Classification of Mushrooms Using Machine Learning

UNIVERSITY OF
LINCOLN

Jacob Morgan
25234606 | MOR20789131

25234606@students.lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of BSc(Hons) Computer Science

Supervisor Dr. John Atanbori

Word Count - 11844

May 2023

Acknowledgements

I would like to thank my project supervisor Dr John Atanbori for their continuous help and support throughout the duration of the project. John helped direct the project in the right direction alongside aiding with theoretical understanding.

I would also like to thank my family for their support throughout this project, especially the constant support of my Ma.

I would also like to thank my friends for their support, specifically, Joseph Connor Smith, Stewart Charles Fisher II, Eddie Leonard Niels Muschamp, Alexander James Leather, Patrick Richard Jameson, William David Cole, Josh Cooper, and Franco Cimmino.

Finally, I would like to thank my partner Paige Morrison for her inspiration and support throughout the project.

Abstract

Objective To classify mushrooms using machine learning methods.

Design A paper detailing the implementation of three machine learning models capable of classifying mushrooms.

Setting An open-source dataset from the Kaggle website was used to train models made using TensorFlow

Participants Open-source images collected from the Kaggle dataset

Conclusion Machine learning techniques are effective for classifying mushrooms, especially convoluted neural networks, achieving accuracy scores of 80% and upwards.

Keywords Mushrooms, Convoluted Neural Network, Deep Learning, Machine Learning, Image Classification

Table of Contents

1 Introduction **1**

1.1 Background 1

1.2 The Intentions of the Project 3

1.3 The Significance of the Project 3

1.4 Aims & Objectives 4

1.4.1 Project Aims 4

1.4.2 Objectives 5

2 Literature Review **7**

2.1 Introduction 7

2.2 Regarding Datasets 7

2.3 Regarding Machine Learning 9

2.3.1 Conventional Machine Learning 9

2.3.2 Deep Learning 11

2.3.3 Image classification 12

2.3.4 Convolutional Neural Networks (CNNs) 12

2.4 Similar Previous Studies 13

2.5 Conclusion 15

3 Requirements Analysis **16**

3.1 Requirements 16

3.1.1 Dataset Aqusition Requirements 16

3.1.2 Machine Learning Requirements 17

3.1.3 Testing Requirements 17

4 Design & Methodology **18**

4.1 Project Management 18

4.1.1 Software Methodology 18

4.1.2 Scheduling 20

4.2 Project Supervision 21

4.3 Documentation 21

4.4 Toolsets . . . 22
4.4.1 Python . . . 23
4.4.2 Kaggle . . . 23
4.4.3 TensorFlow . . . 24
4.4.4 Google Colab . . . 25
4.4.5 Github . . . 25
4.4.6 Artifact Libraries . . . 26
Pandas . . . 26
Numpy . . . 26
Matplotlib . . . 26
4.5 Testing . . . 26
4.5.1 Performance Metrics . . . 26
4.5.2 Hyper Parameters . . . 27
4.6 Risk Analysis . . . 28

5 Implementation **29**
5.1 Preparing The Development Environment . . . 29
5.1.1 Hardware Accelerators . . . 29
5.1.2 Mounting and using Google Drive . . . 32
5.2 Aquisition & Processing Of A Dataset . . . 34
5.2.1 Dataset Aquisition . . . 34
5.2.2 Loading Dataset . . . 36
5.3 Models & Training . . . 42
5.3.1 Creating champignon-net . . . 44
5.3.2 Fine Tuning champignon-net . . . 48
5.3.3 Implementing Other Models . . . 51
EfficientNet . . . 51
ResNet . . . 54

6 Results & Discussion **56**
6.1 Dataset Results . . . 56
6.2 Champignon-net Results . . . 58
6.2.1 Standard Champignon-net . . . 58
6.2.2 Fine Tuned Champignon-net . . . 59
6.3 EfficientNet & ResNet Results . . . 62
6.3.1 EfficientNetB1 . . . 62
6.3.2 ResNet101 . . . 65
6.4 Comparison . . . 68

7 Conclusion **70**

References **71**

List of Figures

2.1 Working process of the proposed models from (Zahan et al., 2021) . . 8
2.2 Representation of the mathematical model of a neuron (Marsland, 2015) 10
2.3 Scheme of image classification systems (Lorente, Riera and Rana, 2021) 12
2.4 A simple CNN architecture, comprised of five layers (O'shea and Nash, 2015) 13
2.5 Comparison between models in (Zahan et al., 2021) 14

4.1 Project Kanban Board 20
4.2 Gantt chart made to track progress, from the interim report and project proposal (Morgan, 2023) 21
4.3 Example of the documentation process 22

5.1 Notebook settings menu accessible via the Runtime tab in Google Colab 30
5.2 The result of !nvidia-smi 30
5.3 GPUs detected by TensorFlow, Setting default GPU device to detected GPU 32
5.4 Google Colab environment with mounted Google Drive 33
5.5 Mushroom dataset stored in Google Drive 36
5.6 Visualisation of a random image in the Agaricus class, along side information about the dataset 38
5.7 Visualisation of a random image in all classes from the TensorFlow training dataset made earlier 42
5.8 Visualisation of Champignon-net 47

6.1 Amount of images in each class for both datasets 57
6.2 Champignon-net Results 58
6.3 Champignon-net-2 Results 60
6.4 Champignon-net-2 TensorBoard Accuracy Results | Red - accuracy , Blue - val_accuracy 61
6.5 Champignon-net-2 TensorBoard Loss Results | Red - loss, Blue - val_loss 61
6.6 EfficientNetB1 Accuracy Metrics 62

6.7 EfficientNetB1 Loss Metrics 63
6.8 EfficientNetB1 predictions visualised 64
6.9 ResNet101 Accuracy Metrics 65
6.10 ResNet101 Loss Metrics 66
6.11 ResNet101 predictions visualised 67

List of Tables

2.1 Comparison between results obtained from literature 15

6.1 Results of all models 68
6.2 Comparison between results obtained from literature and results obtained from this project 69

Chapter 1

Introduction

> It starts with Adam and Eve, and it will continue after the ultimate man has looked his last on a dying world.
>
> Rolfe R.T and Rolfe F.W, 1925

1.1 Background

The mushroom is interwoven into the history of man, from the first Homo erectus plucking wild fungi to Sapiens cultivating and ascribing mushrooms to the very messiness and rottenness of life. Mushrooms have always been loved or hated, by species, religions and movements. A cornerstone of debate in cuisine. In the West, people were polarised on the topic, either hating or loving mushrooms, as the East predominately looked through a mycophilic [1] lens. In recent years, science has allowed mushrooms to go from folk tales to trials, exploring the possible benefits mushrooms could give to us.

Regarding the current climate crisis, mushrooms can support us in the process of waste reduction. A study conducted by the University of Yale shows the breaking down of a variety of pollutants through the use of fungi (Russell et al., 2011). Many environmental movements have sprung out of the crisis, specifically movements re-

[1] Mycophilia, Enthusiasm for fungi, esp. edible ones; fondness for eating mushrooms (Dictionary, 2023)

garding the reduction of the consumption of meat. This opens the floodgates to alternatives, such as the rich, meaty texture of mushrooms. Mushrooms are firmly re-centred on the modern menu as a leading alternative to meat.

Mushroom identification is a difficult task. With the thousands of mushroom species and their complex combination of gills, spores and caps. This makes it quite hard for someone to identify the type of mushroom they're dealing with and without prior knowledge of mycology, this can be dangerous. Some mushroom species are not edible by humans, for example, Amanita phalloides (The Death Cap). A large fleshy agaric that occurs during summer to autumn and is part of the Amanita genus, which also happens to be lethal to consume (Jordan, 2004). Amanita phalloides look extremely similar to other mushrooms in the genome, in which a lot is edible. This has led to incidents in which a large number of people have mistaken Amanita phalloides for another mushroom, leading to their unfortunate demise. One of the most severe cases of this was in 1918 near Poznań in Poland, in which thirty-one schoolchildren died due to eating a dish containing Amanita phalloides prepared by the school (Benjamin, 1995).

The standard way humans identify mushrooms has always been through descriptions, pictures or memory, all being inefficient ways of classifying an organism. Due to the multi-variability of mushrooms and innate inefficiency in human brains. In recent years, however, a solution has emerged.

Machine learning is a branch of artificial intelligence which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy (IBM, 2023b). This means a computer learning how to reliably complete a task with a high success rate can do the job of a human within seconds with confidence the task is done correctly. Machine learning is used a lot in image classification, using algorithms to predict certain features of an image to predict what the image is. This can be used in fields like robotics to identify what an object in front of the robot is to then decide what to do next with the object.

One growing method in machine learning is Convoluted Neural Networks (CNN). CNNs are based on traditional Artificial Neural Networks (ANN). A network of

nodes, or neurons, that model neurons in a human brain and simulate learning using weighted values manipulated by non-linear functions. CNNs are similar to this in the way that the neurons self-optimise through learning. The notable difference between ANNs and CNNs is that CNNs are used primarily on pattern recognition in images (O'shea and Nash, 2015).

This method of image classification is used in many fields, specifically agricultural research and nature conservation. Here the problem of misclassification of mushroom species is solved. Allowing machines that can repeat highly accurate results to tell you what mushroom you are currently looking at could make the task of mushroom picking more efficient and safer for all.

1.2 The Intentions of the Project

This project intends to implement image classification for multiple mushroom species using machine learning methods, specifically using convoluted neural network models. A dataset of mushroom species will be pre-processed and used to train the models. An analysis will then be conducted to compare which of the models is the most accurate at classifying mushrooms.

1.3 The Significance of the Project

As stated previously mushrooms are cemented in human development, so it is no surprise there are vast commercial and domestic interests in mushrooms. From Mushroom cultivation for mass consumption to picking mushrooms in the forest, machine learning can provide a better quality of service and life.

In the industry of agriculture and cultivation, machine learning can provide many prospects that benefit efficiency and yield. Precision farming uses IoT devices to map crops and sample soil to precisely fertilise and spray pesticide on crops that need it (Auernhammer, 2001). This means less fertiliser and pesticide is used and saves on cost and environmental impacts. Using machine learning in conjunction with this means that in places where for example, weather patterns are unpredictable due to

rapid climate change, farmers can predict and automatically adjust to the conditions. Expanding on this, farmers can also predict soil properties and the nutrients of the land, meaning they can adjust what crop to grow based on the location, leading to greater yields (Sharma et al., 2021). This type of farming can be expanded to all sorts of crops, including mushrooms.

In domestic settings such as mushroom picking for personal use, services such as `iNaturalist` provide a safe environment for people to learn about nature using their smartphones (iNaturalist, 2023). The open-source app uses the camera on the phone to take a picture, iNaturalist then uses this image to predict what the image is of. iNaturalist can be used to track what organisms you have encountered and interact with other "Citizen Scientists" eager to learn more. iNaturalist also submits the image to the Global Biodiversity Information Facility, an international network and data infrastructure that provides open access data about all organisms (*GBIF* n.d.). This allows people searching for organisms such as mushrooms to identify and learn more information about the organism. This could be useful when mushroom hunting, identifying the organism as the one of interest could save a life.

1.4 Aims & Objectives

1.4.1 Project Aims

In machine learning, there are a plethora of models used for the purpose of image classification, some better than others. The aim of this project is to compare these models against different species of mushrooms, to determine which convoluted neural network is the best at classifying mushrooms. The aims of the project are the following

- Aquire a high-quality dataset(s)
- Train a minimum of three convoluted neural networks including one of my own self-made model
- Conduct an analysis of the results

- Carry out further research into convoluted neural networks and machine learning

1.4.2 Objectives

The objects outlined have been designed based on the project's aims. This is in order to formulate a methodical approach to the task.

1. Aquire high-quality dataset(s) that can be used to train and test the models.
   (a) The dataset(s) should be `high-quality`, meaning the dataset should be varied enough, contain enough data to generate a training dataset and have high-resolution images.
   (b) Dataset(s) should be gathered from reputable sources such as universities and well-known online sources such as Kaggle. This is so the project can be conducted with confidence and so the project can be compared to other studies.
   (c) A minimum of one dataset should cover the previous two points, to set the foundations for the project, so that the project can continue with confidence.
2. Implement at least three convoluted network models, including a self-made model, and train them on the dataset(s).
   (a) Use pre-made models that are known to work on other image classification studies.
   (b) Implement my own convoluted neural network
   (c) These networks should be using libraries that are public and open-source
   (d) Each network should aim for 80% accuracy or more
3. Compare the results of all models against one other.
   (a) The comparison should be made using scores such as validation accuracy and loss.

(b) Models must be compared using the same dataset. This should produce fair results.

4. Carry out further research into convoluted neural networks and machine learning

   (a) Perform a comprehensive literature review. This should include literature from books, academic journals or research papers.

Chapter 2

Literature Review

2.1 Introduction

Many studies have already been conducted regarding machine learning and its use in mushroom classification. This chapter aims to perform comprehensive research into, machine learning, the handling of datasets and its application in mushroom classification.

2.2 Regarding Datasets

In (Zahan et al., 2021) the use of preprocessing techniques are utilised to make the dataset more suitable for the models. First, they collect the dataset from various places, although not explicitly stated where from, they use images collected from mushroom centres as well as the internet. The paper mentions the fact some of these images are not suitable in size and colour, noting that some are "noisy". They try and improve the quality by applying the CLAHE method. CLAHE, Contrast Limited Adaptive Histogram Equalization is seen in (Reza, 2004). In this work they describe how the method can be successfully implemented, giving examples of medical images in which the method has "produced good results".
In (Zahan et al., 2021) they propose a working process, the first section in this process describes similar Previous Studies and how they prepare the dataset. You can see this in figure 2.1.

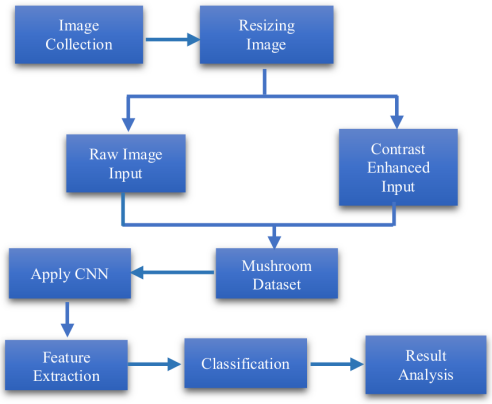


Figure 2.1: Working process of the proposed models from (Zahan et al., 2021)

In (Kiss and Zúni, 2021), this work mentions a new dataset, in which they have scraped from `2018 FGVCx Fungi Classification Challenge data-set` and The Mushroom Observer website to create The Mushroom Observer 106 Data-Set. They filter the images using the criteria:

- Species with at least 400 images
- A certainty more than or equal to 2

. After filtering these images they had over 106 classes downloaded. They then further clean the images using a small CNN, applying a ReLU activation function on each layer. They set the parameters of this model as :

- Input resolution 150 × 150
- Mini-batch size of 32
- Binary cross entropy loss function
- Adam optimiser with 0.01 learning rate
- 32 filters per convolution layer with 3 × 3 kernel size.

They finally had 29,100 images in 106 classes.
In (Schulzová et al., 2009) they use academic sources for their mushroom dataset from "the Center for Machine Learning and Intelligent Systems, University of California,

Irvine". This dataset isn't an image dataset and is purely based on the parameters of the mushrooms.

2.3 Regarding Machine Learning

Machine learning is the technique that improves system performance by learning from experience using computational methods (Zhou, 2021). This experience comes from data and learning is developed using algorithms that build models. Using this experience allows the models to make predictions on new data. The term "model" is a general term for the outcome learned from the data.

2.3.1 Conventional Machine Learning

The algorithms are designed to learn, but what is learning? (Mitchell, 1997) states "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". The task T in this definition is what the algorithm is learning to do, not the algorithm itself, for example, learning to play checkers. The performance measurement P in the context of checkers would be the percentage of games won and the Experience E would be games it has played against itself. This can be expressed through a linear function where coefficients or weights are chosen by the algorithm. These values will determine the importance of features in the context of completing a task (Mitchell, 1997).

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_n x_n$$

This can also be represented as

$$\sum_{i=1}^{m} w_i x_i$$

Where x is the features of the task, in the context of checkers this could be something as simple as the number of black pieces on the board. The weights are then updated

after each cycle using a function in which a constant is used called a learning rate (η).

Conventional methods of machine learning are based on the techniques previously stated. This takes the form of Artificial Neural Networks (ANN), which are models that are designed based on biological neurons in the brain. Marsland describes these neurons as such "Each neuron can be viewed as a separate processor, performing a very simple computation: deciding whether or not to fire.", this can be translated over computationally as a node deciding to produce a 1 or a 0(Marsland, 2015).

This node consists of three components, weighted inputs, an adder and an activation function. The weighted values described earlier are vital in decision-making in the neuron as the values enter an activation function once added. This decides the neurons state, producing either a 1 or a 0, based on a threshold value (Marsland, 2015).

