NEAR EAST UNIVERSITY

INSTITUTE OF GRADUATE STUDIES

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

LEVERAGING ARTIFICIAL INTELLIGENCE FOR TICK TAXONOMY: EXPLORING DEEP LEARNING MODELS FOR PARASITE CLASSIFICATION

M.Sc. THESIS

Ibrahim AME

Nicosia

June, 2024

IBRAHIM AHMED AME

LEVERAGING ARTIFICIAL INTELLIGENCE FOR TICK TAXONOMY: EXPLORING DEEP LEARNING MODELS FOR PARASITE CLASSIFCATION

MASTER THESIS

2024

NEAR EAST UNIVERSITY

INSTITUTE OF GRADUATE STUDIES

DEPARTMENT OF ARTIFICIAL INTELLIGENCE ENGINEERING

LEVERAGING ARTIFICIAL INTELLIGENCE FOR TICK TAXONOMY: EXPLORING DEEP LEARNING MODELS FOR PARASITE CLASSIFICATION

M.Sc. THESIS

Ibrahim AME

Supervisor

Professor Fadi AL-TURJMAN

Nicosia

June, 2024

Approval

We certify that we have read the thesis submitted by **Ibrahim AME** titled **" LEVERAGING ARTIFICIAL INTELLIGENCE FOR TICK TAXONOMY: EXPLORING DEEP LEARNING MODELS FOR PARASITE CLASSIFICATION "** and that in our combined opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Master of Artifiicial Intelligence Engineering.

Examining Committee	Name-Surname	Signature
Head of the Committee:	Prof Dr. Fadi Al-Turjman	[illegible]............
Committee Member:	Asst. Prof. Dr. Abdullahi Umar Ibrahim	[illegible]............
Committee Member:	Asst. Prof. Dr. Mubarak Auwalu Saleh	[illegible]............
Supervisor:	Prof Dr. Fadi Al-Turjman	[illegible]............

Approved by the Head of the Department

…../…../2024

[illegible]..........................

Prof.Dr. Fadi Al-Turjman

Head of Department

Approved by the Institute of Graduate Studies

NEAR EAST UNIVERSITY INSTITUTE OF GRADUATE STUDIES

…../…../20…

[illegible]

Prof. Dr. Kemal Hüsnü Can Başer

Head of the Institute

Declaration

I hereby declare that all information, documents, analysis and results in this thesis have been collected and presented according to the academic rules and ethical guidelines of Institute of Graduate Studies, Near East University. I also declare that as required by these rules and conduct, I have fully cited and referenced information and data that are not original to this study.

Ibrahim Ahmed Ame

…../…../…..

Day/Month/Year

Acknowledgments

First and foremost, gratitude is given to Allah, for guiding me through this research and giving me the energy and wisdom to figure out the intricacies of this project. To my parents, Prof. Ahmed Mohammed Ame and mother, Dr. Khalila Ali Sharif. Thank you for emotionally supporting me and cheering me on through the tough days, realigning my focus and instilling faith.

I would then like to express deep thanks to my supervisor, Prof Dr Fadi Al Turjman, for his unconditional guidance and supercharged motivation throughout this study. I have learnt a lot from him, and I plan to use all this new knowledge to propel me to new heights.

To my closest confidantes and colleagues, Mahmoud Abubakar, Hadjer Benyamina and Abdulmunaim Saleh. Thank you so much for your support, willingness to help me out through the tough times of this research and moral support. I couldn’t have gone through the days of this research without the three of you.

Finally to my brother, Yasir Ame. Thank you for the words of confidence, reassurance and hope. Thank you for checking in on my progress throughout this whole study.

Ibrahim Ame

Abstract

LEVERAGING ARTIFICIAL INTELLIGENCE FOR TICK TAXONOMY: EXPLORING DEEP LEARNING MODELS FOR PARASITE CLASSIFICATION

Ame, Ibrahim
MS.c, Department of Artificial Intelligence Engineering
June, 2024, 76 pages

Bugs are everywhere and they are a vital part of the ecosystem, helping in keeping things in balance. However, some bugs are more parasitic than other, feeding off other animals in order to survive. Ticks are parasites that are tiny arachnids that feed off dogs, cattle and sometimes humans. They can sometimes leave you with a small bite, paralyse you, or worse even kill you with strong venom. Detecting if the bug is dangerous required so form of expertise, mostly a scientists, a vet or a farmer that has experience in identifying these bugs for a very long time. This study uses Artificial Intelligence to hone in all the skills from scientists in order to predict if the bug is dangerous or not by classifying one of the three available classifiers, Hyalomma, Rhipicephalus and Unidentified. On top of that, separate models will be able to predict a bug from the position it is taken from either it's head, center belly portion or a corner portion. Furthermore, an application will assist scientists, farmers or even normal people who own animals that can easily be infected with ticks to easily evaluate photos of various to predict if they fall in one of the three types. The implementation of the proposed application is built with a simple UI to be able to make fast predictions from various Deep Learning models trained with high accuracies of a VGG16 model reaching 99.57%, ResNet50 model with a 98.98% and a custom basic CNN model with 95.71%. This will help keep animals and people safe, identify dangerous ticks bugs and eventually save lives.

