process.

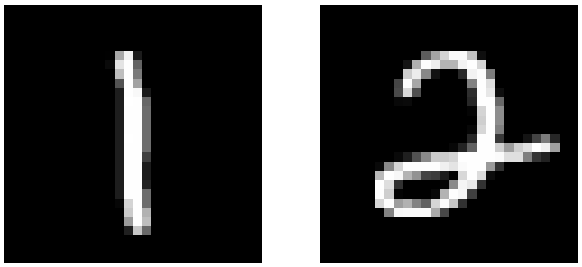


Figure 14: Examples of the images in the MNIST dataset

CIFAR-10

The CIFAR-10 [32] dataset contains various images that are commonly used to train and test machine learning algorithms. This dataset was selected due to its size. It is a small dataset that can be trained quickly, but it has bigger, and coloured images, which makes it harder than MNIST.

During testing, only the 9 out of 10 classes are trained and the 10th is added during the retraining process.

Figure 15: Examples of the images in the CIFAR-10 dataset

STL-10

The STL-10 [33] dataset that was inspired by the CIFAR-10 [32], but it has bigger images. This dataset was selected because of the bigger image. The images are bigger than both CIFAR-10 and MNIST which makes the model harder to create, and train.

During testing, only the 9 out of 10 classes are trained and the 10th is added during the retraining process.

Figure 16: Examples of the images in the STL-10 dataset

ArtBench

The ArtBench [34] dataset is a dataset that contains artworks annotated with their art style that is intended to train generative models. This dataset was selected due to the even bigger images than the previously tested models.

During testing, only the 9 out of 10 classes are trained and the 10th is added during the retraining process.

Figure 17: Examples of the images in the ArtBench dataset

Incompatible datasets

There were attempts to test other datasets against the system, but those datasets were incompatible. The datasets had irregular images sizes, which, as it was mentioned previously, the system does not support. This caused a large section of images inputted being rejected, which means that it would have not trained.

A list of datasets that are incompatible because of this are:

- Caltech 256 [35]
- FGVC-Aircraft [36]
- IFood 2019 [37]

Results

Dataset	Import Time	Train Time	Classification Time	Extend Time	Accuracy
MNIST	$8s$	$2m$	$> 1s$	$50s$	98%
CIFAR-10	$6s$	$41m38s$	$> 1s$	$1m11s$	95.2%
STL-10	$1s$	$37m50s$	$> 1s$	$1m10s$	95.3%
Art Bench	$10s$	$4h20m31$	$> 1s$	$1m41s$	41.86%

Table 2: Evaluation Results

The system was able to easily import all the datasets provided in an incredibly fast time, this included the incompatible datasets. While the system was able to load and verify the images of the incompatible datasets, it correctly marked the images as incompatible, which can be seen in Figure 18. Which would make them not being able to be used for training, which would mean the model would have not had any data to train, which would obviously result in terrible accuracy results.