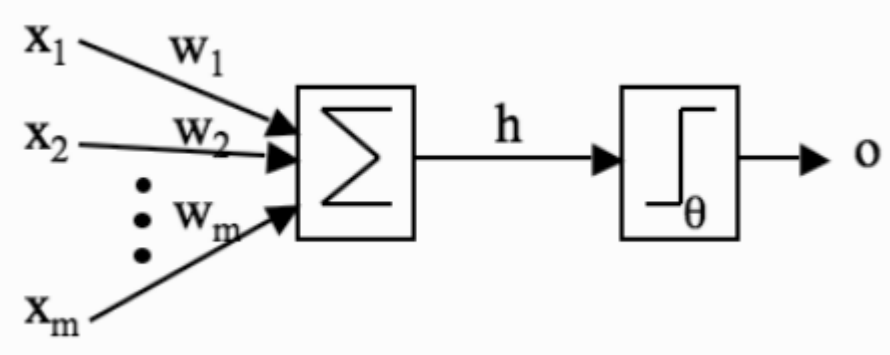


Figure 2.2: Representation of the mathematical model of a neuron (Marsland, 2015)

The neuron in figure 2.2 was invented by McCulloch and Pitts and is also known as a perceptron (McCulloch and Pitts, 1943).

These perceptrons can be used in conjunction with each other to form multilayered perceptions (MLP), a larger network, this is what would be called a "vanilla" artificial neural network (Hastie, Tibshirani and Friedman, 2009). During training MLPs feed information through the network in a process called forward propagation to produce a scalar cost $J(\theta)$ (Hastie, Tibshirani and Friedman, 2009). After forward propagation, backpropagation is used. This allows the information to flow backwards through the network in order to compute the gradient (Goodfellow, Bengio and Courville, 2016).

Applications of MLPs include Support-Vector networks, first described by Vladimir Vapnik in his paper by the same name (Cortes and Vapnik, 1995). The algorithm is given a training dataset, marked with one of two categories, it then builds a model that allows new examples to be assigned in one of the categories. This makes the Support Vector Machine (SVM) a binary linear classifier, a type of supervised learning.

In the context of classification, a paper from 2018 shows the use of SVMs when classifying edible and non-edible mushrooms (Wibowo et al., 2018). The implementation of an SVM performs significantly well, getting 100% accuracy at classifying whether the mushroom is edible or not.

The basics of artificial intelligence and machine learning are needed in order to understand the complex and treacherous territory of deep learning.

2.3.2 Deep Learning

Deep learning is a subset of machine learning and can be described as a neural network with three or more layers (IBM, 2023a). Deep learning differs from traditional machine learning, where traditional machine learning uses structured labelled data, Deep learning can make predictions and learn from unstructured data. This removes the dependency on humans as the features that would have previously been labelled by humans now are inferred by the algorithm.

In recent years deep learning has had a mainstream attraction with the GPT models. The latest of these is the GPT-4 model, this is an extremely deep model, that needs supercomputers to carry the workload (OpenAI, 2023b). This model is a Generative pre-trained transformer and takes its form in the viral chatbot Chat-GPT, this is a model which is trained to follow instructions and provide detailed responses (OpenAI, 2023a).

Although this doesn't seem relevant, these large models which have lots of funding will only lead to a deeper understanding in a somewhat unexplored field. Deep learning is taking off and its utilisations could lead to a very different world for us

all (Eloundou et al., 2023). Utilising these models could lead to mass automation as well as generative content on a scale never seen before.

2.3.3 Image classification

The previously mentioned utilisation of deep learning for inferred labelling of datasets is extremely useful when used in image classification.

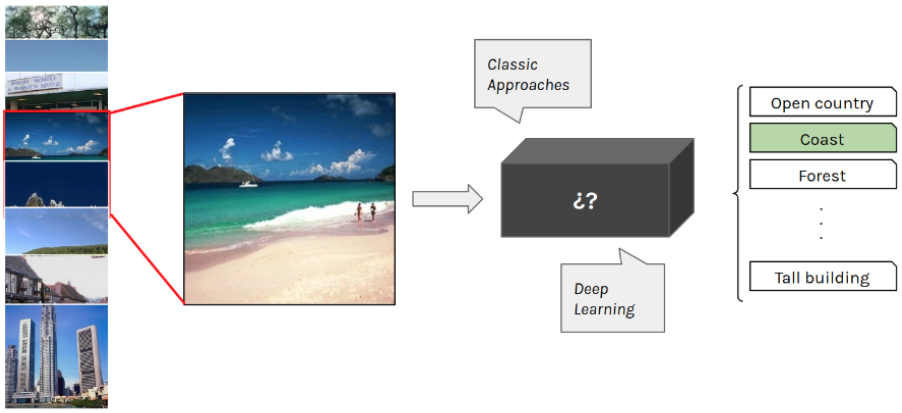


Figure 2.3: Scheme of image classification systems (Lorente, Riera and Rana, 2021)

The paper Image Classification with Classic and Deep Learning Techniques uses three techniques to compare traditional methods with deep learning methods in the context of Image Classification (Lorente, Riera and Rana, 2021). First, they use a technique known as Bag of Visual Words which retrieves key points of interest by treating image features as words (Sivic and Zisserman, 2003). They then compare it to a traditional MLP, to improve the performance of the system they they use a Convolutional Neural Network, a technique which is "more suitable for image classification".

2.3.4 Convolutional Neural Networks (CNNs)

CNNs are based upon traditional ANNs, they contain neurons that self-optimise through learning (O'shea and Nash, 2015). As this is a method of deep learning CNNs are primarily used in the field of pattern recognition within images. CNNs focus on the basis that the input will be comprised of images (O'shea and Nash, 2015).

CNNs comprise three types of layers, convolutional layers, pooling layers and fully connected layers.

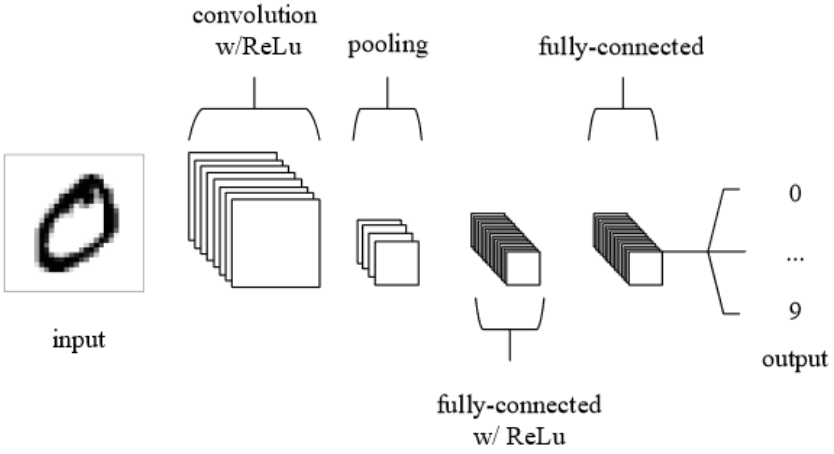


Figure 2.4: A simple CNN architecture, comprised of five layers (O'shea and Nash, 2015)

The input layer holds the image, specifically the pixel values of the image. The convoluted layer acts like a filter, looking for specific patterns or features. Each filter consists of a set of weights which are multiplied with a small section of the input, this is what is known as the scalar product. the neurons in a convoluted layer are connected to a small region, this helps the layer focus on specific features (O'shea and Nash, 2015). An activation function is then applied, specifically a rectified linear unit, which aims to apply a function such as sigmoid to the output of the activation produced in the previous layer (Nair and Hinton, n.d.). The pooling layer then downsamples the input further reducing the number of parameters. The fully connected layers will then perform similar duties found in a traditional ANN and attempts to produce scores from the activations to be used for classification (O'shea and Nash, 2015).

2.4 Similar Previous Studies

In (Anil, Gupta and Arora, 2019) the proposed methods are shallow machine-learning methods. Using a database of 60 images of fresh mushroom samples and 75 images

of non-fresh they attempt to classify the freshness of these mushrooms. The methods of classification that are used are :

- Support Vector Machine
- K-nearest neighbour
- Logistic Regression
- Classification Tree

A total of 20 features are extracted from the mushrooms and reduced to only 5 to reduce classifier complexity. They found that the classifier that performed the best was the Support Vector Machine with an accuracy of 80%.

In (Zahan et al., 2021) they compare 3 deep learning methods, VGG16, Resnet50 and InceptionV3. They split the dataset 80% for training and 20% for validation. All these models used transfer learning, a method that uses the knowledge gained from a model to solve a problem and then reuses this knowledge as the starting point. The learning rate for the models was set to 0.001 and they use the Adam optimiser and Softmax functions. All of this was done using the Python programming language in an anaconda environment. After getting the results from a confusion matrix the model that outperforms the rest is InceptionV3, seen in figure 2.5.

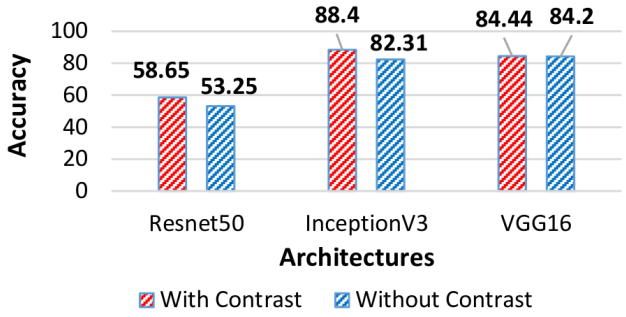


Figure 2.5: Comparison between models in (Zahan et al., 2021)

In (Szegedy et al., 2015) the researchers go further in-depth into the Inception model. Proposing the idea of the architecture of Inception explaining it simply as finding the optimal local construction and repeating it spacially. They also show the GoogLeNet

incarnation, showing the 22-layer deep CNN and describing the training methodology that follows.

In (Kiss and Zúni, 2021) they use the EfficientNet (TensorFlow, 2022) from tensorflow (TensorFlow, 2023b) to produce the results and use ImageNet for the transfer learning and noisy student weights. They concluded that the B5 variant with noisy student weights produced the best results when classifying mushrooms, with an accuracy of 92.6%.

2.5 Conclusion

As seen in the majority of the papers, a common practice is pre-processing the dataset. This increases accuracy, as seen in figure 2.5, and it seems like a vital part of the process when training models for the purpose of mushroom classification. When regarding machine learning it is evident throughout the papers that CNNs are a lot better at classifying mushrooms than shallower models.

Models	Accuracy	Study
InceptionV3 (Raw image)	0.82	(Zahan et al., 2021)
InceptionV3 (CLAHE)	0.88	(Zahan et al., 2021)
EfficientNetB5	0.92	(Kiss and Zúni, 2021)
Support Vector Machine	0.80	(Anil, Gupta and Arora, 2019)

Table 2.1: Comparison between results obtained from literature

As seen in 2.1 deep learning methods produce far better results than standard machine learning methods.

Chapter 3

Requirements Analysis

An overall understanding of what is required from the project is needed in order to proceed. In this section an outline of what is required in order for the project to be deemed a success will be laid out in full.

3.1 Requirements

3.1.1 Dataset Aqusition Requirements

First, a high-quality dataset is to be acquired, either from a trusted online community such as Kaggle or from an academic source such as a university. This is a high priority since the rest of the project can not be conducted without data to train the model on. The data must be high quality, as the higher quality of the image the better the features will be that is extracted from the input data. The quality of the data is defined by how high the resolution of the image is, how clear the image is and how visible the features of a mushroom are in the input image. As previously stated the dataset must be from a reputable source, this is important since the dataset could be used in applications where a user is identifying between edible and non-edible mushrooms.

Secondly, to make sure the dataset does not cause any errors due to corrupted files, a function needs to be written to scan the dataset and remove any files that may be corrupted. Having corrupted data could severely damage the overall results of the models so the data needs to be scanned to ensure the dataset is of high quality.

Expanding on the quality of the dataset, pre-processing techniques can be applied to

the dataset to make features of the input image stand out. This isn't a requirement but it might be helpful if implemented. On acquiring a dataset, manually downloading images could be tedious, an automated process could be beneficial but again isn't a requirement but would be helpful.

3.1.2 Machine Learning Requirements

First, testing between a minimum of three models is a must-have requirement. This ensures the data produced from all models can be evaluated to compare which is the best model for classifying mushrooms.

At least one of the models has to be produced by myself. This means I can tweak the model to fit with the dataset, meaning the model will be suited for mushroom classification. A comparison will then be made against the other models, which will be pre-made and will likely use Image-net for weights.

Models are expected to obtain an accuracy of over 85% this ensures that the model shows potential. If the model does not obtain this then another model will be chosen, or changes will be made.

Callbacks will be required for the models such as the EarlyStopping callback, which stops the training if the validation accuracy target is not being met. This helps with training over a long period of time as leaving a model on and not surveying the model could waste time.

3.1.3 Testing Requirements

It is a requirement that quantifiable metrics are produced to measure the performance of a model. Metrics such as accuracy, validation accuracy, recall, f1 score and precision. These metrics should be able to assist in the analysis of each model.

After training is completed, the models will be required to classify a brand new image to see which model reliably classifies the correct mushroom.

Chapter 4

Design & Methodology

4.1 Project Management

In order to meet the requirements of this project, good organisation and time management skills are vital. To make sure this happens project management methodologies are to be used to help with the workflow of the project. Specifically task allocation and scheduling methodologies.

4.1.1 Software Methodology

Software methodologies are the backbone of software engineering, whether it's a waterfall or the more versatile agile method, these are used to ensure that workflow and change are constant. Implementing several of these methodologies will allow me to keep on top of tasks as well as make constant progress to meet the requirements set. Agile is a methodology which started as an idea that was forged on a ski trip, the attendees (all being software enthusiasts) came up with the idea would be a system that would allow for an alternative approach the software development (Beedle, 2001). This is now applied widely to running all sorts of projects and products, agile works on twelve principles which allows the users to work more effectively together, these principles won't be laid out here but are available in the manifesto. If waterfall is serial, agile is parallel, allowing users to do separate processes at the same time instead of one at a time. You start small in alpha phases and grow the project into something larger and you only go live once feedback and data show your service working. This allows the users to continuously learn and improve to build something

from the ground up. This is the methodology this project will be following, as It will allow for researching, designing, building and testing all the way up until final implementation. Agile has many methods, the one this project will be using is the Kanban method. This is a method inspired by production systems that focus on reducing waste and improving quality, perfect for machine learning development as improving the quality of datasets and models is exactly what is required. Kanban is a way of visualising and improving the current working practices so that work can flow (Gov, 2023). Kanban utilises a kanban board, an agile project management tool designed to help visualise the work being done, it also helps limit work-in-progress designs and maximizes efficiency (Atlassian, 2023). These boards utilise the use of cards and columns to visualise tasks. The cards are used as visual signals to offload work items onto them, this helps visualise the progress and understand what is currently being worked on. The columns compose workflow and usually are structured as "To-Do", "In Progress" and "Complete". This project utilises this method to keep track of the progress of the project, assigning small tasks in the To-Do section and moving them up as progress is made through each task. This shows a sense of progress which not only is a great way to visualise the progress but also keeps motivation high.

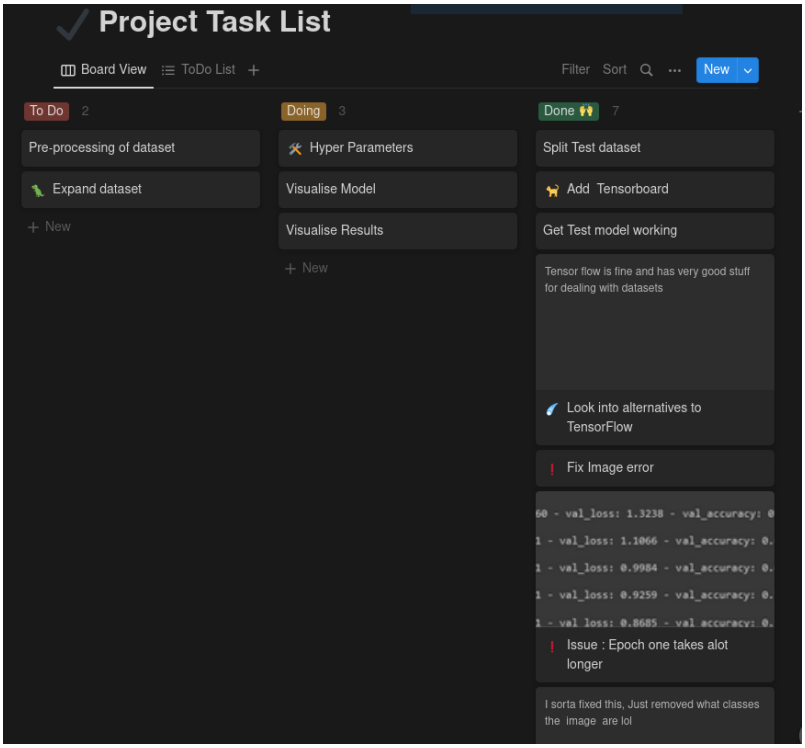


Figure 4.1: Project Kanban Board

Here Notion is used, a freemium [1] tool for productivity and note taking application. It offers tools such as task management, project tracking, lists and bookmarks and can be used by companies, teams or individuals to manage a product as well as document it (Labs, 2023). Notion is used throughout the project to document progress and make to-do lists and notes.