Key Words: Tick, Hyalomma, Rhipicephalus, Deep Learning, Computer Vision

Table of Contents

List Of Tables

List Of Figures

List Of Abbreviations

AI:	**Artificial Intelligence**
App:	**Application**
DL:	**Deep Learning**
CNN:	**Convolutional Neural Network**
VGG:	**Visual Geometry Group**
ResNet:	**Residual Network**
ML:	**Machine Learning**
DOM:	**Document Object Model**
API:	**Application Programming Interface**
RNN:	**Recurrent Neural Network**
CPU:	**Central Processing Unit**
GPU:	**Graphics Processing Unit**
WSGI:	**Web Server Gateway Interface**
GD:	**Gradient Descent**
SGD:	**Stochastic Gradient Descent**
Adam:	**Adaptive Moment Estimation**
GAN:	**Generative Adversarial Network**
FTP:	**File Transfer Protocol**
VAR:	**Variant**
RAM:	**Random Access Memory**
HTML:	**HyperText Markup Language**
PHP:	**Hyper PreProcessor**
SQL:	**Structured Query Language**
UI:	**User Interface**
CLI:	**Command Line Interface**
URL:	**Uniform Resource Locator**

CHAPTER I

Introduction

Pandemics have impacted societies all around the world dramatically throughout history. The earliest recorded pandemic occurred during the Peloponnesian War and caused two-thirds of the population to die from this disease, which included symptoms like fever, thirst, bloody throat, tongue, red skin, and lesions after the disease passed through Libya, Ethiopia, and Egypt reaching to Athens, Greece 430 B.C.

Humanity faced since then several recorded pandemic outbreaks such as the Antonine plague, Cyprian plague, Justinian plague, and more diseases that killed over 26% of the world population in less than 4 centuries. It is believed to be the first significant appearance of the bubonic plague, which features enlarged lymphatic glands and is carried by rats and spread by fleas. Known as the Black Death, killed one to two-thirds of Europe's population as it spread over the continent. Acute fever and blood in the vomit were common symptoms, which were followed by painful buboes (swollen lymph nodes) in the armpits, neck, and groin. With a 60 to 90% fatality rate, victims often passed away two to seven days after infection.

The bubonic plague, often known as the Black Death, was a catastrophic pandemic that changed society and economies throughout Europe and Asia. It is recorded to have killed a considerable proportion of the global population at the time of the attack. Trade routes and population movement aided the disease's fast spread, causing enormous social and economic disruption. The epidemic had long-term implications on ancient regimes and alterations in dynamics of power.

A slow-developing bacterial infection called Leprosy, wrecked humanity once more in the 11th century. It is known as Hansen's disease, and it continues to infect several numbers of individuals each year resulting in sores and deformities, with devastating consequences if not treated with antibiotics.

There were disastrous epidemics worldwide between 1492 and 1875. Native Americans had high death rates as a result of European settlers bringing diseases like smallpox and measles to America. Both the cholera pandemic and the Great Plague of London had equally devastating effects on individuals and society in the United Kingdom.

In 1897, a French - Swiss bacteriologist discovered Yersinia pestis, a disease-causing bacterium and the route of transmission was identified by later studies as fleas, particularly the oriental rat flea that carry bacteria from diseased hosts.

There have been challenges throughout history, and they continue to this day with influenza pandemics and infectious diseases' impact on humanity. From the Russian Flu of 1889, which claimed 360,000 lives, to the devastating Spanish Flu of 1918, which killed 50 million people, to the Asian Flu of 1957, which killed 1.1 million individuals.

In 2003, another outbreak of Severe Acute Respiratory Syndrome (SARS) threatened the public health, as it spread quickly and lead to a high mortality rate which originated in China. The containment of SARS demonstrated the efficiency of worldwide quarantine procedures. Furthermore, it provided as a valuable lesson for handling following outbreaks, underlining the importance of being prepared and taking quick action when new infectious illnesses strike in unconscious and unintentional preparations for another life-threatening pandemic, COVID 19.

The first cases of infectious respiratory disease of coronavirus were reported in China, again, around the end of 2019. That time, The World Health Organization (WHO) classified the coronavirus epidemic as a global pandemic three months after it began to spread quickly to other countries, and by the end of January 2020, they had issued a Public Health Emergency of International Concern (PHEIC).

The world has not taken a break from all the chaos caused by COVID-19 and the quarantine policy that imposed isolation and cancelled flights before another outbreak saw the light once again. According to France's Union Chamber of Insect Control, the number of pest control visits surged by 65% in 2023 after bed bug infestations in Paris, particularly, occurred just before Paris Fashion Week in October of the previous year and the upcoming Summer Olympics in 2024, which are predicted to draw 1.5 million spectators. After gathering information and conducting the required research, the French health and safety organization, ANSES, discovered that about 10% of French families had reported bedbug infestations in just the years 2017 to 2022. This indicates that the problem was not new; rather, it was just not effectively addressed. As a survival mechanism of the poisons employed to eradicate them, the bedbugs evolved and acquired defences that made detecting them early a near-impossible mission.

With their astounding diversity, more than 1.5 million species, insects rule the world. They are integral to ecosystems, occupying almost every ecological niche, from nutrient cycling to seed dispersal. Beneficially, they pollinate plants, produce useful substances, and control pest insects. Scientifically, they contribute to genetics research, population biology, and biodiversity studies.

Insects may also be used as environmental indicators to determine soil pollution and water quality. However, parasites, bacteria, fungi, bugs, or protozoa, live on or within other living things. They take nutrients straight from their hosts and give nothing back in exchange.

Understanding their life cycles is crucial for effective treatment and control. Some parasites have complex life cycles involving vectors, which are some species that transmit the parasite to intermediate hosts. The relationship between insects, parasites, ticks, and public health underscores the need for comprehensive approaches to addressing their impact. In recent years, due to climate change, there has been a rise of ticks (Nuttall, 2022) and tick borne infections. The changes in temperature can exercabate the growth and lifespan of ticks from a few months and years to decades.

In short, ticks and mites belong to the order Parasitiformes. Ticks are members of the Ixodida suborder, whereas mites are further classified into many superfamilies, such as Sarcoptiformes, which contain scabies mites, and Trombidiformes, which include chiggers.

There are two primary families among the parasitiformes:

- Soft ticks (Argasidae) don't have a scutum like hard ticks do. They lay many batches of eggs, eat sporadically, and finish their life cycles in the nest or house of their host. Ornithodorinae is one of the family's genera.