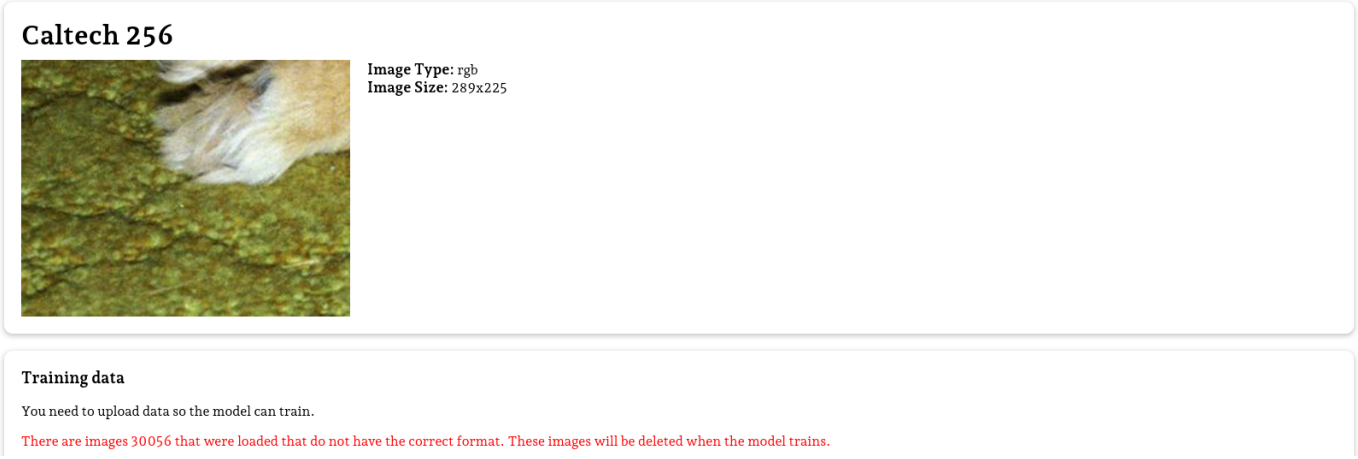


Figure 18: Screenshot of a web application showing many images that do not have the correct format.

The system was able to train, classify, and extend the MNIST, CIFAR-10, and STL-10 datasets, with high accuracy rates. This is expected as these models are models that are commonly known for being easy to train. The system could also train these models in a relatively short, small amount of time. The classification time is optimal, with all datasets being able to classify an image in less than a second. The time to extend is also very promising, and the system could extend a new set of classes fairly quickly.

The system was unable to achieve a high level of accuracy while training for the ArtBench dataset. And the training time to achieve that lower level of accuracy was also much higher than the other datasets. The longer training time can be attributed to the larger images, which make the model harder to train, as the model has to make more computations. Another factor for the increased training time is the necessity for the model to train longer to achieve a higher accuracy, due to the model's decreased learning rate. As for the low accuracy ratting, I hypothesise that this is caused by the nature of the dataset. The dataset is categorized into various art styles. Even within a single art

style, artists' individual styles can vary significantly. Given the relatively small sample size of only 5000 training images per art style, this variability poses a challenge for the model's ability to discern between distinct styles. Another option is that the system did not generate a good enough model for this dataset. The system was still able to fairly quickly classify and image, with the classification time still being under less than a second. The expansion time was also fairy quick, being on par with the other models.

Testing limitations

There are some limitations caused by this testing. The biggest problem is in the training, classification and expansion timings, this value will depend on what hardware the system that is running the model has. The small sample size of the datasets is also limiting, as it does not fully prove that the system can create generalized models.

7.2 API Performance Testing

The application performance was also tested. To test the performance of the API, a small program was written that would simultaneously request an image to be classified. The selected image was one of the sample images provided in the MNIST dataset. The program tries to perform 1, 10, 100, 1000, 10000 simultaneous requests, and waits 2 seconds between each set. The program would then record how much time it would take for the image classification task to be completed. And after all requests are completed, the program call calculates the mean and max requests times.