4.1.2 Scheduling

Gantt charts are commonly used in project management as a useful tool to show activities (Tasks or events) against time. The left of the chart is a list of activities and along the top is the time scale. Each activity is represented by a bar and the position and length of the bar reflect the start date, duration and end of the activity. This is a great way to show a sort of "roadmap" for the future of your project, it also helps keep track of what needs to be done and the priority of each task(Gannt, 2023).

[1]Freemium, a pricing strategy by which a basic product or service is provided free of charge, but money (a premium) is charged for additional features (Wikipedia, 2023)

The Gantt chart shown in figure 4.2 was made at the start of production as a guideline for tasks to follow. This gives an overall duration for how long should be spent on certain tasks.

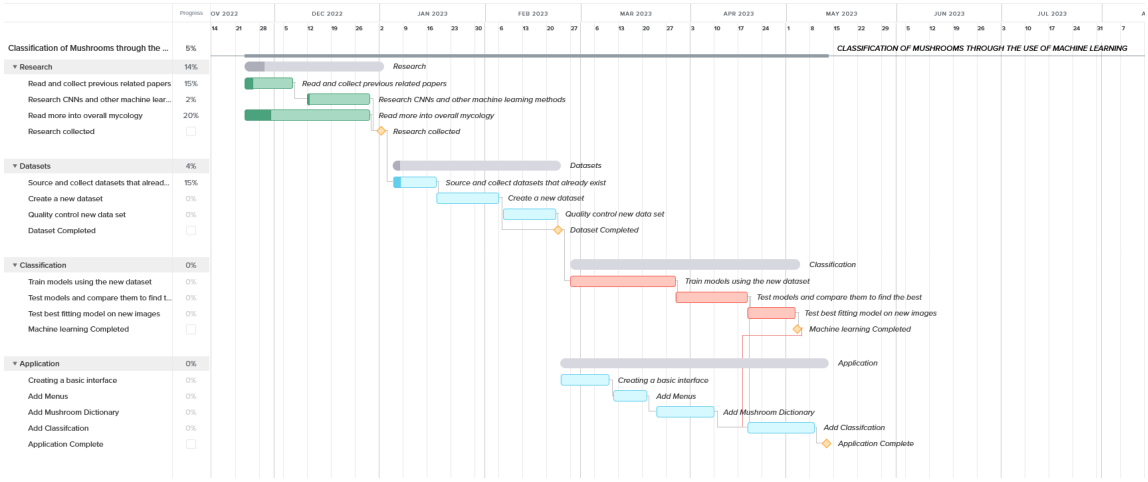


Figure 4.2: Gantt chart made to track progress, from the interim report and project proposal (Morgan, 2023)

4.2 Project Supervision

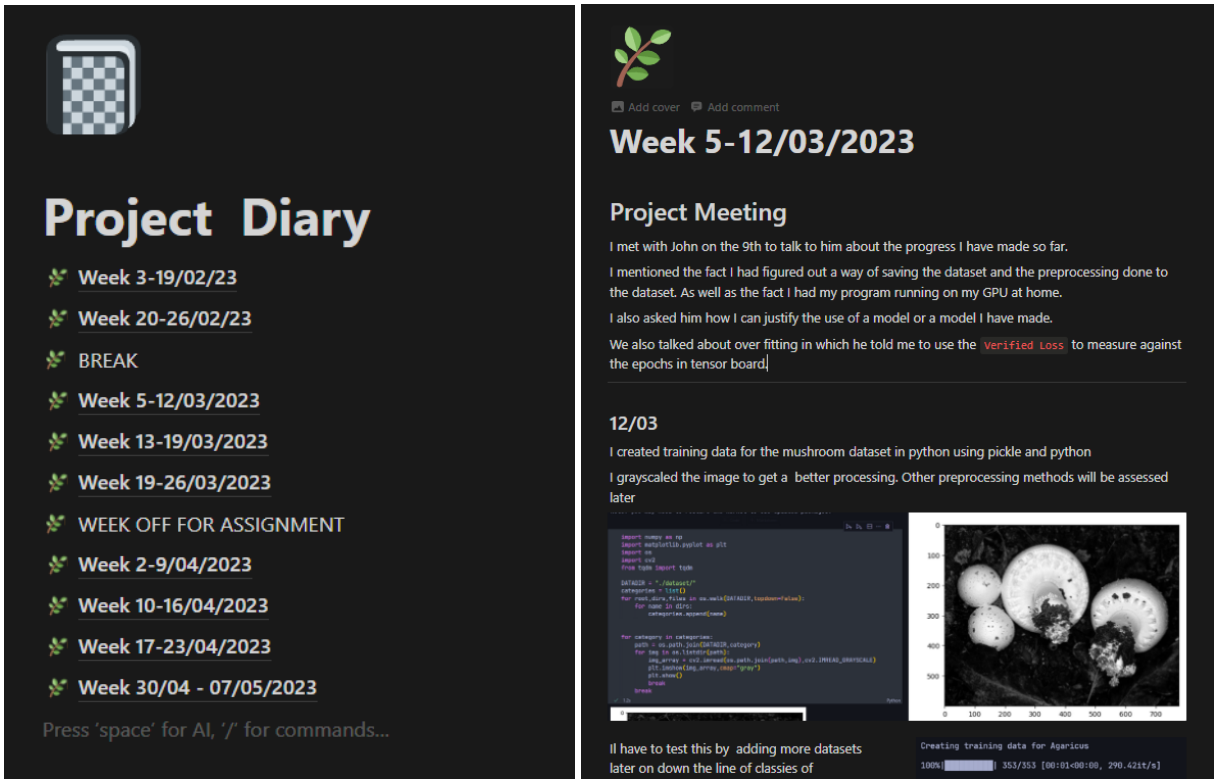
Weekly meetings between the project supervisor and me were set up over the duration of the project. This was to ensure communication and track progress made on the project. It also allowed me to ask for advice and ask technical questions about the project as well as to identify issues early on and talk about actions that could be taken. These meetings were detailed in Notion in the project diary, more of which will be talked about in section 4.3. Overall the meetings were extremely helpful for understanding and evaluating what needed to be done but most of all the overall support throughout the project was well appreciated.

4.3 Documentation

A well-documented process makes it much easier to know what has been done and what needs to be done. It also makes it a lot easier to talk about later on, after you have left the same mindset you were in beforehand. Crucial details about how something was implemented or issues you were having could be lost in time, this makes

documentation all the more important. In industry many companies use Git-hub pages to log their open source projects, using markup languages like markdown(md). Notion also has this functionality, allowing for documentation to be written. Usually, the term documentation refers to how to use a library or a piece of software but in this sense, this refers to the documentation of the development process. These are sometimes referred to as "dev-logs" and are very common on developers' websites, especially for big projects or games.

The documentation for this project was done in Notion, as Notion is accessible from anywhere due to its mobile and web applications, this allowed for notes to be made in project meetings as well as note down ideas on the go.



(a) Project Diary directory (b) Contents of an entry

Figure 4.3: Example of the documentation process

4.4 Toolsets

In order to implement the requirements tools must be identified which will be used to develop the classification software. These tools span both categories of the pro-

ject's requirements, covering machine learning, dataset acquisition and dataset pre-processing.

4.4.1 Python

Python is a very popular high-level general-purpose programming language(Rossum, 2023). The language has sat on top of the TIOBE index for years, solidifying itself as one of the best programming languages to use at the moment. The success of Python could be due to its previously mentioned simple syntax as well as its underlying complexity which allows for rapid prototyping. This kind of prototyping is much harder in lower-level and hard-typed languages such as C or C++.

Python also allows for the use of a vast amount of libraries, including a large number of scientific libraries. This makes it very appealing for machine learning as Python contains libraries such as Numpy, TensorFlow, PyTorch and more. These libraries perform most of the work allowing developers to not worry about the complexity of machine learning, allowing more time for innovative ideas.

4.4.2 Kaggle

Kaggle is an online community of data scientists and machine learning enthusiasts (Kaggle, 2023). It allows users to find and publish datasets as well as machine learning models. Kaggle also runs competitions each year that allows users to collaborate and learn machine learning together. Kaggle is mostly community-run, which can pose an issue regarding the quality of the dataset as anyone can submit a dataset there can sometimes be no quality control. Kaggle tackles this however by having an upvoting system, this means when someone searches for a dataset the most upvoted dataset will be at the top of the search results. This means that good datasets will get recognition by the community, meaning even the quality control is community driven. Kaggle as of 2023 has over 12 million users making it one of the biggest data science communities on the internet. This website is where the dataset for the project will be acquired, as the datasets are open source and easily accessible. It is

also easy to check the quality of the dataset as Kaggle has functionality which allows the user to preview the data.

Kaggle also allows for notebooks to be run on its servers, so if needed models can be run on Kaggle servers. Although this is more of a secondary functionality of Kaggle, primarily Kaggle will be used for dataset acquisition.

4.4.3 TensorFlow

TensorFlow is an open-source library for machine learning and can be used to train and infer deep neural networks (TensorFlow, 2023b). It was developed by Google for internal use but was released open-source in 2015. It can be used in a variety of programming languages, including Python.

In development, TensorFlow offers the developer, multiple models that you can start with right away and through the use of the Keras API, this makes TensorFlow highly accessible.

Keras is a machine learning API written in Python which designs the syntax "for humans, not machines" (Keras, 2023). It focuses on conciseness, maintainability and deployability making the code smaller and more readable. It is built on top of TensorFlow and can scale to large clusters of GPUs or TPUs (Tensor processing units). Keras in conjunction with TensorFlow is used by CERN and many other scientific organisations, even the Large Hadron Collider uses it.

Tensorflow and Keras' extensive documentation, makes it easy to search and research how to implement tasks, which means you spend less time searching and more time coding. It also allows the user to understand how machine learning works by abstracting the complexity.

Tensorflow offers a range of APIs in its library, however, one of the most useful APIs is the Dataset API (TensorFlow, 2023c). Used to write efficient and descriptive input pipelines, this API is great for easily importing any Dataset and utilising it in code. This API also offers data preprocessing and augmentation on the newly made Dataset type.

Tensorboard is a visualisation toolkit for TensorFlow which allows for the visualisation and tracking of metrics such as loss and accuracy (TensorBoard, 2023). The toolkit also visualises models, weights, datasets and much more.

Overall TensorFlow is one of the best machine-learning libraries at the moment. TensorFlow will be utilised to import, preprocess and augment the dataset as well as build, tune and train models.

4.4.4 Google Colab

Google Colab allows for the execution of Python code through a browser and is especially suited for machine learning (Google, 2023a). Colab uses Jupyter Notebook, an interactive development environment for notebooks, code and data (Jupyter, 2015). It allows users to run individual blocks of code that are labelled by markdown blocks, this helps with workflow and can be very useful in machine learning. Expanding on the application in machine learning, specific blocks might contain different models and training functions, since training takes a long time it's beneficial to run one block at a time rather than all of the code. Colab offers the users extra compute resources, allowing for the use of Google servers to run on large blocks of GPUs and TPUs.

A disadvantage to collab is the timing out of the collab session. Colab will time the user out after a while, requiring the user to run everything all over again. It also times the user out of resources meaning that time spent on the GPUs and TPUs can be limited. This can bottleneck development time and issues found in code might take a lot longer to fix, as well as being tedious to restart.

Google Colab will be used as the main IDE for the project, defining and running models and utilising the features of Jupyter Notebook.

4.4.5 Github

GitHub is a hosting service for version control using Git. Originally developed in 2008 and later acquired by Microsoft, Github allows for the online hosting of Git repositories. This allows for the tracking of changes in source code over time. This

will be useful in the development of the project as If a mistake is made I can always have the code in a separate branch, saving the working code in a main branch.

4.4.6 Artifact Libraries

Pandas

Pandas is a tool for doing practical real-world data analysis in Python. It utilises a fast and efficient DataFrame object for data manipulation and also allows for the reading and writing of data between different data structures and formats. It also allows for data visualisation, having the functionality to plot graphs.

Pandas will be used for this precise reason, plot the results of the models to visualise metrics.

Numpy

Numpy is an open-source library in Python that allows for numerical computing (Numpy Team, 2005). It is used in machine learning as a way to format arrays of data, specifically in this project's case features of images. These powerful N-dimensional arrays will be useful to utilise throughout the project.

Matplotlib

Matplotlib is a comprehensive library for creating static, animated and interactive visualisations in Python (Matplotlib development team, 2012). This will be used to visualise the training and validation images.

4.5 Testing

4.5.1 Performance Metrics

All models that are to be tested will have metrics to analyse after training. After training the evaluate function will be used on the model, this outputs metrics such as Loss and Accuracy. However, more metrics can be obtained using a confusion

matrix, inside the matrix the values of true positives, true negatives, false positives and false negatives are found. Using the confusion matrix, accuracy, sensitivity, recall, precision and an F1 score can be calculated.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1Score = 2 \times \frac{precision \times recall}{precision + recall}$$

These calculations can be done using the Python package Sklearn, a machine-learning model used mostly for data analysis. Sklearn allows for multi-class analysis on a micro and macro level, meaning all metrics can be calculated for individual classes.

Tensorboard will also be used in testing the metrics of the models, Tensorboard allows for the visualisation of the metrics as well as detailed graphs and logging capabilities.

The two metrics that will say if a model is overfitting will be the validation accuracy compared to the training accuracy. This is because overfitting is when a model fits too closely to training data and consequently has poor performance on newly seen data meaning if the validation accuracy is low and the training accuracy is high, overfitting is taking place. Validation accuracy is a significant metric since its the indicator of how well unseen data will perform against the model. For example, if someone was to input a newly taken picture of a mushroom into a model with around 80% accuracy, the model will predict the mushroom correctly approximately 80% of the time.

4.5.2 Hyper Parameters

Some fine-tuning of the models can be done manually but for the best performance, automatic fine-tuning will produce the best results. Keras has a function that allows for this and randomly picks parameters to change in the model, comparing different

configurations of them against each other to figure out the best parameters for the model. This could help increase the accuracy of the model even further than before.

4.6 Risk Analysis

Risk	Likelihood	Assessed Impact	Mitigation Strategy
Over or Under fitting	Likely	Trying to find the reasons why a machine learning model may be over or under fitting is a long and tedious task. This could set the project back days or weeks. Not only that but if a model is used in real-world applications it could miss classify a mushroom.	Adding a loss regulator to the model could help prevent this. Also tuning hyper parameters such as learning rate, batch size which can significantly affect performance.
Corrupt data files	Likely	Obtaining a dataset from an open source website could lead to inconsistences in the dataset.	Remove the corrupt data files.
Dataset quality is not up to standards	Likely	If the contents of the dataset is not high quality it could affect the performance of the model overall.	Find a higher quality dataset or augment the dataset to fit the parameters of the model.

Chapter 5

Implementation

5.1 Preparing The Development Environment

5.1.1 Hardware Accelerators

Using TensorFlow requires the set up of a development environment in a specific way. TensorFlow, in order to run efficiently, needs access to hardware accelerators such as GPUs (Graphical Processing Units) or TPUs (Tensor Processing Units). TensorFlow has multiple guides on how to do this with different types of hardware accelerators and is easy to set up due to Tensorflows API (TensorFlow, 2023g). The hardware that will be used will be a GPU since it will gain a higher throughput than running it on a standard CPU training a machine learning model. GPUs are the most popular architecture to use in machine learning due to their general-purpose processor that can support millions of different applications (Google, 2023b).

Google Colab allows for a choice in hardware accelerator, allowing the user to choose between TPUs and GPUs, even allowing the user to choose between GPU types. Google Colab also has additional compute units that can be paid for, additional compute units will not be used due to a lack of funding. The menu that allows the user to choose between hardware accelerators can be seen in figure 5.1.