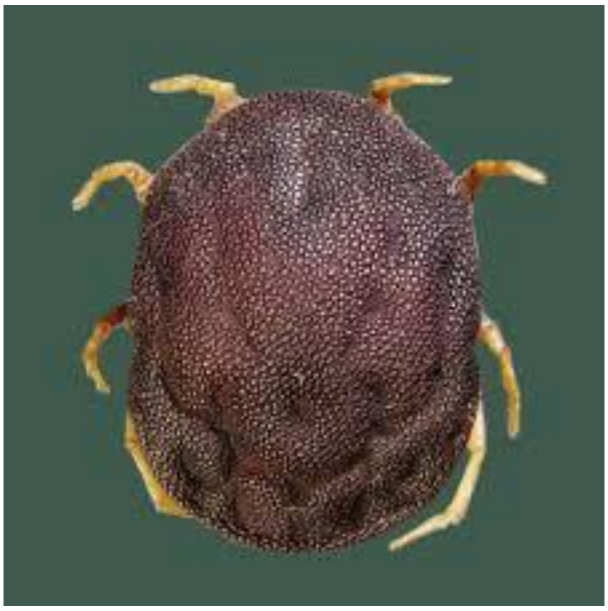


Figure 1: An image of a soft tick, Argasidae

- Hard ticks (Ixodidae): The scutum is a hard shield that these ticks possess shown in Figure 2. The hard tick family has more than 700 species, including Hyalomma viewable on Figure 3, and Rhipicephalus seen on Figure 4.