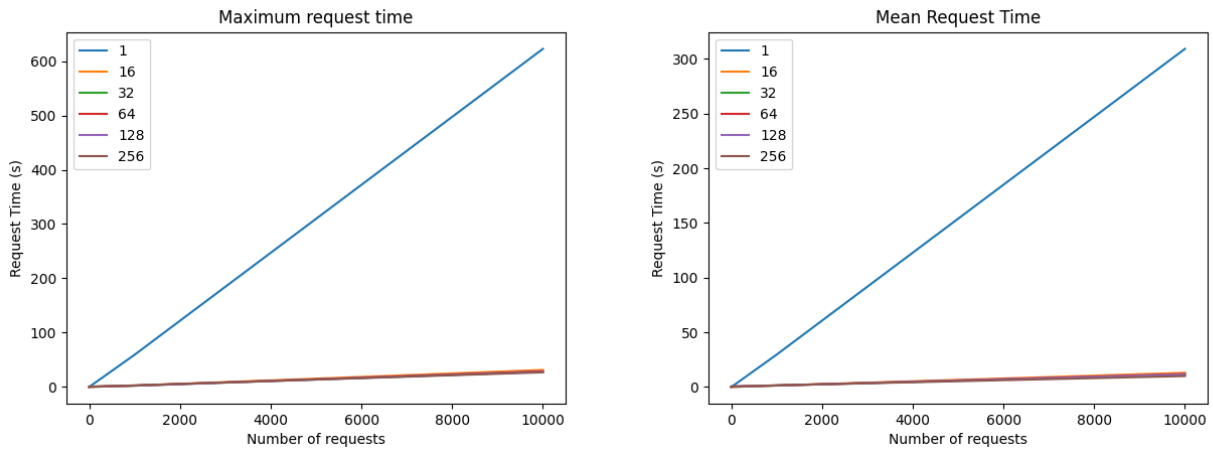


Figure 19: Results of the API testing

The values shown in Figure 19 show that if you configure the system to only have one runner, it will struggle to handle large amounts of simultaneous requests. This is expected, as only having one process trying to classify large amounts of images would be unwise. In reality this would never be set up this way since only having one runner in a production environment would never be acceptable.

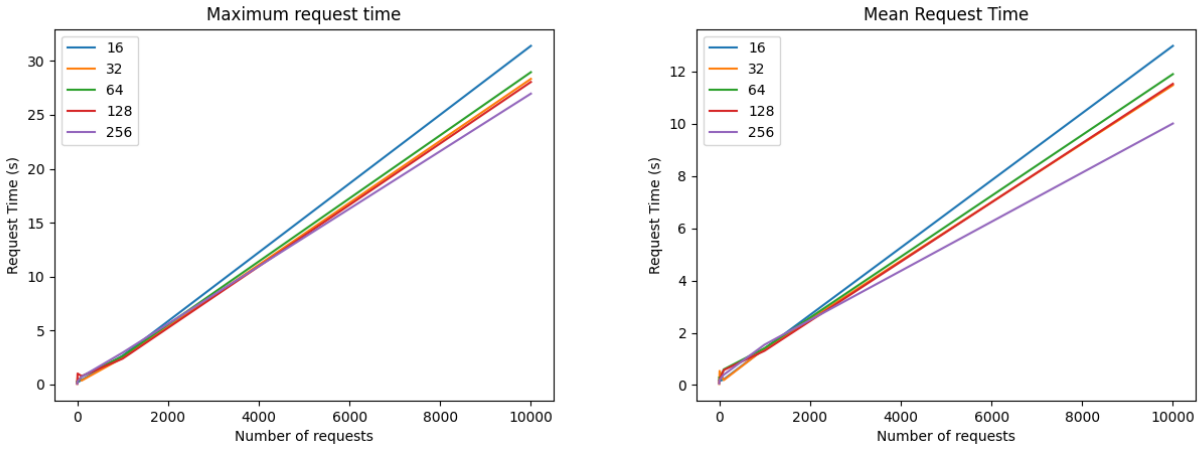


Figure 20: Results of the API testing

Figure 20 shows the same graph as Figure 19 but with the results for the test where the API only had one runner removed. The graph indicates that the system was able to handle, 10000 simultaneous requests in less than 30 seconds, which more than exceeds the expectations of the project. The results also indicate that the numbers of runners have demising returns, as the values maximum and mean request time are within a small range. This can be caused by multiple reasons. One such reason is that were not enough requests to show a significant difference between then number of runners. Another reason is that the amount of work that the system has to perform to manage all the runners outweighs the benefits of having more runners.

While testing, the ram usage was monitored but not recorded. As expected, the memory usage significant increased with the number of runners, but did not exceed 5 GiB. The higher memory usage is a result of the runners caching the model used. The memory footprint of the system limited by the model selected as the model generated for MNIST dataset is not large. And larger models are expected to generate larger memory footprints. When deploying the application, an administrator should take considerations the expected model sizes as well as the usage frequency expected and configure the application accordingly.

These results are very positive since the project was running on my personal computer and not on professional server hardware. This indicates that when deployed to a production environment the service is most likely to perform extremely well.

Testing limitations

As with the previous testing, this test has also some limitations. Including the same hardware limitation where different hardware will give different results for this test. Another limiting factor is that the test did not use different models or images which could cause the service to have to reload models from disk, affecting performance.