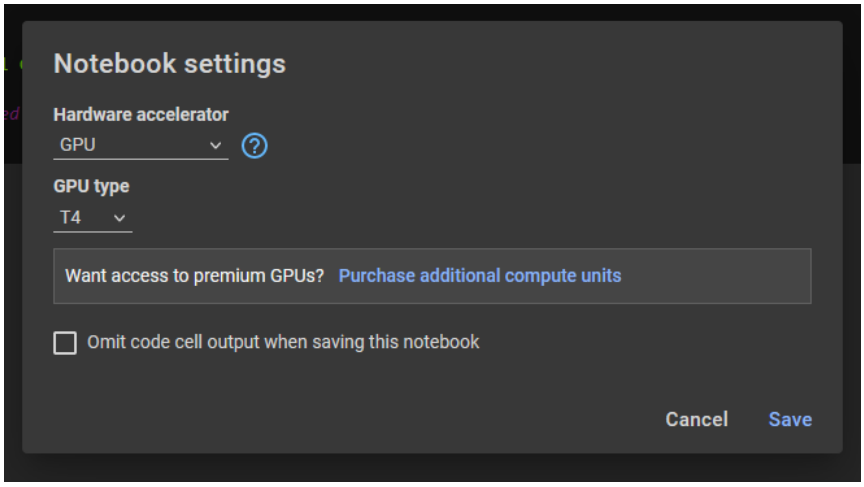


Figure 5.1: Notebook settings menu accessible via the Runtime tab in Google Colab

This allows for Google's dedicated servers to be accessed to run the machine learning models.To check if the GPU is working in the environment `!nvidia-smi` is called, this command outputs the GPU device state as well as the version of CUDA the GPU is running (NVIDIA, 2012). CUDA is a toolkit created by NVIDIA which is used for creating high-performance GPU-accelerated applications (NVIDIA, 2013). This means applications can be easily deployed on GPU-accelerated systems. The result of the command can be seen in figure 5.2

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   39C    P8     9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure 5.2: The result of !nvidia-smi

Now that the environment has a hardware accelerator, the next task is to import TensorFlow and configure the library in order to utilise the hardware accelerator. The code for this can be seen in listing 1.

```
import tensorflow as tf
from tensorflow.python.client import device_lib

print(device_lib.list_local_devices())
if tf.test.gpu_device_name():
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
else:
    print("Please install GPU version of TF")

gpus = tf.config.list_physical_devices('GPU')
if gpus:
  # Restrict TensorFlow to only use the first GPU
  try:
    tf.config.set_visible_devices(gpus[0], 'GPU')
    logical_gpus = tf.config.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPU")
  except RuntimeError as e:
    # Visible devices must be set before GPUs have been initialized
    print(e)
```

Listing 1: Importing TensorFlow and setting the device to Colab GPU

The code above lists all devices present in the environment, similar to the `!nvidia-smi` command. An attempt to set the device to the first device in the list is then made, because there is only one device that device is chosen, if no devices are found an error is thrown. The output can be seen in figure 5.3.

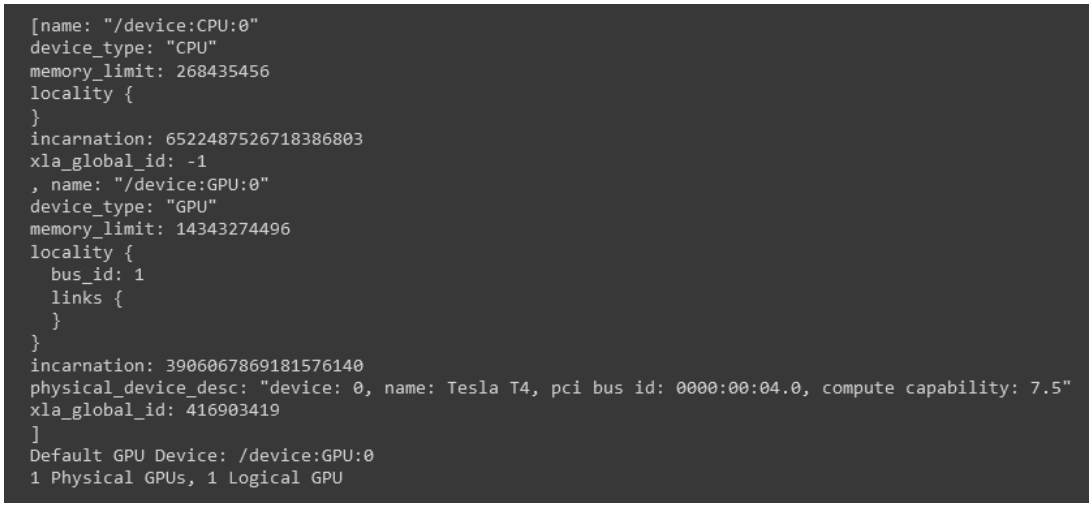


Figure 5.3: GPUs detected by TensorFlow, Setting default GPU device to detected GPU

Now that TensorFlow has been configured to use the Google Colab GPU, it is ready to be used in the further implementation.

5.1.2 Mounting and using Google Drive

After each session in Google Colab, the local directory gets wiped, this is for performance reasons as if thousands of people are using Colab and are storing items in local directories it could slow the entire service down. This is an issue, as if a dataset was downloaded into a local directory, each time Colab is loaded or restart Colab session all the files would disappear, this would not be ideal. Luckily Google has a service called Google Drive. Google Drive is a file storage system developed by Google and has over 1 Billion users worldwide. Google Drive stores files on Google servers and offers a free 15GB worth of storage (Google, 2012). This will be used to store the dataset as this is a permanent storage solution, meaning when the Colab session is restarted the dataset can still be retrieved. The only limitation here is the 15GB storage limit that comes with Google Drive, luckily I have Google One (Google, 2018) which means the Google Drive has 100GB, which is a lot better for storing large datasets and results.

Google Colab has the feature to "mount" a Drive, which means in the environment

files can be accessed directly from Google Drive using Python. This can be done using the built-in Colab library made by Google, as seen in Listing 2.

```
from google.colab import drive
drive.mount('/content/drive',force_remount = True)
```

Listing 2: Mounting Google Drive to Google Colab

The drive is now mounted to the environment and is ready to be used to store and retrieve the dataset.

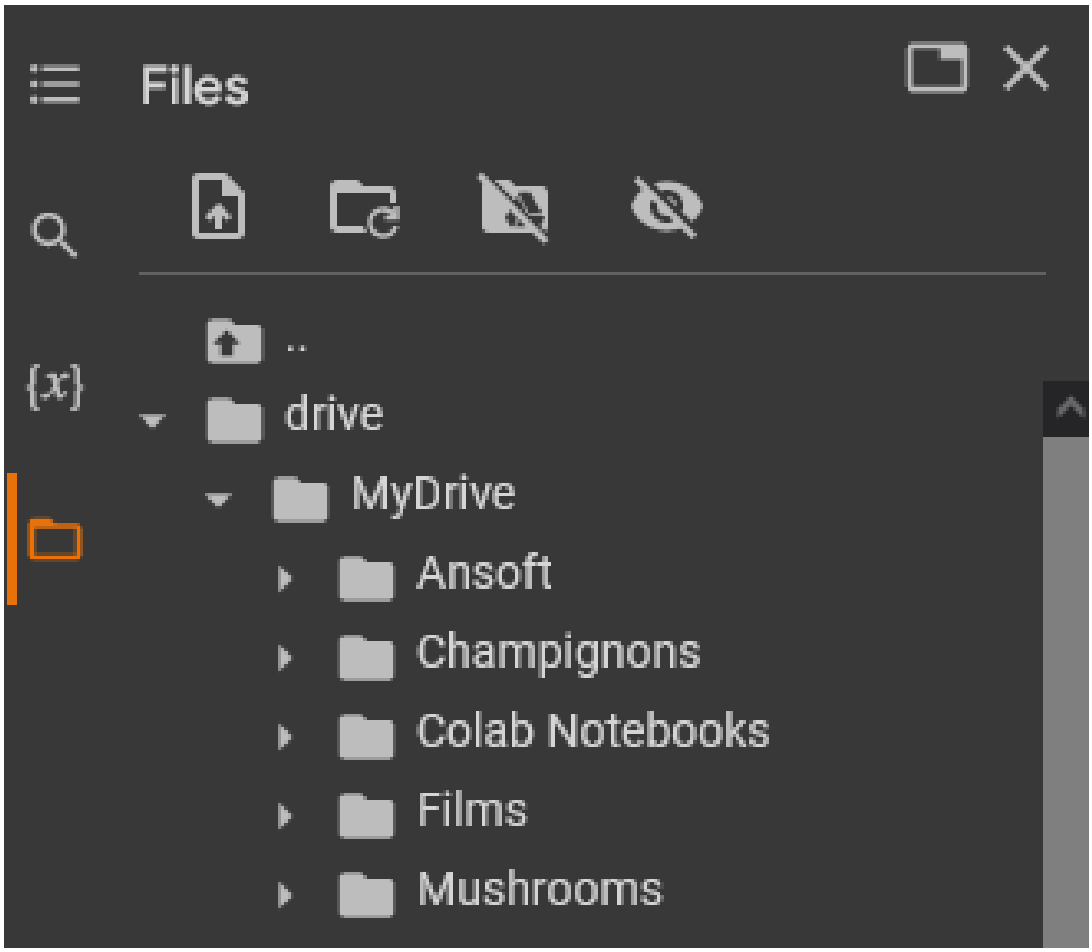


Figure 5.4: Google Colab environment with mounted Google Drive

5.2 Aquisition & Processing Of A Dataset

5.2.1 Dataset Aquisition

Without a dataset any implementation of machine learning can not be done, so first before creating any models a dataset must be acquired and processed. A dataset was searched for with the following qualities, high-quality images, clearly labelled images and a dataset without too many classes. A dataset from the website Kaggle was used, specifically Mushroom classification - Common Genus's images by the user CATODOGO (CatoDogo, 2019). This dataset is made up of 9 species of mushrooms, all being some of the most common in Northern Europe. Each mushroom is classified by the name of the folder it is in, with each folder consisting of 1500 to 3000 images. To acquire this dataset multiple methods were tried, using the Kaggle API and manually downloading the dataset.

The first method, the Kaggle API, was an attempt to try and automatically download the dataset, however, there wasn't any need for this. The Kaggle API was used to download the files into Google Drive.

```
import kaggle
import shutil
import os
os.environ["KAGGLE_CONFIG_DIR"] =
"/content/drive/MyDrive/Champignons/.kaggle/"
kaggle.api.authenticate()

if os.path.exists("./Mushrooms") != True:
  print("Downloading dataset ...")
  kaggle.api.dataset_download_files(
  "maysee/mushrooms-classification-common-genuss-images ",
  unzip = True)
  shutil.rmtree("./mushrooms")
else:
  print("Dataset already in working directory")
```

Listing 3: Automatic download of the Kaggle Dataset

This did in fact work and download the dataset to Google Drive, however, the dataset was never needed to download again, therefore making it useless. The second option was opted for over the first one, downloading the dataset manually, unzipping the files and placing them in Google Drive.

Both of these methods however had a flaw, when downloaded the dataset would have all the classes of mushrooms as separate directories but then have another folder just labelled "Mushrooms" with a duplicate of the dataset inside. Im not sure if this was intentional or not but to mitigate this issue the duplicated dataset was deleted. This is quite a large issue if not spotted early as having a duplicate piece of data can lead to overfitting, as the model becomes too specialized to the training data and performs poorly on new data. Just deleting the duplicated folder could however not be enough, what if there is duplicated data inside the directory containing the

images? Manually going through each image could take a while so a script would need to be written to check for duplicate images. however, this was not implemented as it was deemed not necessary. Now with the dataset downloaded on Google Drive and Google Drive mounted work can now be done on the dataset.

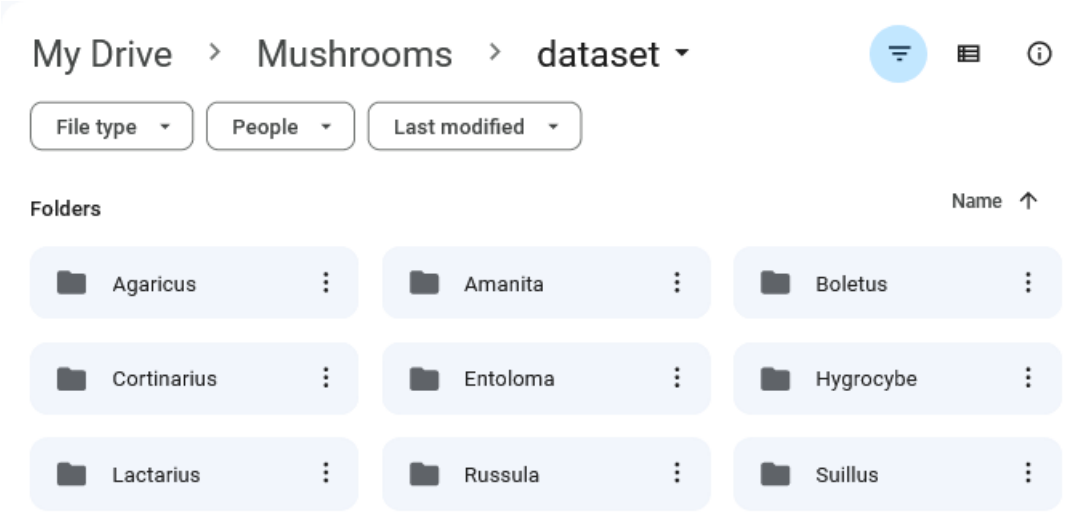


Figure 5.5: Mushroom dataset stored in Google Drive

5.2.2 Loading Dataset

To load the dataset first the directory of the dataset must be set. Here pathlib is used in order to define a Path class which is then used later to pass into functions which require the directory of the dataset. Pathlib is useful since the class it creates is immutable and hashable making them secure and great at making sure the directory path is never accidentally changed. Pathlib can also break down a directory into various components this is great when working with datasets whose classes are stored as directory names as Pathlib can easily parse this information from the directory (Python Software Foundation, 2023).

```
import pathlib
import random

#Storing the path of the dataset as a string
DATADIR = "/content/drive/MyDrive/Mushrooms/dataset"

#Converting this string into a Path class
data_dir = pathlib.Path(DATADIR).with_suffix('')

#Gets how many images there are in the dataset
image_count = len(list(data_dir.glob('*/*.jpg')))
```

Listing 4: Getting the directory of the dataset

The data is then visualised, a long side showing how many Images are in the dataset and the number of classes. This is to ensure no data is lost and to ensure the quality of the images in the dataset. A random image from the first class (Agaricus class) is shown. Later on once the images are read into a dataset type more data visualisation will take place.

Figure 5.6: Visualisation of a random image in the Agaricus class, along side information about the dataset

Early in development, an error occurred when a model was trained. At around the 66th step in an epoch, the model would crash. This was due to an "Invalid JPEG file" or a corrupted image file, the model was trying to perform calculations on the image but could not due to its corrupted state. This posed quite a challenge as it was impractical to go through the images manually and check which ones were corrupted. A small script was found on an issue page on Github relating to Tensorflow, one user was having a very similar issue leading another user (milad-4274) to reply with the script used (Milad, 2020). The script effectively cleans up the dataset, scanning each image and checking for certain flags and raising an issue when the data can not be read. The image path is then stored in a list which is then iterated through and removed using the Os library remove function. The script found around three "bad images" and removed them from the dataset.

Now the dataset is clear of all corrupted files, the dataset can be formally cre-

ated using TensorFlow. Utilising the Keras utility package specifically the image_dataset_from_directory function the TensorFlow dataset was created, which allows for the manipulation of the dataset alongside allowing the dataset to be used in machine learning models (TensorFlow, 2023d).

```
batch_size = 32
img_height = 200
img_width = 200

training_ds = tf.keras.utils.image_dataset_from_directory
(
    data_dir,
    labels = "inferred",
    label_mode = "categorical",
    color_mode = "rgb",
    validation_split = 0.33,
    subset = "training",
    shuffle = True,
    seed = 555,
    image_size = (img_height, img_width),
    batch_size = batch_size,
    crop_to_aspect_ratio = True,
    class_names = categories
)
```

Listing 5: Creation of Tensorflow Image Dataset, for training dataset

The dataset made in the listing above is for the training data. Here image_dataset_from_directory is used to read the images directly from the dataset directory. Tensorflow allows for the labels to be inferred, meaning the dataset will work out automatically what labels each image needs, it also sets the label mode to categorical meaning that the labels

as a categorical vector, this can be used in the categorical_crossentropy function. The colour mode parameter simply converts the images into having three channels for RGB. The validation split is the split that will be used for the validation dataset which will be defined after this one. The reason for the 67/33 split is due to overfitting, the split was changed after overfitting was taking place, meaning there was too much training data. The split could be further improved, most models use the Pareto principle, a rule created by Joseph M Juran to describe a connection in economics where 80% of the effects come from 20% of the causes (Data Detective, 2020). However, there is a better way, using the scaling law the split can be set specifically for certain datasets. The split is determined by how many unique features there are in the dataset (Kaplan et al., 2020). As described in the article "The 80/20 Split Intuitaion and an Alternative Split Method", "They find that the fraction of patterns reserved for the validation set should be inversely proportional to the square root of the number of free adjustable parameters" (Data Detective, 2020).

The training dataset is then labelled as the training dataset and is shuffled using the seed "555", this ensures the data is the same for testing, in a real environment there wouldn't be a seed.