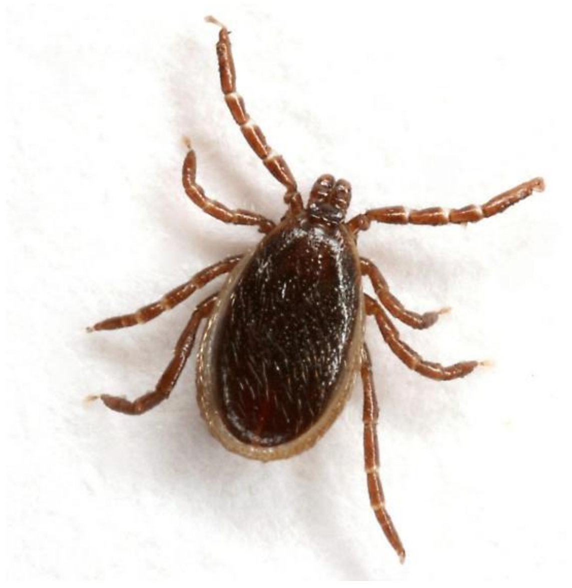


Figure 2: An image of a hard tick, Ixodes or Ixodidae

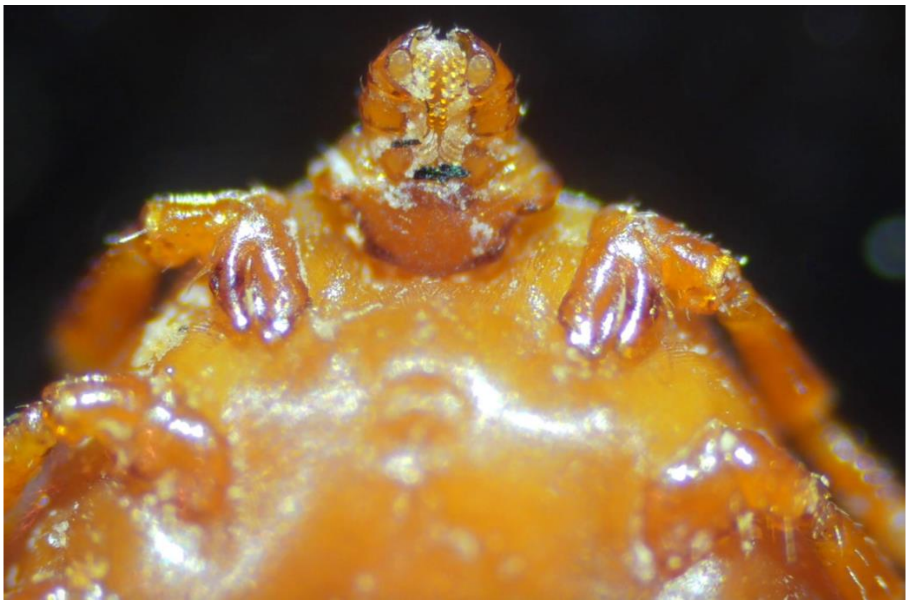


Figure 3: A microscopic view of the Hyalomma Tick Bug

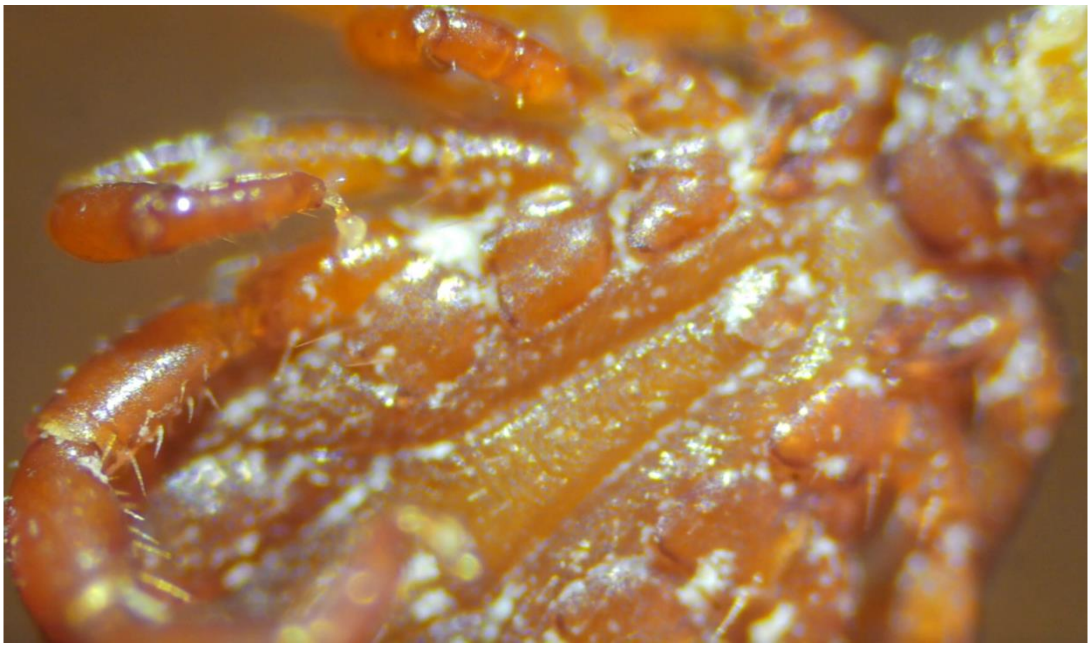


Figure 4: A microscopic shot of the Rhipicephalus Tick

Every organism is entangled in a complicated web of relationships. While most of them are harmless, not all of them are. Scientific research invests significant resources in classifying and eliminating external dangers from microorganisms to insects and ticks. Ticks' interactions with hosts, pathogens, and the environment shape disease transmission dynamics, which impose several implications for human and animal health, including the transmission of serious diseases leading to long-term illnesses, paralysis, disability, and even death.

These ticks are found in many different parts of the world, including Africa, Asia, Europe, and the Americas. They are well-known for carrying several illnesses, including tick-borne encephalitis, Rocky Mountain spotted fever, and Lyme disease.

Therefore, our research mainly focuses on exploring the taxonomy of hard ticks in particular and parasites, in general, using artificial intelligence techniques such as classifying ticks by type.

Since 1956, AI has evolved significantly, leading to ground-breaking research and applications across various domains, including a fascinating area of research where AI intersects with the insect and bug worlds.

AI depends on employing intelligent models to process enormous amounts of data. One of the fathers of artificial intelligence, Marvin Minsky, describes AI as the science of enabling machines to perform tasks that would require human intelligence. Alternatively, Alan Turing proposed (Christensen, 2015) that a machine might be deemed intelligent if it can emulate human reactions in particular scenarios.

The primary goal of artificial intelligence is to enable machines to perform activities requiring intelligence more quickly and efficiently than humans by using a learning process. The learning curve flattens out with increasing data availability and system complexity.

Evolved by refining previous ones to obtain more precise outcomes, it was believed by scientists, particularly neurologists, that machines could be trained similarly to answer various mathematical and statistical issues that conventional approaches were still unable to resolve. This belief stems from biology and the way the brain learns from its surroundings. That marked the start of the next chapter in machine learning (ML).

Machine Learning (ML) is a branch of artificial intelligence that uses detection and classification algorithms to automatically find patterns in data sets. These algorithms are then fed with recurring patterns from any kind of digitally stored data, including numerical values, images, videos, figures, and so on.

ML generates rules on its own, depending on input and the desired outcomes. The three fundamental machine learning paradigms are reinforcement learning, supervised learning, and unsupervised learning. Each is designed for a certain kind of learning task.

In general, machine learning models are trained on small amounts of structured data, low-end hardware, and simple processing procedures. In contrast, large amounts of unstructured data require sophisticated and high-performance algorithms beyond what we have been exposed to thus far for classification and more complex processing procedures. To put it another way, Deep Learning (DL) joins the game fully equipped with the capacity to learn from massive datasets, unstructured data, and large-scale problem resolution. But the most significant feature of DL that sets it apart from ML is that, because of the DL algorithm's inherent ability to recognize patterns, feature engineering is carried out implicitly and automatically without the assistance of a data scientist.

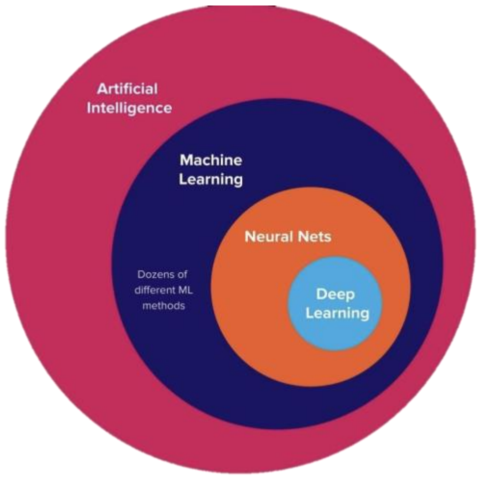


Figure 5: The Subsetting definition of Artificial Intelligence, Machine Learning, Neural Networks and Deep Learning Terms.

A computer science method known as Artificial Neural Networks (ANNs) was developed to replicate bio neurons in a mathematical model based on scientific study. Neural Networks is another name for ANNs. The first interest ignited in 1943 following the publication of research by neuroscientist Warren S. McCulloch and logician Walter Pitts. The two scientists attempted to simplify the analogy of the human brain through the use of a highly simplified mathematical model of neurons known as MCP Neurons, or Threshold Logic Units, which connect the units of an artificial neural network. The brain is built in precisely that manner (Floréen et al., 2010).

Neural networks are designed to perform cognitive tasks like machine learning and problem-solving. The volume of research data is increasing constantly, and sophisticated analytic techniques are required to uncover hidden data or information that might be attributes or entries for learning algorithms. The artificial neural network (ANN) is one of the numerous methods available to meet the requirements of scientific research. Artificial neural networks (ANNs) are a more dependable, faster processing method, and more scalable option for modelling complex non-linear interactions than standard regression techniques. Learning techniques using Artificial Neural Networks (ANN) are modelled after the structure of the human brain. Similar to how the human brain is trained, neural networks are also trained by identifying patterns through experimentation.