7.3 Usability

While if a service is usable differs vastly from user to user, the implemented system is simple enough where a user who does not know anything about image classification could upload images, and obtain a working classification model. This simplicity may pose limitations for users with advanced knowledge, which would fall short of optimal usability standards for that user. As this user might choose not to use the system because it does not allow the level of control that they might want.

The administrator area is less user-friendly than the rest of the application, but that is less critical. An administrator is not the target user of the application, and is expected to manage this system, which requires prior knowledge about the system.

7.4 Summary

The service can create models, and train models that service the user's needs. These models will most likely be able to achieve high accuracy targets, but in some cases the system might fail to generate a good enough model for the provided dataset. During testing, the limitations of the strict image size requirements were also shown, as the system, would have failed to train those datasets because most of the images would have been removed before the model started training.

While classifying images, the service performed extremely well. The API performance tests showed that if configured correctly, a single server configuration can handle a large amount of simultaneous images extremely fast. These results indicate that the system has the performance required to be put in a production environment and perform well.

As for the usability of the service the system, the system is usable by beginners, but might detract more advanced users from using it.

Overall, the service evaluation is positive, as the system was able to create and train new models, as well as being user-friendly to users who might not have the skills to perform image classification.

8 Critical Review of Project Outcomes

This chapter will go into details to see if the project was able to achieve the goals set forth in the introduction. The chapter will be analysing if the goals of the project were met, then shortcomings and improvements off the implementation will be discussed. After analysing shortcomings and improvements, possible future work that be done to the project will be discussed. The section will end with a general statement about the state of the project.

8.1 Project Objectives

In the introduction section of this project, some objectives were set for this project.

By the end of the project, the developed solution can achieve the goals set forth.

A system to upload images that will be assigned to a model

This goal was achieved. One of the abilities of both the API and the webpage are to be able to upload images to the service. Which means that a system was created that allows users to upload images that will be linked with a model.

A system to automatically train and create models

This goal was achieved. The designed server can create models based only using the data provided by the user without any human interaction. The model creation system is not as efficient, this inefficient will be discussed more in a future subsection it could be but can still achieve the desired goal.

Platform where users can manage their models

This goal was achieved. A web-based platform was developed where users can manage all the data related to machine learning models that were created. The platform that was implemented allows users to create models, upload images related to the model, and then manage the submitted classification tasks.

The platform allows managing any models easily they create with within, meaning that the developed solution can achieve the first goal of the project.

A system to automatically expand models without fully retraining the models

This goal was achieved. A system was created that allows users to add more images and classes to models that were previously created. And this is done without having to fully retrain the model.

An API that users can interact programmatically

This goal was achieved. The API implementation allows users to programmatically access the system. The efficacy of the API is proved by its use in the front end application. The front end application uses the API to fully control the service. This means that everything that can be done in the frontend can be done via the API. Which means that the API can satisfy every need that a possible user might have; therefore this goal was accomplished.

8.2 A retrospective analysis of the development process

This project was complex to implement, with many interconnected systems working together to achieve the goals of the project. This complexity was a result of open-ended design and scope expansion. If the scope of the project had been more limited, the project could have achieved higher overall results.

While there were no technical setbacks done during the development process. There were times when software updates of libraries made the implementation unusable, which slowed considerably the development velocity, as those issues required fixing. If actions such as creating OCI containers were done in the earlier stages of development, issues such as this could have been prevented. One of these

software updates, made it so that images were not being able to classified. This then prompted me to try to use a different library to train and classify the images, but this ended up not being achievable, and ended up with just fixing the original library problem. While the time used to try to integrate the different machine learning library helped the project improve, most of the effort put into this possible transition as spent inefficiently.

As far as tools aiding the development, this project followed industries norms by having the source code tracked in Git and issues tracked in an issue tracker. Which greatly helped in the development process, by having one centrailized repository of both code and known issues of that code.

8.3 Project Shortcomings and Improvements

Although the project was able to achieve the desired goals, the project has some shortcomings that can be improved upon in future iterations. This section will analyse some of those shortcoming and ways to improve the service.