The images are resized to 200x200, this is to make sure all images are the same size to reduce the resources needed when pooling from the images. The batch size is how many images are used before weights are changed, I set this 32 as this is the default in TensorFlow. The crop_to_aspect_ratio option makes sure there is no aspect ratio distortion when resizing the images, reducing potential corrupted or distorted images in the dataset. Finally the class names are set to a list of all categories in the dataset.

```
validation_ds = tf.keras.utils.image_dataset_from_directory
(
data_dir,
labels = "inferred",
label_mode = "categorical",
color_mode = "rgb",
validation_split = 0.33,
subset = "validation",
shuffle = True,
seed = 555,
image_size = (img_height, img_width),
batch_size = batch_size,
crop_to_aspect_ratio = True,
class_names = categories

)
```

Listing 6: Creation of Tensorflow Image Dataset, for validation dataset

The validation dataset is implemented the same as the training dataset and is split the same, making the validation dataset 33% of the original dataset.

Now that both the testing and validation split is done, a visualisation of the augmentations made to the dataset is needed.

Figure 5.7: Visualisation of a random image in all classes from the TensorFlow training dataset made earlier

The visualisation is done by using the matplotlib library and iterating over the images in the training dataset and adding them to a subplot. This visualisation shows that all of the images are the same size and all high quality.

5.3 Models & Training

Before creating any models callbacks must first be created. Callbacks traditionally are executables that are passed as an argument into a function and are expected to be run during the runtime of the function. In machine learning callbacks are

extremely useful for example some callbacks which will be used in this project are, an early stopping callback and a Reduced learning rate on plateau callback. These callbacks will be used at certain points during the model's training. Both of these callbacks were imported and set up in the environment.

```
early_stop = tf.keras.callbacks.EarlyStopping(monitor = "val_accuracy",
                                              patience = 5,
                                              restore_best_weights = True)

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor = "val_loss",
                                                 factor = 0.2,
                                                 mode = "auto",
                                                 patience = 5,
                                                 min_lr = 0.001,
                                                 cooldown = 0)
```

Listing 7: Creation of Callbacks for later use in models

The early stopping callback is part of TensorFlow's Keras API and aims to stop training the model once the monitored metric has stopped improving (TensorFlow, 2023e). This can help stop overfitting which can occur when the model gets too complex, by stopping the training at an optimal point it can improve the general performance of a model. Here the call back is monitoring the validation accuracy of the model, and stopping once the validation accuracy has plateaued.

The reduced learning rate call back, also part of TensorFlow's Keras API, does the same but with a learning rate. This callback monitors a metric and once it has stopped improving lowers the learning rate (TensorFlow, 2023f). Whenever the validation loss stops improving the learning rate is then reduced, this is due to the fact the model may have converged to a suboptimal solution and the step size needs to change.

These callbacks will reduce the model from overfitting alongside making sure the

model does not stop improving. This will hopefully increase the accuracy of the model.

5.3.1 Creating champignon-net

Champignon-net[1] is the self-made model that was described in the aims and objectives. Utilising the Tensorflow Keras layers functions champignon-net was constructed. It is a convolutional neural network with multiple layers which are then followed by fully connected layers. The CNN consists of multiple convoluted layers which are each followed by a layer of batch normalisation, activation and max pooling layers. In the model, these layers are repeated five times and reduce the image features and are then flattened into a one-dimensional vector towards the end of the model, the fully connected dense layers can then classify the output. Flattening also reduces the number of parameters in the model, this can prevent overfitting and reduce training time. The fully connected dense layers aim to learn non-linear combinations of the features learned by the previous convolutional layers, they are used at the end of image classification models to perform the classification task.

[1] Champignon, the french word for mushroom (Camridge Dictionary, 2023)

```
Model: "champignon-net"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 198, 198, 256)     7168

 batch_normalization_5 (Batc  (None, 198, 198, 256)    1024
 hNormalization)

 activation_5 (Activation)   (None, 198, 198, 256)     0

 max_pooling2d_5 (MaxPooling  (None, 99, 99, 256)      0
 2D)
```

Listing 8: One layer of convolution in champignon-net

The code in the listing above is the output of the summary function in Tensorflow being applied to champignon-net. It shows the previously mentioned convolution layer has been added to the model, with an input layer of 200x200, which is the size of the inputted image, as well as the filter size of 256. After this is the batch normalisation layer, the activation layer and the max pooling layer. The batch normalisation layer is a supervised learning technique that converts the interlayer outputs of the convolutional layer into a standard format, this resets the distribution of the output of the previous layer to be processed better by the next layer (DeepAI, 2019). The activation layer used here is the ReLU activation function or rectified linear unit and introduces the property of non-linearity or deep learning models (Nair and Hinton, n.d.). The max pooling layer accumulates features generated by convoluting a filter over an image, it reduces the size of the image to reduce the amount of parameters and computation in the network. This is repeated 4 more times to make up the convolutional layers.

```
activation_9 (Activation)    (None, 8, 8, 16)          0

max_pooling2d_9 (MaxPooling  (None, 4, 4, 16)          0
2D)

flatten_1 (Flatten)          (None, 256)               0

dense_2 (Dense)              (None, 128)               32896

dropout_1 (Dropout)          (None, 128)               0

dense_3 (Dense)              (None, 9)                 1161
```

Listing 9: Final few layers of champignon-net

The layers above are the final few layers of the model, these layers are the fully connected layers. Before the fully connected layers are activated the final few layers of the convolutional layer are shown. As shown the pooling layer has reduced the image shape down to 4x4. Once the layer is flattened the one-dimensional array is passed into the fully connected layers where the final steps of classification take place.

```
=================================================================
Total params: 435,129
Trainable params: 434,137
Non-trainable params: 992
_________________________________________________________________
```

Listing 10: Parameters of champignon-net

Finally, the total amount of parameters is displayed, showing that the convoluted neural network is quite convoluted. With 434,137 trainable parameters, these parameters include weights and biases of the layer which update during training. The

rest of the parameters are not trained, these include statistics used in batch normalisation.

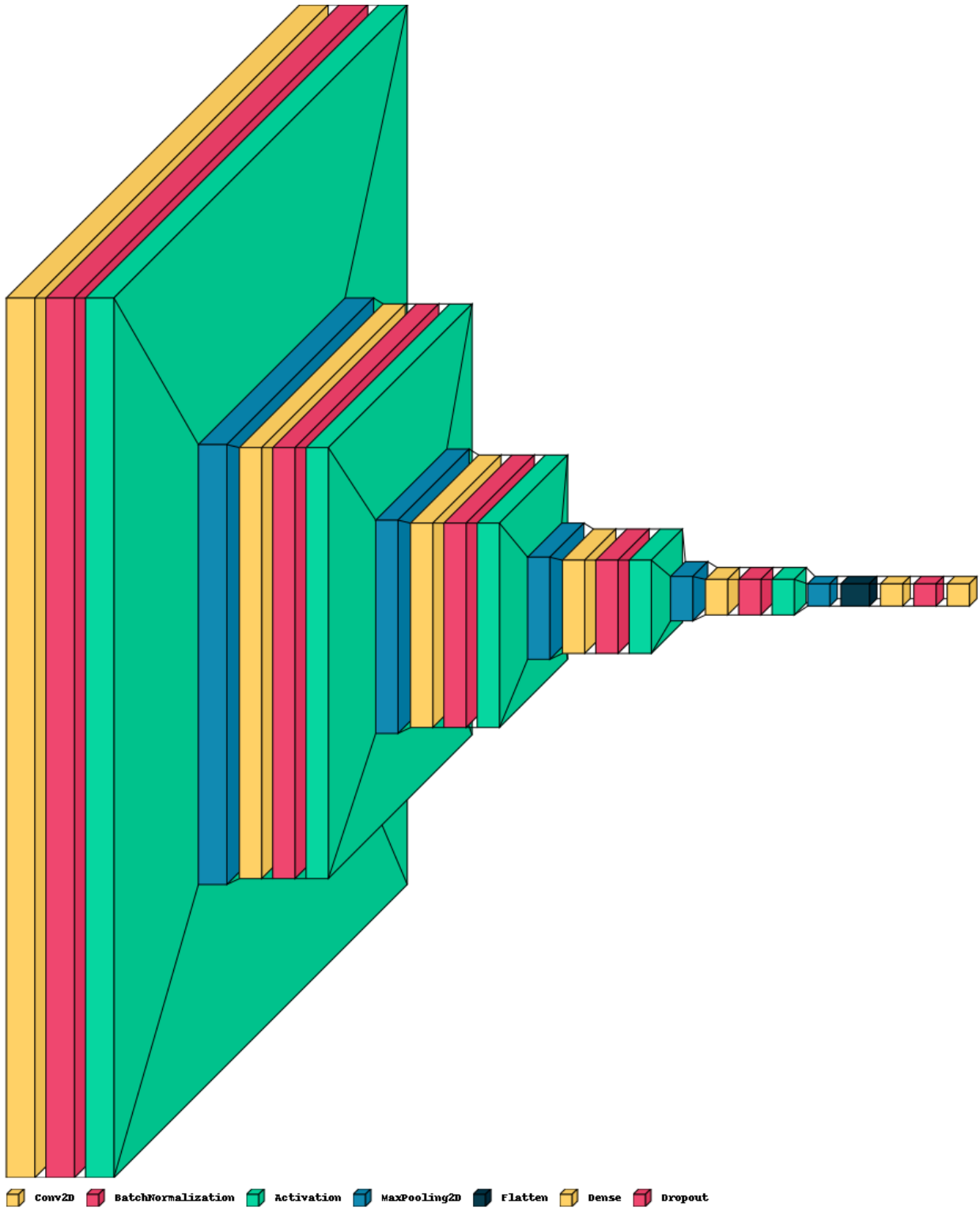


Figure 5.8: Visualisation of Champignon-net

In figure 5.8 champignon-nets architecture can be seen and clearly shows how deep the model is and how the layers are sequentially laid out. This allows for a deeper understanding of the model's structure and how each layer interacts with each other.

Once the model is built the model must be compiled, compiling a model allows for the configuration of the training process for the model and allows for an optimiser, loss function and evaluation metric to be defined.

```
model.compile(
  optimizer = Adam(learning_rate=0.001),
  loss = categorical_crossentropy,
  metrics=['accuracy']
)
```

Listing 11: Compiling champignon-net

Here the Adam optimiser is used with a learning rate of 0.001. The Adam optimiser is a gradient descent method which is computationally efficient and dynamically adjusts the learning rate during training to improve performance (Team Keras, 2023a).

The loss function chosen was categorical cross entropy, a loss function for multi-class and multi-label classification and also goes by Softmax loss (Raul, 2018). This function measures how well the model's predictions match the actual labels in the training data.

5.3.2 Fine Tuning champignon-net

To gain further performance gains, the model's parameters can be automatically set and tested to see what the best configuration is for the model. Here the Keras API has a package called kerastuner, a package that allows for hyperparameters to be defined and changed through a function that randomly chooses different settings for the parameters until the best combination is found. This is the RandomSearch method and is used here to randomly change the parameters until the best ones are found.

First, the model must be rewritten in order to accommodate for these hyper parameters. Simply using the choice function in Kerastuner allows for the random search function to have a choice between parameters.

I will only show segments of this since the structure of the model is the same as champignon-net.

```
model = keras.Sequential()
  model.add(Conv2D(hp.Choice('num_filters_1', [128,256]),(3,3),input_shape=(img_h
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
```

Listing 12: Hyper parameters on champignon-net

This is the first layer of convolution in the model and allows the hyperparameters to choose between different filter sizes allowing for a difference in feature extraction which may or may not effect performance.

```
# Define the tuner
tuner = RandomSearch(
    champignon_model,
    objective='val_accuracy',
    max_trials=10,
    directory='tuner_dir',
    project_name='champignon_model_15')

# Start the search for the best hyperparameters
tuner.search(training_ds,
             epochs=25,
             validation_data=validation_ds)

best_model = tuner.get_best_models(num_models=1)[0]
best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]
```

Listing 13: Fine tuning function

The Randomtuner is then initialised and the model is passed through, the objective is also set, meaning that the goal of the hyperparameters is to maximise the value of the validation accuracy. This will try and ensure that the model can identify new images successfully.

The search is then started, training for 25 epochs 10 times, iterating through each combination of parameters to see which one works best. In this search only 10 run-throughs are made, this is not a lot and more may be needed to truly check all the combinations of the hyperparameters.

The best model is then chosen out of all of the combinations alongside the best hyperparameters, which will be later passed into the model and trained upon.

```
Trial 10 Complete [00h 10m 35s]
val_accuracy: 0.5207768678665161

Best val_accuracy So Far: 0.5207768678665161
Total elapsed time: 01h 29m 51s
```

Listing 14: Tuning Trails

This tuning took quite a long time, nearly 1 hour and 30 minutes were spent tunning champignon-net, which is quite a long time to wait. Google Colab also frequently would run out of computing units during this tuning session making the tuning even slower. However, the best validation accuracy was found and ended up being 52%.

5.3.3 Implementing Other Models

In order to test champignon-net against other models, the models had to be acquired. The TensorFlow Keras Applications allow for premade architectures with pre-trained weights to be used in the environment (TensorFlow, 2023a). Each architecture has its own module in the Keras Applications and each module has sub-architectures inside of it. These architectures use weights that are pre-trained, specifically image-net, a dataset organized by WordNet, image-net aims to put at least 1000 images to these words(Stanford Vision Lab, 2020).

The models that will be used in this project are EfficientNet and Resnet. These models will be downloaded from Tensorflow and placed into a sequential model as the first layer. The sequential model will also include a dense layer with a softmax activation function which will act as an output layer

EfficientNet

EfficientNet was first introduced in Tan and Lees's paper EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (Tan and Le, 2020). It is among the most efficient models and reaches a high accuracy on both image-net and common image classification tasks. EfficientNet introduces a heuristic way to scale the model

and provides up to seven different models. The seven different models ranging from B0 to B7 have a good combination of efficiency and accuracy, scaling its complexity with the name, B0 being less complex but more efficient than B7. This allows for the models to run on a variety of devices with high or low computational power (Team, 2023).

EffcientNet can be easily downloaded in Google Colab using the Keras module.

```
eff_model = tf.keras.applications.EfficientNetB1
    (weights = "imagenet",
     include_top = False,
     input_shape = (img_height, img_width,3),
     pooling = "avg")
```

Listing 15: Initial EfficientNet

The B1 model was used here as anything more complex than this would be overfitting, not only that but the model could use most of the compute units left in Google Colab, which are limited. When the model is initiated the weights are set, the weights chosen here are the pre-trained image-net weights. The input layer is then set to the height and width of the images, as well as how many channels are in the image.

```
eff_model = tf.keras.models.Sequential(
    [
        eff_model,
        tf.keras.layers.Dense(number_of_classes,activation = "softmax")
    ]
)
eff_model.layers[0].trainable = False
eff_model.compile(loss = "categorical_crossentropy",
                  optimizer = "adam",
                  metrics = ["accuracy"])
```

Listing 16: Dense Layers added to EfficientNet

The EfficientNet model is then added to a sequential model, and the output layer is added.

The initial trainability of the EfficientNet layers is then set to false, this freezes the weights in the model, this means during training the weights can not be updated and will act as a fixed feature extractor for the next layer. This can be useful in transfer learning so that the learned representations are not overwritten during training allowing the model to better generalise a new dataset. This also helps with overfitting and reduces the risk of the model memorizing the training data.

```
Model: "efficientnet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb1 (Functional)  (None, 1280)             6575239

 dense (Dense)               (None, 9)                 11529

=================================================================
Total params: 6,586,768
Trainable params: 11,529
Non-trainable params: 6,575,239
_________________________________________________________________
```

Listing 17: Layers in the sequential model and the parameters of each

Here the sequential model is summarised, showing the weights of the EfficientNet mentioned earlier and the dense layer. As seen in the summary the only trainable layer is the dense layer which produces the probability distribution over nine of the 9 classes in the dataset.

The model is then trained for 50 epochs and makes an early stop around 44 epochs.

ResNet

ResNet or Residual Network was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper "Deep Residual Learning for Image Recognition" (He et al., 2015). They propose a framework to ease the training of networks that are extremely deep, they show in the paper that these residual networks are easier to optimise and gain accuracy from increased depth.

This network in Keras has multiple architectures to choose from, ranging from ResNet50 to ResNet152, the depth increasing with the value in the name (Team Keras, 2023b).

This project will utilise the ResNet101 model, as it is a good middle ground between complexity and efficentcy. As previously mentioned Google Colab limits the amount of compute units that can be used meaning that a model that isnt too computationally complex but also priortises accuracy is optimal.

ResNet101 was implemented the same way the EffientNet was, downloaded from Keras, placed into a sequential model with an output layer. The Resnets weights were then set to not be trainable.

```
Model: "resnet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet101 (Functional)      (None, 2048)              42658176

 dense_1 (Dense)             (None, 9)                 18441

=================================================================
Total params: 42,676,617
Trainable params: 18,441
Non-trainable params: 42,658,176
_________________________________________________________________
```

Listing 18: Resnet summary

As seen above resnet is much more complex than the effcienet, this could bad for the training since it could encounter overfitting, however the earlystop callback should prevent that from happening.