It took decades of intense effort to teach a machine to recognize images before researchers were eventually able tosuccessfully imitate the visual recognition system found in the human brain. Lawrence Roberts (Roberts, 1965) first suggested the concept of extracting 3D geometrical information from a 2D perspective. This was the first significant development in the field of computer vision. Along the way, several models and algorithms were proposed, including CNNs or ConvNets, a visual structure model, and the generalised cylinder model.

ConvNets and conventional neural networks have similarities in many ways. Both of these are neural structures that rely on weights that have been learned from data.Three dimensions are possible for the configuration of neurons in a CNN: width, height, and depth. Each layer transforms the three-dimensional input into a three-dimensional neural output utilizing an activation function. In summary, CNNs are spatially shared deep neural networks with shared parameters. Additionally, CNNs accept matrices as input in a different way than multi-layer perceptrons, which only accept vectors and ensure that the image's spatial structure is preservedthroughout.

CNNs are the best options when handling computer vision issues because of these qualities, among others. This chapter has covered the history of AI as well as how to use ML approaches to enhance AI models and produce remarkable outcomes. Several over 90 studies were conducted between 2016 and 2022 on insect detection in traps using deep learning focused on insect classification, counting, and detection (Teixeira et al., 2023). Researchers have explored automatic insect detection using machine learning and deep learning techniques and algorithms namely SVMs, Convolutional Neural Networks, and Generative Adversarial Networks.

To create bio-inspired robotics that emulate the behaviour of insects, scientists study insects. These tiny robots, like insects, possess sufficient computing capability to do various tasks. For example, scientists have developed flying robots that are modelled after ants and bees. These robots are capable of navigating their surroundings, dodging hazards, and changing with the times. Insect brains are remarkably neuronally dense for such little bodies. Utilizing this, researchers have created effectivealgorithms that use a lot less processing power than deep learning systems by reverse-engineering insect brains. Notably, a portion of the honeybee brain was successfully replicated in silicon to produce a fully autonomous drone with obstacle-avoidance capabilities.

Likewise, we consider standard approaches that implement methodologies proposed by other authors, such as the VGG, ResNet, AlexNet, Inception, and GoogLeNet architectures, to build an effective deep learning model capable of classifying a multiclass dataset of ticks micro-images that were captured through video to image sequence conversion using video to image conversion software. As you will see, we employ state-of-the-art tech stacks, highlighted by many research articles where CNNs performance produced noteworthy outcomes in a variety of industries and fields.

CHAPTER II

Literature Review

2.1 Background

Identifying insects is extremely challenging, especially when using regular eyesight. This has prompted an increase in image-based systems by implementing deep learning models in identifying those insects. There have been some experiments employing deep learning algorithms to classify the various varieties of insects observed in nature, as evidenced by research (Amarathunga et al., 2021). He reviewed 69 image-based insect identification studies published between 2010 and 2020. Their findings indicated a trend in the use of deep learning techniques in insect identification. (Gill et al., 2023) created a learning-based picture classification MobileNetV3 model to distinguish different insects with model's accuracy as 80%. Artificial Neural Network (ANN), Support Vector Machine (SVM), K-nearest Neighbor (KNN), Naïve Bayes and a proposed Convolutional Neural Networks were compared by (Kasinathan et al., 2021) in detecting and classifying insects in field crops. Other studies include the use of VGG, Inception, Xception, MobileNet, DenseNet, ResNet and EfficientNet (Malik et al., 2023), Bayesian learning (Badirli et al., 2023) and Random Tress (Yang et al., 2010) in identifying and classifying different insect species.

2.2 Related work

As much as most studies focused on insects, there have been a few studies which specifically focus on ticks. Ticks are a nuisance to humans and animals as they serve as vectors of diseases. It is critical to recognize and categorize the many tick species. Previously, (Estrada-Peña, 1999) used state images for remote sensing from the National Oceanographic and Atmospheric Administration (NOAA) to estimate the link between vegetation factors, temperature, and the distribution of the cattle tick, Boophilus microplus. To extrapolate the results, a cokriging mechanism was designed. The habitat suitability data obtained from two vegetation and four temperature variables were strongly related to the known distribution of B. microplus, with a sensitivity of 0.91 and specificity of 0.88, allowing for a good prediction of tick habitat appropriateness. As technology has advanced in recent years, (Pérez-Otáñez et al., 2024) carried out research on the tick affecting cattle in Ecuador. The authors used Random Forest (RF) models to determine the distribution of Rhipicephalus microplus and Amblyomma cejannense sensu lato. The data included in their model came from 2895 farms, where ticks were gathered during animal inspections. Furthermore, vapor pressure and bioclimatic data were collected as overlays to create predicted maps for each species. The results showed that both Rhipicephalus and Amblyomma predictions were highly accurate, with values of 0.97 and 0.98, and sensitivity of 0.96 and 0.93, respectively. The scientists discovered that increasing levels of precipitation increased Rhipicephalus presence in cattle while decreasing Amblyomma. Another study (Omodior et al., 2021) compared the accuracy of deep learning models to a custom-built Convolutional Neural Network (CNN) model for classifying common hard ticks. A dataset of over 2000 pictures from four tick species was used. The researchers trained two distinct Convolutional Neural Network (CNN) models on tick images: ResNet-50 and a simple custom-built model. They tested the models' performance using a separate set of tick pictures that were not used in the training phase. The shallow custom-built model outperformed the ResNet-50 model in terms of training and validation accuracy, achieving 99.7% and 99.1%, respectively. When tested with new tick picture data, the shallow custom-built model enhanced mean prediction accuracy to (80%), increased confidence in genuine identification (88.7%), and reduced mean response time (3.64 seconds). Despite the limited training dataset, these results suggested that the custom-built CNN model had substantial potential for categorizing common hard ticks.