Model Generation

The model generation system is a complex, and due to all the moving parts that make the system work, it requires a large amount of to work to maintain. It is also very inefficient due to the having to generate custom tailored python scripts, that cause the data to be reloaded every time a new a round-robin round needs to happen.

A way more efficient way is to perform all the training directly on go server. Running the training directly in go would allow the service to be able to keep track of memory and GPU usage, move data from the GPU and CPU effortlessly between runs, and would remove uncertainty from the training system.

The model generation was originally implemented with TensorFlow, this ended up limiting the generation of the models in go as the bindings for TensorFlow were lacking in the tools used to train the model. Using Lib Torch libraries would allow more control over data, and allow that control to be done in go, which would improve both control and speed of the process. Unfortunately, when a version of the service was attempted to be implemented using Lib Torch, the system was too unstable. Problems were encountered with the go bindings for the Lib Torch library or, the Lib Torch library was causing inconsistent behaviour with between runs. That compounded with time limitations make it impossible for a Lib Torch implementation to come to fruition. Having a full go implementation would make the system more maintainable and fast.

Image storage

The image storage is all local, while this does not currently affect how remote runner works. This is less problematic when the runner is on the same network as the main server, but if a possible user would like to provide their runners. This would require a lot of bandwidth for the images to be transferred over the network every time the model needs to run.

A better solution for image storage would allow user provided runners to store images locally. During the upload time, the API, instead of storing the images locally, would instruct the users' runner to store the images locally, therefore when the runner would need to perform any training tasks with local data instead of remote data.

This would not also not require modification of the current system. The system was designed and implemented to be expanded. The dataset system was designed to be able to handle different kinds of storage methods in the future, such as remote storage and Object Buckets, like Amazon S3.

User Interface

The user interface is simplistic, this helps new users use the program but limits what advanced users might want to do. The user interface also might need an overhaul as it not visually appealing. A future improving for this project is definitely getting a professional graphical designer that can create a better-looking and recognizable application.

8.4 Future Work

This section will consider possible future work that can be built upon this project.

Image Processing Pipelines

The current system does not allow for images of different sizes to be uploaded to the system, an interesting project would be to create a new subsystem that would allow the user to create image processing pipelines.

This new system would allow users to create a set of instructions that images would go through to be added to the system. For example, automatically cropping, scaling, or padding the image.

A system like this would add versatility to the system and remove more work from the users of the service as they don't have to worry about handling the image processing on their side.

Different Kinds of Models

The runner system could be used to train and manage different kinds of models, not just image classification models.

If the system was modified to have different kinds of models, it would allow the users to run different kinds of models. Such as Natural Language Processing Models, or Multi Model Models. This would increase the versatility of the service, and it would allow users to automate more tasks.

8.5 Conclusion

With the increase in automation recently, having a system that allows users to quickly build classification models for their tasks, would be incredibly useful. This project provides exactly that, a simple-to-use system that allows the user to create models with ease.

The implemented system is able to accept images provided by the user, then create and train that model, and then allow user to classify the images with that created model. To achieve this, the developed software is large and complex. Developing such large and complex systems comes with compromises. In this case the model generation, training, and classfication; and API systems were prioritized over other systems such as file management systems.

While there are still improvements that can be made, and more features, that can be added to the service to make it event better, such as image processing pipelines and diferent kinds of models. The service is in a state that It could be deployed in a production enviroment and work. Therefore this project is successful.

9 References

[1] *What Is Amazon Rekognition? (1:42)*, [Online; accessed 18. Dec. 2023], Dec. 2023. [Online]. Available: https://aws.amazon.com/rekognition.

[2] *What is Amazon Rekognition Custom Labels? - Rekognition*, [Online; accessed 18. Dec. 2023], Dec. 2023. [Online]. Available: https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/what-is.html?pg=ln&sec=ft.

[3] *Training an Amazon Rekognition Custom Labels model - Rekognition*, [Online; accessed 18. Dec. 2023], Dec. 2023. [Online]. Available: https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/training-model.html#tm-console.

[4] Google. "Vision AI — google cloud." (2023), [Online]. Available: https://cloud.google.com/vision?hl=en.