ResNet was ran over 50 epochs and the early stop callback was called after 35 epochs.

Chapter 6

Results & Discussion

All results in this chapter were calculated and visualised using both TensorFlows Tensorboard and Pandas. To keep the results consistent a seed was set in the dataset, this is so when the shuffle of the dataset happens the results are the same each time. This could further be implemented into TensorFlow , extending the seed to the weights and bias of each model could make the results more repeatable and reliable.

All testing made use of the Google Colab GPU as using the CPU would have taken alot longer, the CPU would have also reached Google Colabs 12 hour limit.

6.1 Dataset Results

The requirement of dataset aquision was met , the result of the split and distrobution of the classes were accetable and suitable fro training

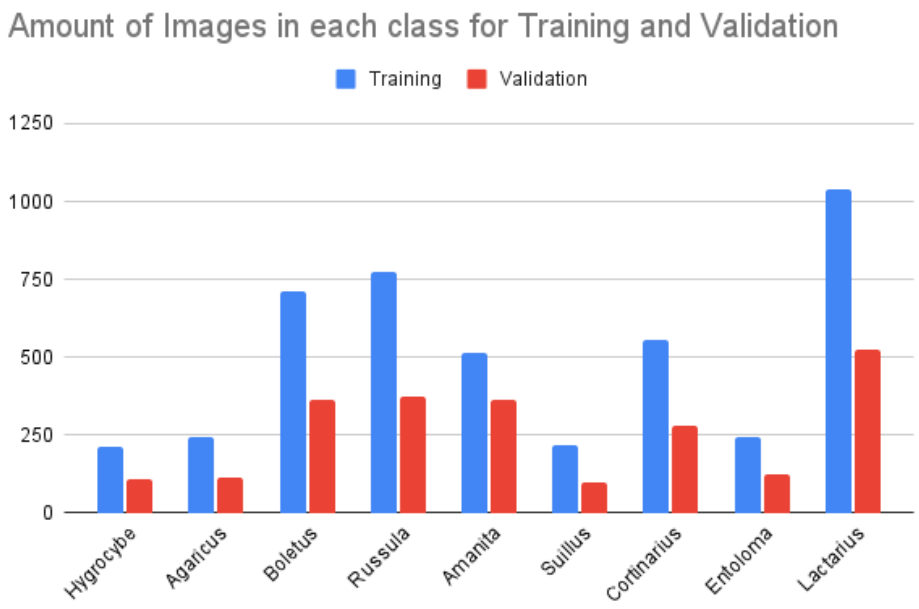


Figure 6.1: Amount of images in each class for both datasets

One trend that can be seen in figure 6.1 is that there is a class imbalance in the dataset. This could have had an affect on the models having difficulty learning as it may have been a lot harder for the models to distinguish a class of a smaller proportion from a more prevalent class. This may have led to the models having a lower accuracy and recall rate for the smaller classes. To mitigate this techniques such as data augmentation could have improved the models performance.

6.2 Champignon-net Results

6.2.1 Standard Champignon-net

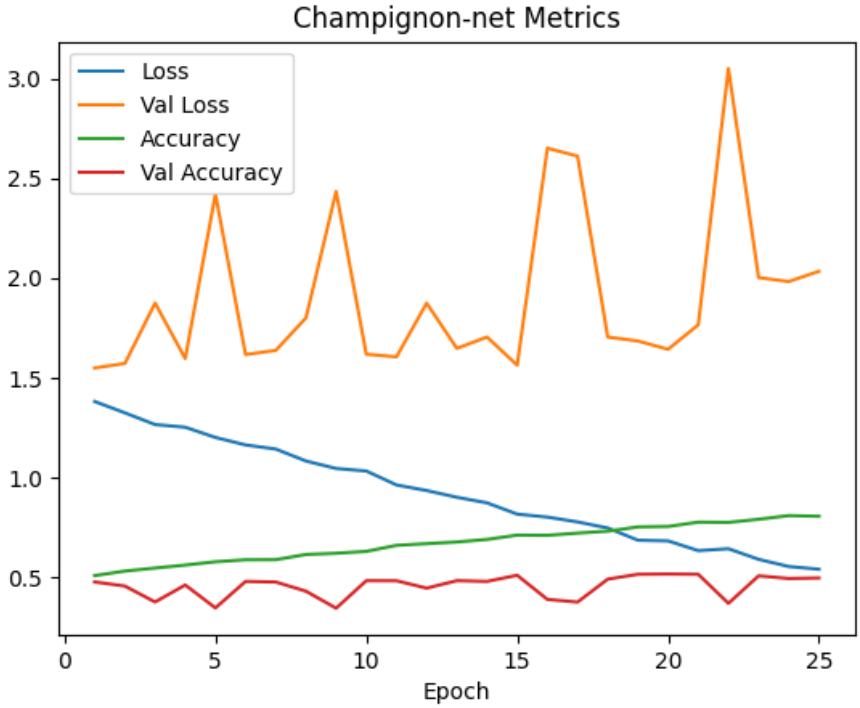


Figure 6.2: Champignon-net Results

After running champignon-net for 50 epochs the callback function early stop was called on the 24th epoch, this was due to stagnation of the validation accuracy. Overall however the model performed decent, having both the validation accuracy and the accuracy start low and increase over time. This trend can be seen in figure 6.2. As seen in the figure there are fluctuations in the metrics over time, for example, there are multiple spikes in validation loss and not only that validation loss does not drop throughout the training period. This could be due to overfitting, the model likely starts memorizing training data and performing poorly on new data. The dip in validation accuracy at epoch 5 could indicate that the model is struggling to generalise to new data.

Further evaluation using the TensorFlow evaluate function shows the following metrics.

```
loss: 1.6435 - accuracy: 0.5154
```

First, the loss function indicates that the model overall is not performing very well on new data, which can also be seen in figure 6.2. The accuracy means that the model will correctly predict the mushrooms only 51% of the time. This shows the model isn't the best at classifying mushrooms, this could be due to overfitting or could be an issue with the dataset. To identify this further metrics could be generated, such as precision, recall and an F1 score, all of which could be acquired from a confusion matrix. Unfortunately due to a lack of computing units in Google Colab, a confusion matrix could not be generated, there for further insights into why the model is behaving this way could not be made.

6.2.2 Fine Tuned Champignon-net

Champignon-net was fine-tuned using the best parameters found in the random search. As previously mentioned this took 1 hour and 30 minutes and slightly improved the model.

The best validation accuracy found by the model was found to be 52%, increasing the accuracy on new-found data by only 1%

However, after training this model, the results got even worse. The new model was trained over 50 epochs

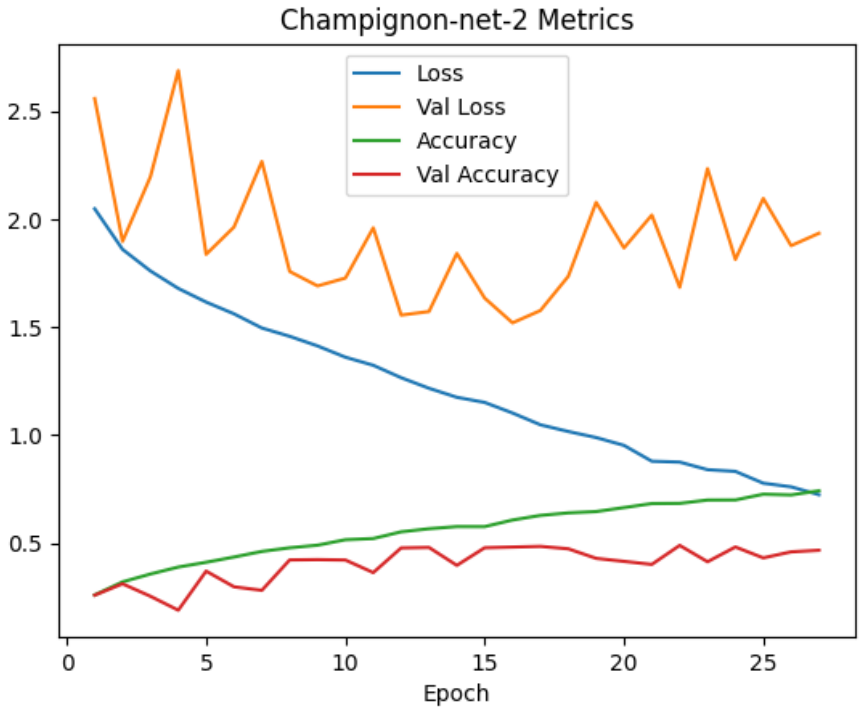


Figure 6.3: Champignon-net-2 Results

Figure 6.3 was generated using the training metrics throughout 27 epochs, this is because the early stop call back was called because of the stagnation of the validation accuracy. The trend here is highly similar to the original model because not much success was made in finding the better hyperparameters. In fact when evaluating the new model the performance was even worse.

```
loss: 1.6857 - accuracy: 0.4892
```

With a decrease in 3% accuracy when evaluating new images, this model is not much worse than the champignon-net model but still bad overall. This again could be due to overfitting, however, as stated previously since a confusion matrix was never calculated further insights can not be made. To further delve into the results graphs were made using the Tensorflows Tensorboard package.

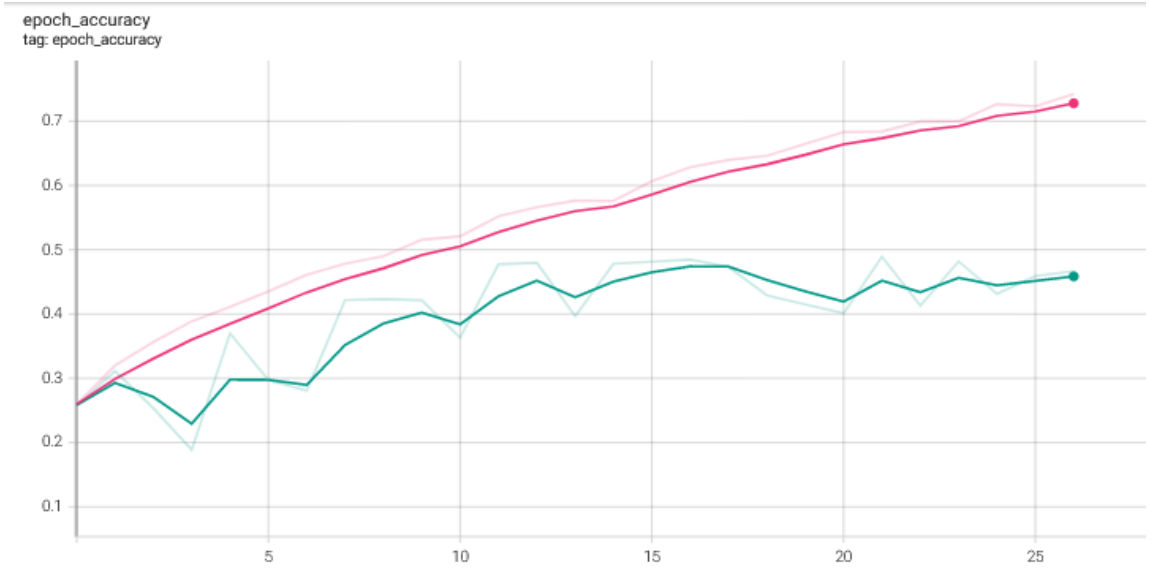


Figure 6.4: Champignon-net-2 TensorBoard Accuracy Results | Red - accuracy , Blue - val_accuracy

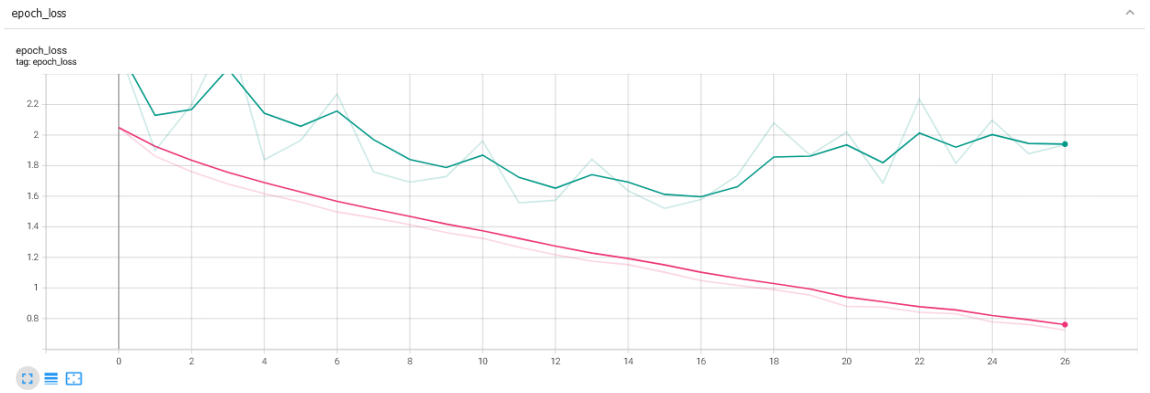


Figure 6.5: Champignon-net-2 TensorBoard Loss Results | Red - loss, Blue - val_loss

As seen in figures 6.4 and 6.5 the trends are very clear here, accuracy goes up over all and loss goes down over time. This would be great if the validation loss and accuracy followed the same trend, however, they don't, validation accuracy does go up but stagnates around epoch 22/23 whilst validation loss decreases for a while and then stagnates.

6.3 EfficientNet & ResNet Results

6.3.1 EfficientNetB1

The results for EfficientNetB1 were acquired using the panda library and TensorBoard. Similarly to champignon-net, no confusion matrix could be acquired due to constraints limiting the amount of insight into the model's behaviour.

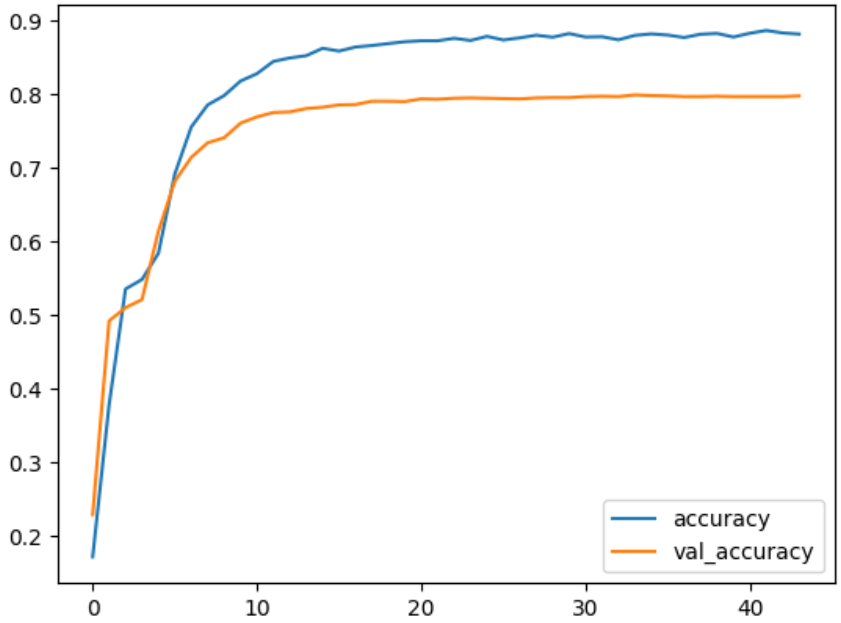


Figure 6.6: EfficientNetB1 Accuracy Metrics

As seen in figure 6.6 the training accuracy of this model is very high, almost reaching 90% accuracy after 44 epochs before the early stop callback was called due to the stagnating validation accuracy. The validation accuracy increases alongside the training accuracy until it stagnates at around 79%, this is good as it almost meets the requirement of 80% accuracy, in fact, the precise accuracy according to the training epoch is 79.81%. However the stagnation of the validation accuracy shows that the model may be slightly overfitting, this could be prevented by using regularisation techniques such as drop-out layers or using a less complex model such as EfficientNetB0.