(Justen et al., 2021) devised a way for identifying ticks using a smartphone. The system was linked to a CNN model known as "TickIDNet". The model was trained using a dataset of 12000 photos representing the three most frequent tick species. With augmentation, the authors were able to expand the dataset to 90000 photos. The model scored an accuracy of 87.8% across all the species. Even though the model performed well, it failed to match the expert’s performance. In another research, (Akbarian et al., 2020, 2022) built a detection tool used to differentiate blacklegged ticks from others using CNN model with their model achieving an accuracy of 92%. Moreover, (Akbarian et al., 2022; Luo et al., 2022) developed a computer vision system that uses a deep learning model to aid with tick species identification. Their findings demonstrated that the computer vision model could recognize ticks with 99.5% accuracy.

Table 1: Studies made in classifying different types of ticks using deep learning models especially the CNN model. There is significantly less work done in this domain as it is still developing research.

Variable Predicted	Models	Accuracy	Reference
Vegetation variables, temperature, and distribution of Boophilus microplus	State imagery, cokriging	Sensitivity: 0.91, Specificity: 0.88	Estrada-Peña, 1999
Distribution of Rhipicephalus microplus and Amblyomma cejannense sensu lato	Random Forest (RF)	Accuracy: 0.97 (Rhipicephalus), 0.98 (Amblyomma); Sensitivity: 0.96 (Rhipicephalus), 0.93 (Amblyomma)	Pérez-Otáñez et al., 2024
Classification of common hard ticks	ResNet-50, custom-built CNN	Training accuracy: 99.7%, Validation accuracy: 99.1%; Mean prediction accuracy: 80%, Confidence in detection: 88.7%, Response time: 3.64 seconds	Omodior et al., 2021
Identification of ticks through a smartphone	TickIDNet (CNN Model)	Accuracy: 87.8%	Justen et al., 2021
Differentiation of blacklegged ticks	CNN Model	Accuracy: 92%	Akbarian et al., 2020
Identification of tick species	CNN Model	Accuracy: 99.5 %	Luo et al., 2022

CHAPTER III

Methodology

3.1 Introduction

This research utilises a myriad of Machine Learning methods (Mahesh, 2020) and techniques to be able to achieve the desired results and accuracies. It has employed some core Machine Learning tools but all this research is done using Deep Learning models to boost the metrics required to predict the correct tick from an image. By doing so, we can leverage some powerful learning techniques on the images to get better insights and classify them correctly.

Some of the advantages of using Deep Learning models alone instead of the base Machine Learning models are that we can perform multi-class classification a lot easier (Lahijany et al., 2021). By passing arrays of classification tags, we can attach folders to those tags and be able to state that they belong in one group and the rest in other groups. Secondly, we can extract new features from the images that we could not easily perform if we were trying to hand-engineer the features one at a time. Even though some literature confidently states that hand engineering features are better. Some images are a lot harder to extract features from due to their variability. Tick bugs seem to come in extremely different shades of brown, red and dry gold. On top of that, these colours may not be a differentiator between the bug classes. The shapes, sizes and patterns of the bugs may also be shared across the species. The choice to use Deep Learning models was to be able to extract features automatically through the use of convolutional layers.

The biggest disadvantage to using Deep Learning models is that it is impossible at the current time of writing this research, to see what the model is trying to do between its middle layers. You only know what its input is and its final output. During training, a lot of educated guesswork is required to visualise what the model is learning and how improvement can be made.

To use a Deep Learning model, a lot of data is required. All of this data is fed into the network either one by one or in batches depending on the speed and power of the resources available on the training machine. The input of the model, referred to as the input feature of the model is passed at the input layer of the model. These input features go through hidden layers in the middle of the model to extract information about the image and then a final result is given at the output layer of the model.

The model input features for all the models built for this paper were unstructured data in the form of images that weretaken under a microscope. The image entailed only 1 tick bug per view to ensure experiment integrity. The image was shot in clear 4k quality and then exported from the microscope. The total count of all the images used in this study was around 25,800 including images 6,000 images that were not from the laboratory. Even though videos were supplied, this project required smaller data to be able to go through the model networks with ease and not consume a lot of resources. One large limitation was experimenting with all these networks on a personal computer before moving the code online to servers for training. Even with these small-sized images, loading the large network structures and performing convolutions and multiplications on the large matrices sometimes crashed training on the server. It was a personal choice for this study to choose simple methods and still data like images to save resources and to be able to capture the intricate details of these tiny parasites.

Hyalomma and Rhipicephalus bugs are only viewable properly under a microscopic lens, making it difficult to take high-quality shots of them using a phone camera. Thus, the team at Desam Laboratory, which is where this data was collected from, had to first acquire these bugs from stray dogs. The team had a sample count of around 10 - 20 different bugs that were then analysed and photographed. Because of the nature of the project and the choice in algorithms, the data had to then be augmented to reach the sample count of 19,345 usable tick bug images. This augmentation was done manually, where the images were cropped, rotated and or flipped to create new data. This process was done by recording a video of the bug under different zooms, and rotation with movement from one spot of the bug to another.

The video was then exported as image screenshots of each frame of the bug. Under some of the other models mentioned, we employ a different strategy to combat some of this data augmentation to test out if it has a positive effect on the learning and accuracy of the models. A summary workflow on this data was data collection and exploration, data cleaning, and data restructuring followed by data training.

3.2 Data Collection and Exploration

The data was acquired from the lab in 2 batches. The first batch had around 12,000 photos and then the second around 7,000. This gave an imbalance of images of Hyalomma and Rhipicephalus. There were double the amount of Rhipicephalus bugs in the dataset than Hyalomma. However in the results section, it will be seen that we managed to ensure this did not cause the models to overfit. The first batch was used to train the first round of models to experiment if it was feasible to achieve any form of result. The files themselves are in the form of JPEG files with high resolutions measuring 1920×1080 pixels and around 0.1MB - 1MB each. Making them perfect for training as we have no quality loss from the images. Images with high quality are great for extracting features, especially when dealing with small tick bugs that can only be seen properly under a microscope.