[5] *Pricing | Vertex AI Vision | Google Cloud*, [Online; accessed 20. Dec. 2023], Dec. 2023. [Online]. Available: https://cloud.google.com/vision-ai/pricing.

[6] *Product Recognizer guide*, [Online; accessed 20. Dec. 2023], Dec. 2023. [Online]. Available: https://cloud.google.com/vision-ai/docs/product-recognizer.

[7] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[8] S. An, M. J. Lee, S. Park, H. Yang, and J. So, "An ensemble of simple convolutional neural network models for MNIST digit recognition," *CoRR*, vol. abs/2008.10400, 2020. arXiv: 2008.10400. [Online]. Available: https://arxiv.org/abs/2008.10400.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[12] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "Eca-net: Efficient channel attention for deep convolutional neural networks," *CoRR*, vol. abs/1910.03151, 2019. arXiv: 1910.03151. [Online]. Available: http://arxiv.org/abs/1910.03151.

[13] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. arXiv: 1905.11946. [Online]. Available: http://arxiv.org/abs/1905.11946.

[14] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[15] C. Szegedy *et al.*, *Going deeper with convolutions*, 2014. arXiv: 1409.4842 [cs.CV].

[16] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[17] M. Tan and Q. V. Le, *Efficientnet: Rethinking model scaling for convolutional neural networks*, 2020. arXiv: 1905.11946 [cs.LG].

[18] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[19] Martín Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[20] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[21] *PyTorch vs TensorFlow: Deep Learning Frameworks [2024]*, [Online; accessed 14. May 2024], Dec. 2023. [Online]. Available: https://www.knowledgehut.com/blog/data-science/pytorch-vs-tensorflow.

[22] R. O'Connor, "PyTorch vs TensorFlow in 2023," *News, Tutorials, AI Research*, Apr. 2023. [Online]. Available: https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023.

[23] K. Hnatyuk, "130+ API Statistics: Usage, Growth & Security," *MarketSplash*, Oct. 2023. [Online]. Available: https://marketsplash.com/api-statistics.

[24] *Advanced Load Balancer, Web Server, & Reverse Proxy - NGINX*, [Online; accessed 12. Mar. 2024], Feb. 2024. [Online]. Available: https://www.nginx.com.

[25] *PostgreSQL*, [Online; accessed 14. May 2024], May 2024. [Online]. Available: https://www.postgresql.org.

[26] *Svelte • Cybernetically enhanced web apps*, [Online; accessed 12. Mar. 2024], Mar. 2024. [Online]. Available: https://svelte.dev.

[27] *State of JavaScript 2022: Front-end Frameworks*, [Online; accessed 12. Mar. 2024], Nov. 2023. [Online]. Available: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks.

[28] *Interactive Results*, [Online; accessed 12. Mar. 2024], Mar. 2024. [Online]. Available: https://krausest.github.io/js-framework-benchmark/current.html.

[29] *SvelteKit • Web development, streamlined*, [Online; accessed 12. Mar. 2024], Mar. 2024. [Online]. Available: https://kit.svelte.dev.

[30] N. Provos and D. Mazieres, "A future-adaptable password scheme," Mar. 2001.

[31] *The Go Programming Language*, [Online; accessed 1. Nov. 2023], Nov. 2023. [Online]. Available: https://go.dev.

[32] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[33] *STL-10 dataset*, [Online; accessed 11. May 2024], Nov. 2015. [Online]. Available: https://cs.stanford.edu/~acoates/stl10.

[34] P. Liao, X. Li, X. Liu, and K. Keutzer, "The artbench dataset: Benchmarking generative models with artworks," *arXiv preprint arXiv:2206.11404*, 2022.

[35] G. Griffin, A. Holub, and P. Perona, *Caltech 256*, Apr. 2022. DOI: 10.22002/D1.20087.

[36] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," Tech. Rep., 2013. arXiv: 1306.5151 [cs-cv].

[37] P. Kaur, K. Sikka, W. Wang, s. Belongie, and A. Divakaran, "FoodX-251: A Dataset for Fine-grained Food Classification," *arXiv preprint arXiv:1907.06167*, 2019.