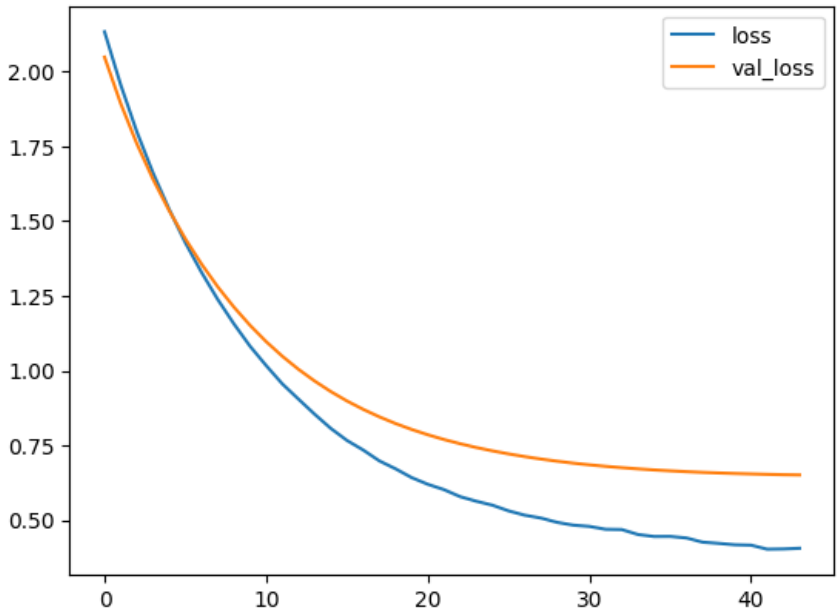


Figure 6.7: EfficientNetB1 Loss Metrics

The model shows a fall in loss, the lower the loss the closer the model is to predicting the correct class. The validation loss follows this trend and is not too far behind the training loss meaning that there is only a slight case of overfitting in this model.

Evaluating the model further using the TensorBoard evaluation function shows that this trend persists.

```
loss: 0.6716 - accuracy: 0.7995
```

The accuracy of this model when tested on new images is high and extremely close to the required accuracy of 80%. For this model visualisations of the mushrooms the model predicted were made.

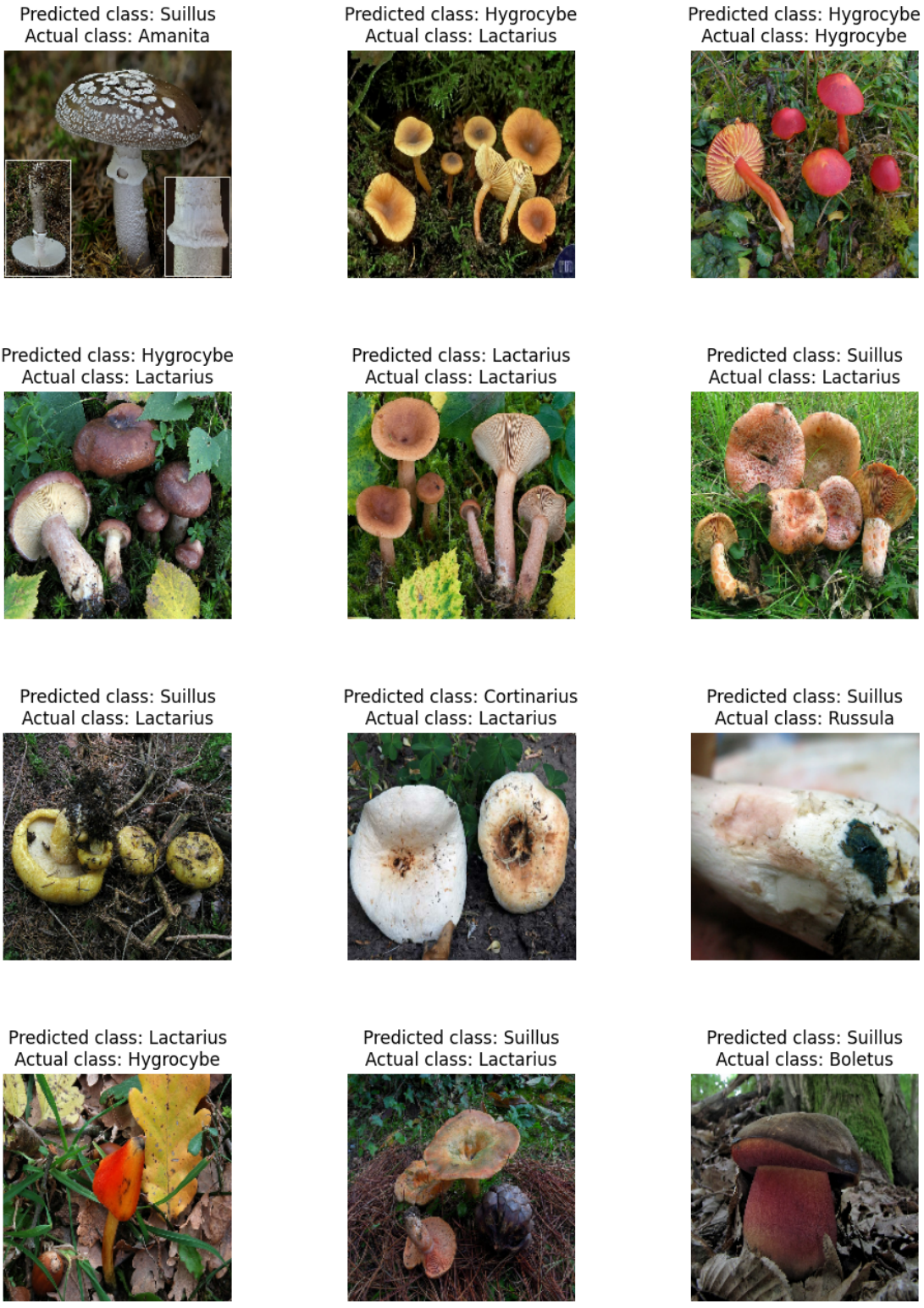


Figure 6.8: EfficientNetB1 predictions visualised

The visualisation seen in figure 6.8 is a very small portion of the validation dataset, therefor is not the best representation of what the model has actually predicted.

6.3.2 ResNet101

The results for ResNet101 were acquired using the panda library and TensorBoard. Similar to the other models a confusion matrix could not be acquired due to the same reasons.

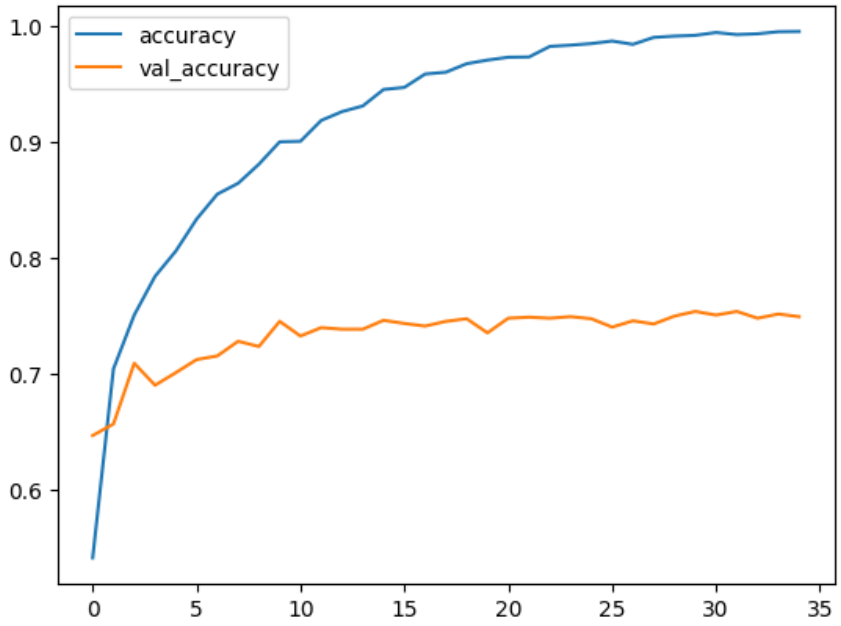


Figure 6.9: ResNet101 Accuracy Metrics

The training accuracy seen 6.9 increases significantly over the 35 epochs, getting to 99% accuracy. The validation accuracy however stagnates at around 75%, although this is close to the required accuracy of 80% the validation accuracy is stagnating, this could be due to overfitting. To mitigate this a dropout layer could be added or a smaller model such as ResNet50 could be used to reduce the complexity of the model and prevent generalisation of the testing data.

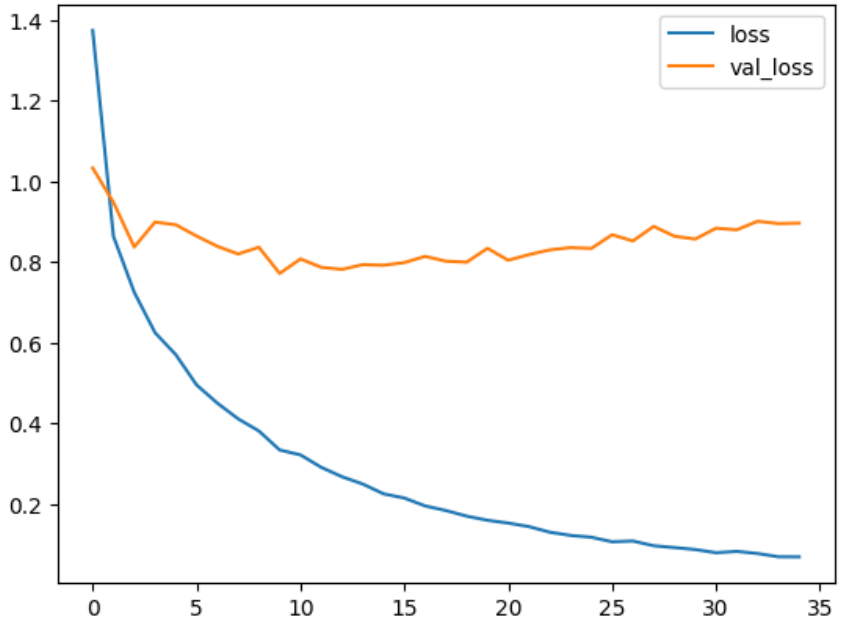


Figure 6.10: ResNet101 Loss Metrics

The loss metrics seen in 6.11 show a similar trend to the accuracy, for the validation loss stagnates around the same epoch as the validation accuracy. This further implies that the model is overfitting to the dataset. Alongside the stagnation, the validation loss seems to fluctuate throughout the training, climbing slightly then shortly after falling.

It is clear that the ResNet101 model is overfitting, but again due to the fact no confusion matrix has been produced a conclusion on why it is overfitting can not be made.

Visualisation that was similar to what was done for EfficientNetB1 was implemented. Showing the mushrooms predicted by the model alongside the actual mushroom.

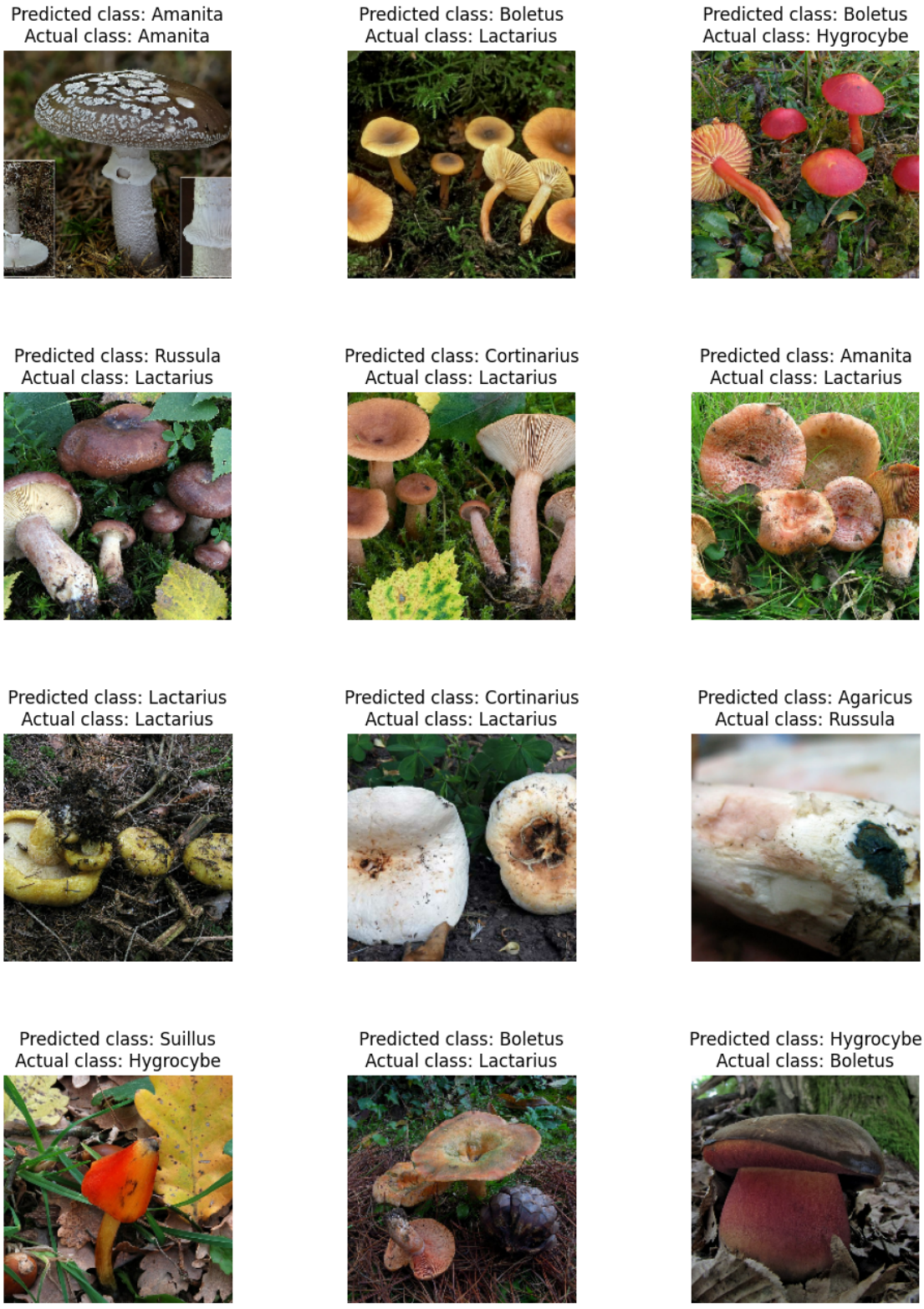


Figure 6.11: ResNet101 predictions visualised

6.4 Comparison

Model	Evaluation Accuracy	Evaluation Loss	Validation Accuracy	Validation Loss	Training Accuracy	Training Loss
Champignon-net	0.515	1.643	0.495	2.035	0.805	0.594
Champignon-net-2	0.489	1.685	0.466	1.936	0.742	0.724
EfficientNetB1	0.799	0.671	0.798	0.651	0.882	0.406
ResNet101	0.774	0.756	0.743	0.889	0.990	0.097

Table 6.1: Results of all models

Table 6.1 compares the metrics for each model. According to the results, EfficientNetB1 has the best results, having a high validation and evaluation accuracy and a low loss for validation and evaluation. This means EfficientNetB1 learns new images of mushrooms well and can classify the correct mushrooms 80% of the time, meeting the required accuracy. The models that performed the worse here were models that were either not very deep or were too complex for the dataset, overfitting the results.

The self-made dataset Champignon-net and the fine-tuned Champignon-net2 both performed worse than the other models, this is due to its low complexity and lack of pre-trained weights that were used to train the other models. Against each other however, surprisingly the non fine-tuned model Champignon-net performed better than the fine-tuned model Champignon-net2. Champignon-net obtained a validation accuracy of almost 50% and an evaluation accuracy of over 50% meaning Champignon-net successfully classified a mushroom 50% of the time. Champignon-net2 however performed worse obtaining 48.9% in evaluation accuracy and 46% in validation accuracy, although not much worse it is still unexpected since the fine-tuning was expected to improve the model.

Both pre-made models performed better than Champignon-net since they are both state-of-the-art deep learning models and are known to perform well on image classification tasks as well as utilising image-net.

Models	Accuracy	Study
InceptionV3 (Raw image)	0.82	(Zahan et al., 2021)
InceptionV3 (CLAHE)	0.88	(Zahan et al., 2021)
EfficientNetB5	0.92	(Kiss and Zúni, 2021)
Support Vector Machine	0.80	(Anil, Gupta and Arora, 2019)
EfficientNetB1	0.79	This project
ResNet101	0.77	This project
Champignon-net	0.51	This project

Table 6.2: Comparison between results obtained from literature and results obtained from this project

Table 6.2 shows the models tested in this project against the projects discussed in the literature review. The models trained in this project perform much worse than the others, a support vector machine (a traditional machine learning model) being better than the deep learning networks shows that it may not just be the model that is the issue. The low accuracy rates could be due to the dataset and its distribution of classes, along with the similarity between mushrooms.

Chapter 7

Conclusion

Overall most main objectives if not all were achieved, a high-quality dataset was acquired and split into training and testing data, three convoluted neural networks were trained on the dataset (one being self-made) and the results were compared. However, objectives such as obtaining an 80% accuracy were too ambitious, as making and fine-tuning a model to do so proved more difficult than originally presumed. A sufficient amount of research was done into the problem, as a detailed literature review was made spanning an explanation of machine learning and its implementation in previous works and previous usage of datasets in other studies. The methodology proved suitable as a working artefact was produced alongside metrics to back up that machine learning techniques such as CNNs are suitable for classifying mushrooms.