From the images given, some images were used as a control to feed into our models for the unidentified class. It was also required that the model should predict unidentified for any other objects that were not tick bugs. To achieve that 6000 images of common objects were fed into the models to identify objects that were not ticks bugs. This choice of keeping common objects is something unusual, however, it was done as these bugs are usually spotted around vegetation, common household items and in areas where there are humans. We wanted to makes sure that the common objects and or people would not be classified as a tick bug. These images were taken from the coco dataset. We also had images of spiders, that were the closest species of bugs to ticks. These images were only placed in the test set of the dataset. This was to be able to predict if the spiders would come up as unidentified instead of one of the two groups of ticks.

After accessing the data, and looking through the images, it was clear that the images took different forms in the sense that, bugs had variating features, their shapes and sizes were vastly inconsistent and their colours varied within the same species. Some images showed a bug in its full view, some of them in half the view and the rest only a corner piece of the bug was shown. Under data restructuring, we employ methods and techniques to segregate these images to reduce bias within the models.

Out of all the images, 4779 were Hyalomma and the remaining 11,905 were Rhipicephalus. The varying difference was a bit of a concern at first, but it seemed not to have too much of an effect on the overall accuracies after training as once the data was split, the margin percentages weren't too high. Figure 6 showcases the overall image counts of Hyalomma, Rhipicephalus and Unidentified images.

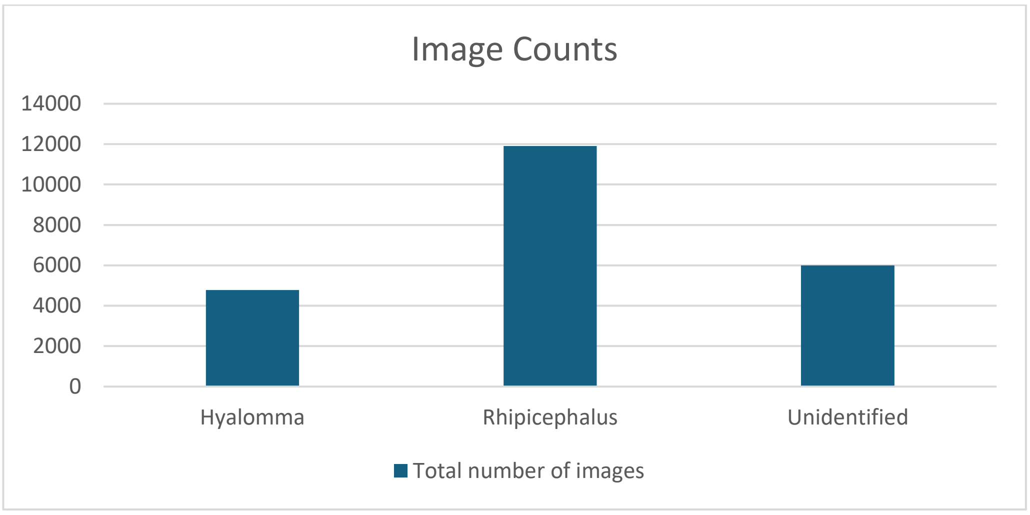


Figure 6: Total number of images for the dataset.

3.3 Data Cleaning

The primary data was collected from a lab and was intended for this project. However, the images had to be checked on by one to ensure that there did exist one bug for each image. Some images did show just black shots of emptiness which were then removed. See Figure 7.

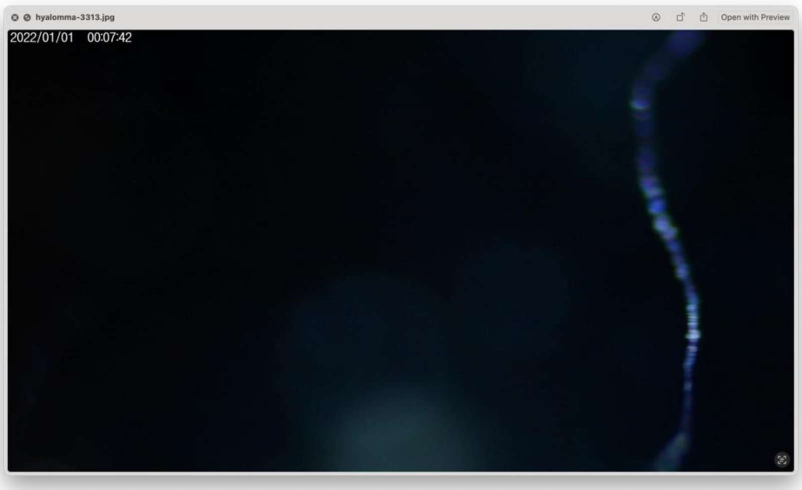


Figure 7: A microscopic shot of nothingness that was found in the dataset.

Removing images like this helps in keeping the model safe from extreme outliers that can cause false predictions. A dark image without a bug is more of an unidentified classification than a tick bug. Even though the choice of moving the image to unidentified was possible. The best option was to remove it altogether. Various pixels in the image could cause the weight in the model to shift the path of gradient descent in a different direction.

3.4 Data Analysis

A look through the images themselves revealed a lot of interesting information about tick bugs and the shapes they took on from the different species. Hyalomma bugs were more shiny, glistening with a sort of oil and fingerprint-like patterns on the centre belly portions of the parasite. They were a bit more larger, more round and had gold cream-like colours. On the other hand, Rhipicephalus bugs were more stern looking with sharper edges, defined shapes, skinnier bodies and strong red to brown colours. They took the form of a shrinking bell shape that resembled a leaf, with a pointy edge at the head of the bug, and a round bell shape at the bottom of the bug with a distant belly button dot at the centre bottom of the parasite. All bugs seemed to be kept in a petri dish, with a liquid-like substance keeping the bug afloat. The bugs were presumed dead before photographing. Some shots of the bugs have legs spread out, and some have legs curled into the centre of the body, which gives us a diverse set of possible scenarios of when the bug could be predicted.