To focus on the acquisition of the dataset, the dataset acquired was suitable for the task as the images were extremely high quality, this allowed for the models to extract features a lot better. The amount of images in the dataset was also sufficient, however, the distribution of the images wasn't optimal and could have been mitigated by utilising pre-processing techniques. The dataset distribution could have been a factor in the overfitting which was apparent in most of the models since an uneven distribution can lead a model to become biased towards a dominant class.

Regarding machine learning methods, Champignon-net could have been improved by making the model more complex and including more mitigations for overfitting such as more dropout layers. Overall Champignon-net performed below the expected performance metric and was quite disappointing, future development could improve this model to make it more fit for the task of classifying mushrooms. The two pre-

trained models performed as expected, with higher performance metrics than the self-made model, however lower than the previously implemented models in previous studies. Again further research and development are needed on the two models' interaction with the dataset.

To make an easily accessible model used for mushroom classification more research and further improvements would need to be made in future endeavours.

References

Anil, Abhishek, Hardik Gupta and Monika Arora (Sept. 2019). 'Computer vision based method for identification of freshness in mushrooms'. In: *2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*. Vol. 1, pp. 1–4. DOI: 10.1109/ICICT46931.2019.8977698 (cit. on pp. 13, 15, 69).

Atlassian (2023). *What is a Kanban Board?* en. URL: https://www.atlassian.com/agile/kanban/boards (visited on 6th May 2023) (cit. on p. 19).

Auernhammer, Hermann (2001). *Precision farming — the environmental challenge*. en. DOI: 10.1016/S0168-1699(00)00153-8. URL: https://reader.elsevier.com/reader/sd/pii/S0168169900001538?token=3450C9957F0DE595AE3EB4C9A0211A965919D originRegion=eu-west-1&originCreation=20230130183355 (cit. on p. 3).

Beedle, Mike (2001). *Manifesto for Agile Software Development*. URL: https://agilemanifesto.org/ (visited on 6th May 2023) (cit. on p. 18).

Benjamin, Denis R. (1995). *Mushrooms: poisons and panaceas : a handbook for naturalists, mycologists, and physicians*. eng. New York: W.H. Freeman. ISBN: 978-0-7167-2600-5 978-0-7167-2649-4. URL: http://archive.org/details/mushroomspoisons0000benj (cit. on p. 2).

Camridge Dictionary (May 2023). *champignon*. en. URL: https://dictionary.cambridge.org/dictionary/french-english/champignon (visited on 9th May 2023) (cit. on p. 44).

CatoDogo (Feb. 2019). *Mushrooms classification - Common genus's images*. URL: https://www.kaggle.com/datasets/maysee/mushrooms-classification-common-genuss-images (cit. on p. 34).

Cortes, Corinna and Vladimir Vapnik (Sept. 1995). 'Support-vector networks'. en. In: *Machine Learning* 20.3, pp. 273–297. ISSN: 1573-0565. DOI: 10.1007/BF00994018. URL: https://doi.org/10.1007/BF00994018 (cit. on p. 11).

Data Detective (Jan. 2020). *Finally: Why We Use an 80/20 Split for Training and Test Data Plus an Alternative Method (Oh Yes...)* en. URL: https://towardsdatascience.com/finally-why-we-use-an-80-20-split-for-training-and-test-data-plus-an-alternative-method-oh-yes-edc77e96295d (visited on 9th May 2023) (cit. on p. 40).

DeepAI (May 2019). *Batch Normalization*. URL: https://deepai.org/machine-learning-glossary-and-terms/batch-normalization (visited on 9th May 2023) (cit. on p. 45).

Dictionary, Oxford English (Apr. 2023). *mycophilia, n. : Oxford English Dictionary*. URL: https://www.oed.com/viewdictionaryentry/Entry/241690 (cit. on p. 1).

Eloundou, Tyna et al. (Mar. 2023). 'GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models'. In: arXiv:2303.10130 [cs, econ, q-fin]. DOI: 10.48550/arXiv.2303.10130. URL: http://arxiv.org/abs/2303.10130 (cit. on p. 12).

Gannt (2023). *Gantt.com*. en. URL: https://www.gantt.com (visited on 6th May 2023) (cit. on p. 20).

GBIF (n.d.). en. URL: https://www.gbif.org/ (cit. on p. 4).

Goodfellow, Ian, Yoshua Bengio and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press (cit. on p. 10).

Google (2012). *Personal Cloud Storage & File Sharing Platform - Google*. en. URL: https://www.google.com/drive/ (visited on 8th May 2023) (cit. on p. 32).

– (2018). *Google One – Cloud Storage, automatic phone backup, VPN and more*. URL: https://one.google.com/about (visited on 8th May 2023) (cit. on p. 32).

– (2023a). *Google Colab*. URL: https://research.google.com/colaboratory/faq.html (visited on 8th May 2023) (cit. on p. 25).

– (2023b). *What makes TPUs fine-tuned for deep learning?* en-US. URL: https://cloud.google.com/blog/products/ai-machine-learning/what-makes-tpus-fine-tuned-for-deep-learning (visited on 8th May 2023) (cit. on p. 29).

Gov, UK (2023). *Agile methods: an introduction - Service Manual - GOV.UK*. URL: https://www.gov.uk/service-manual/agile-delivery/agile-methodologies (visited on 6th May 2023) (cit. on p. 19).

Hastie, Trevor, Robert. Tibshirani and Jerome Friedman (2009). *Elements of Statistical Learning: data mining, inference, and prediction. 2nd Edition*. URL: https://hastie.su.domains/ElemStatLearn/ (cit. on p. 10).

He, Kaiming et al. (Dec. 2015). *Deep Residual Learning for Image Recognition*. arXiv:1512.03385 [cs]. DOI: 10.48550/arXiv.1512.03385. URL: http://arxiv.org/abs/1512.03385 (visited on 10th May 2023) (cit. on p. 54).

IBM (2023a). en-us. URL: https://www.ibm.com/topics/deep-learning (cit. on p. 11).

– (2023b). *What is Machine Learning? | IBM*. en-us. URL: https://www.ibm.com/topics/machine-learning (cit. on p. 2).

iNaturalist, LLC (Apr. 2023). *Seek by iNaturalist*. URL: https://www.inaturalist.org/ (cit. on p. 4).

Jordan, Michael (2004). *The encyclopedia of fungi of Britain and Europe*. eng. London : Frances Lincoln. ISBN: 978-0-7112-2378-3 978-0-7112-2379-0. URL: http://archive.org/details/encyclopediaoffu0000jord (cit. on p. 2).

Jupyter (2015). *Project Jupyter*. en. URL: https://jupyter.org (visited on 8th May 2023) (cit. on p. 25).

Kaggle (2023). *Kaggle: Your Home for Data Science*. en. URL: https://www.kaggle.com/ (visited on 7th May 2023) (cit. on p. 23).

Kaplan, Jared et al. (Jan. 2020). *Scaling Laws for Neural Language Models*. arXiv:2001.08361 [cs, stat]. DOI: 10.48550/arXiv.2001.08361. URL: http://arxiv.org/abs/2001.08361 (visited on 9th May 2023) (cit. on p. 40).

Keras (2023). *Keras documentation: About Keras*. en. URL: https://keras.io/about/ (visited on 7th May 2023) (cit. on p. 24).

Kiss, Norbert and Lászlóc Zúni (2021). 'Mushroom Image Classification with CNNs: a Case-Study of Different Learning Strategies *'. In: *2021 12th International Symposium on Image and Signal Processing and Analysis (ISPA)*. ISBN: 9781665426398. DOI: 10.1109/ISPA52656.2021.9552053. URL: https://svampe.databasen.org/en/imagevision (cit. on pp. 8, 15, 69).

Labs, Notion (2023). *Your wiki, docs & projects. Together*. en-us. URL: https://www.notion.so (visited on 6th May 2023) (cit. on p. 20).

Lorente, Òscar, Ian Riera and Aditya Rana (May 2021). 'Image Classification with Classic and Deep Learning Techniques'. In: arXiv:2105.04895 [cs]. DOI: 10.48550/arXiv.2105.04895. URL: http://arxiv.org/abs/2105.04895 (cit. on pp. vi, 12).

Marsland, Stephen (Sept. 2015). *Machine Learning: An Algorithmic Perspective, Second Edition*. en. Google-Books-ID: $y_oYCwAAQBAJ$. CRC Press. ISBN: 9781498759786 (cit. on pp. vi, 10).

Matplotlib development team (2012). *Matplotlib — Visualization with Python*. URL: https://matplotlib.org/ (visited on 8th May 2023) (cit. on p. 26).

McCulloch, Warren S. and Walter Pitts (Dec. 1943). 'A logical calculus of the ideas immanent in nervous activity'. en. In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259 (cit. on p. 10).

Milad, Yousefi (2020). *Error when train on customized dataset: Invalid JPEG data or crop window, data size 36864 · Issue #455 · tensorflow/tpu*. en. URL: https://github.com/tensorflow/tpu/issues/455 (visited on 8th May 2023) (cit. on p. 38).

Mitchell, Tom M. (Tom Michael) (1997). *Machine Learning*. eng. New York: McGraw-Hill. ISBN: 9780070428072. URL: http://archive.org/details/machinelearning0000mitc (cit. on p. 9).

Morgan, Jacob (2023). *Classification Of Mushrooms Using Machine Learning Interim Report* (cit. on pp. vi, 21).

Nair, Vinod and Geoffrey E Hinton (n.d.). 'Rectified Linear Units Improve Restricted Boltzmann Machines'. en. In: () (cit. on pp. 13, 45).

Numpy Team (2005). *NumPy - About Us*. URL: https://numpy.org/about/ (visited on 8th May 2023) (cit. on p. 26).

NVIDIA (June 2012). *NVIDIA System Management Interface*. en-US. URL: https://developer.nvidia.com/nvidia-system-management-interface (visited on 8th May 2023) (cit. on p. 30).

– (July 2013). *CUDA Toolkit - Free Tools and Training*. en-US. URL: https://developer.nvidia.com/cuda-toolkit (visited on 8th May 2023) (cit. on p. 30).

O'shea, Keiron and Ryan Nash (2015). *An Introduction to Convolutional Neural Networks*. Tech. rep. arXiv: 1511.08458v2 (cit. on pp. vi, 3, 12, 13).

OpenAI (2023a). en-US. URL: https://openai.com/blog/chatgpt (cit. on p. 11).

– (Mar. 2023b). 'GPT-4 Technical Report'. In: arXiv:2303.08774 [cs]. DOI: 10.48550/arXiv.2303.08774. URL: http://arxiv.org/abs/2303.08774 (cit. on p. 11).

Python Software Foundation (2023). *pathlib — Object-oriented filesystem paths*. URL: https://docs.python.org/3/library/pathlib.html (visited on 8th May 2023) (cit. on p. 36).

Raul, Gomez (2018). *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*. URL: https://gombru.github.io/2018/05/23/cross_entropy_loss/ (visited on 9th May 2023) (cit. on p. 48).

Reza, Ali M. (Aug. 2004). 'Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for Real-Time Image Enhancement'. en. In: *Journal of VLSI signal processing systems for signal, image and video technology* 38.1, pp. 35–44. ISSN: 0922-5773. DOI: 10.1023/B:VLSI.0000028532.53893.82. URL: https://doi.org/10.1023/B:VLSI.0000028532.53893.82 (cit. on p. 7).

Rolfe R.T and Rolfe F.W (1925). *The romance of the fungus world; an account of fungus life in its numerous guises, both real and legendary*. eng. London : Chapman & Hall. ltd. URL: http://archive.org/details/romanceoffungusw00rolf (cit. on p. 1).

Rossum, Guido van (May 2023). *Welcome to Python.org*. en. URL: https://www.python.org/ (visited on 7th May 2023) (cit. on p. 23).

Russell, Jonathan R. et al. (Sept. 2011). 'Biodegradation of Polyester Polyurethane by Endophytic Fungi'. In: *Applied and Environmental Microbiology* 77.17. Publisher: American Society for Microbiology, pp. 6076–6084. DOI: 10.1128/AEM.00521-11. URL: https://journals.asm.org/doi/full/10.1128/aem.00521-11 (cit. on p. 1).

Schulzová, V. et al. (Jan. 2009). 'Agaritine content of 53 *Agaricus* species collected from nature'. en. In: *Food Additives & Contaminants: Part A* 26.1, pp. 82–93. ISSN: 1944-0049, 1944-0057. DOI: 10.1080/02652030802039903. URL: http://www.tandfonline.com/doi/abs/10.1080/02652030802039903 (cit. on p. 8).

Sharma, Abhinav et al. (2021). 'Machine Learning Applications for Precision Agriculture: A Comprehensive Review'. In: *IEEE Access* 9. Conference Name: IEEE Access, pp. 4843–4873. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3048415 (cit. on p. 4).

Sivic and Zisserman (Oct. 2003). 'Video Google: a text retrieval approach to object matching in videos'. In: *Proceedings Ninth IEEE International Conference on Computer Vision*, 1470–1477 vol.2. DOI: 10.1109/ICCV.2003.1238663 (cit. on p. 12).

Stanford Vision Lab (2020). *ImageNet*. URL: https://www.image-net.org/index.php (visited on 10th May 2023) (cit. on p. 51).

Szegedy, Christian et al. (June 2015). 'Going deeper with convolutions'. en. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, pp. 1–9. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298594. URL: http://ieeexplore.ieee.org/document/7298594/ (cit. on p. 14).

Tan, Mingxing and Quoc V. Le (Sept. 2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv:1905.11946 [cs, stat]. DOI: 10.48550/arXiv.1905.11946. URL: http://arxiv.org/abs/1905.11946 (visited on 12th Apr. 2023) (cit. on p. 51).

Team, Keras (2023). *Keras documentation: Image classification via fine-tuning with EfficientNet*. en. URL: https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/ (visited on 10th May 2023) (cit. on p. 52).

– (2023a). *Keras documentation: Adam*. en. URL: https://keras.io/api/optimizers/adam/ (visited on 9th May 2023) (cit. on p. 48).

– (2023b). *Keras documentation: ResNet and ResNetV2*. en. URL: https://keras.io/api/applications/resnet/ (visited on 10th May 2023) (cit. on p. 54).

TensorBoard (2023). *TensorBoard*. en. URL: https://www.tensorflow.org/tensorboard (visited on 8th May 2023) (cit. on p. 25).

TensorFlow (Nov. 2022). *tf.keras.applications.efficientnet.EfficientNetB0 | TensorFlow v2.11.0*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet/EfficientNetB0 (cit. on p. 15).

– (2023a). *Module: tf.keras.applications | TensorFlow v2.12.0*. en. URL: https://www.tensorflow.org/api_docs/python/tf/keras/applications (visited on 10th May 2023) (cit. on p. 51).

– (Jan. 2023b). *TensorFlow*. URL: https://www.tensorflow.org (cit. on pp. 15, 24).

– (2023c). *tf.data.Dataset | TensorFlow v2.12.0*. en. URL: https://www.tensorflow.org/api_docs/python/tf/data/Dataset (visited on 7th May 2023) (cit. on p. 24).

– (2023d). *tf.data.Dataset | TensorFlow v2.12.0*. en. URL: https://www.tensorflow.org/api_docs/python/tf/data/Dataset (visited on 7th May 2023) (cit. on p. 39).

– (2023e). *tf.keras.callbacks.EarlyStopping | TensorFlow v2.12.0*. en. URL: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping (visited on 9th May 2023) (cit. on p. 43).

– (2023f). *tf.keras.callbacks.ReduceLROnPlateau | TensorFlow v2.12.0*. en. URL: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau (visited on 9th May 2023) (cit. on p. 43).

– (2023g). *Use a GPU | TensorFlow Core*. en. URL: https://www.tensorflow.org/guide/gpu (visited on 8th May 2023) (cit. on p. 29).

Wibowo, Agung et al. (Mar. 2018). 'Classification algorithm for edible mushroom identification'. In: *2018 International Conference on Information and Communications Technology (ICOIACT)*, pp. 250–253. DOI: 10.1109/ICOIACT.2018.8350746 (cit. on p. 11).

Wikipedia (Apr. 2023). *Freemium*. en. Page Version ID: 1150153685. URL: https://en.wikipedia.org/w/index.php?title=Freemium&oldid=1150153685 (visited on 6th May 2023) (cit. on p. 20).

Zahan, Nusrat et al. (2021). 'A Deep Learning-Based Approach for Edible, Inedible and Poisonous Mushroom Classification; A Deep Learning-Based Approach for Edible, Inedible and Poisonous Mushroom Classification'. In: *2021 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*. ISBN: 9781665414609, pp. 27–28. DOI: 10.1109/ICICT4SD50815.2021.9396845 (cit. on pp. vi, 7, 8, 14, 15, 69).

Zhou, Zhi-Hua (Aug. 2021). *Machine Learning*. en. Springer Nature. ISBN: 9789811519673 (cit. on p. 9).