All bugs had 8 legs, a body and a head. However, not each photo had all these elements present. Most photos had some portions, some had zoomed portions of just one of these elements, and some photos had all the elements present. Thispresented a very interesting opportunity to further segment the photos into subclasses, where portions of the bugs were grouped. It was a common pattern for most images to either have a headshot, a body shot or a shot of the bug in the corner. These 3 subclasses can be viewed in Figure 8, showcasing the different views of the parasites.

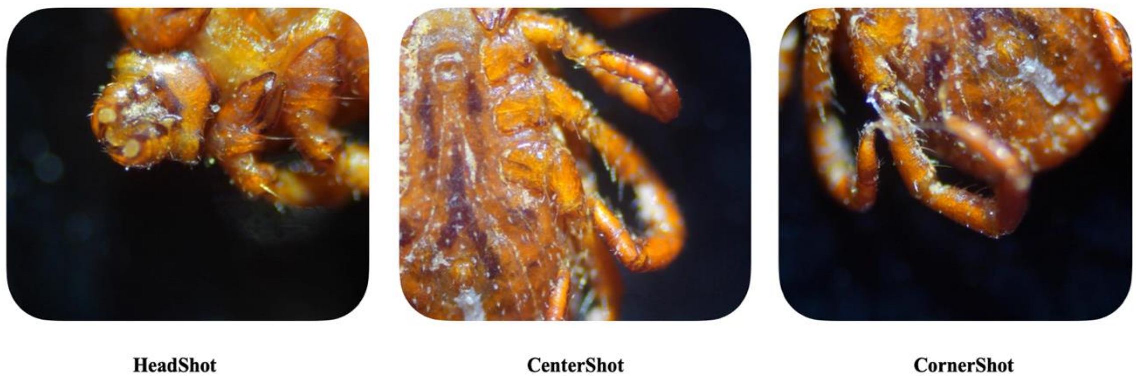


Figure 8:Further subclassing of images. Left is Headshot, Center is CenterShot and right is CornerShot.

By resorting the photos into new subcategories, we were able to form a new dataset that could take the research on a different path to be able to improve the accuracy and perhaps reduce overfitting and bias from the predictions.

3.5 Data Restructuring

The data was restructured into different folders to provide 2 different approaches for finding an optimal solution to classifying the bugs. Firstly, all the images were renamed from their respective folders from their original names that came under a microscope to a more comprehensible name. The format was the classifier name with a dash and then a consecutively iterated number preceding it.

We had 2 main data set restructures. The first one which will be referred to as A, data A or dataset A throughout the paperis the same structure as the original data with only a renaming of the images, removing some unworthy images and shuffling them. The second dataset which will be referred to as B, data B or dataset B is a clone from dataset A, that was then subclassed further to produce 3 new subclasses.

Dataset B is an experimental solution to see if it produces higher accuracies and reduces bias. What it does is have the images be in the form of what shape the bug appears in the image as can be seen in Figure 8. We have some images in the folders that are just heads of the bugs which are denoted as HeadShots. Some images contain the centre portion of the bug which is the belly and sometimes the legs can also be observed. These images were housed under CenterShots. Finally, the last subclass was shots that weren't fully heads or centres of the bugs, there were edges, legs, and corners of the bugs. These were placed under CornerShots. Thus the final subclasses were multiplied by 2 for the Hyalomma and Rhipicephalus classes and can be viewed under Table 1 . The unidentified images were not subclassed as they comprised many random objects. The bugs in that folder were left as is.

Table 2: List of major classes of tick bugs with their corresponding subclasses that were created after reviewing the images.

Classification	Subclass
Hyalomma	HeadShotHyalomma
	CenterShotHyalomma
	CornerShotHyalomma
Rhipicephalus	HeadShotRhipicephalus
	CenterShotRhipicephalus
	CornerShotRhipicephalus
Unidentified	---

The approach with dataset B is to try and reduce the data augmentation that has been done to the images. The images were cropped in so many locations that they produced varying degrees of bug shots. This can be an extreme disadvantage if the augmentation has been done over such a small amount of images. However, perhaps the zooming in of these images into these locations can produce these new subclasses that can later be used to classify parts of a bug that might be even more zoomed in or blurry. The worrying aspect of this approach is that from the images that we already have, we are reducing the overall size of the images per class. Now the 2 major classes have been cut approximately by 1/3 into smaller subclasses.

3.6 Data Splitting

Various splits were considered for this project. Notably, the 80/20 split was first applied to the unstructured data. This is where we split the data with an 80% split for its training set and 20% for its testing set. After multiple iterations of this split, it was seen that the models kept overfitting to the training data, which is where it was performing well on the training set but not well on the test images that it was given at the end.

A new split was devised to resolve this problem, a push further into the training set revealed a 60/40 split. By giving the testing set more images to test on, the model improved its accuracy on both the training set as well as the testing set. However, now the model was overfitting to both sets and was sometimes biased to either the training set or the testing set depending on the epoch count of the model being trained on.

A final split was decided upon by splitting the data 3 ways, a 50/40/10 dataset split. The first split of the data contained 50% of the images was the training set. The second split was a new set called the dev set which was a testing set only used within training. The final split was a test set that was used to test outside the training environment. This enabled the model to reduce overfitting neither on the training nor the dev test set. The test set could also be used to accurately see which images the model was failing on.

3.7 Model Training

Training models include having an environment to be able to pass data efficiently, and utilising as much memory through the model so that it can train quickly. What we are trying to achieve is to create weights within the artificial neural network and biases that match the data in a way that when new data is passed in and matches those weights, it will be guided towards the correct classification. When images of the same type tune weights to certain values, it makes the model more robust to the shapes and colours of the new bug images. Training requires a model, data and a lot of time. All models in this study were babysat. They were watched as each epoch passed through to notice if they were overfitting. A correct model train will track every loss decrease and accuracy increase after each epoch. If the accuracy and loss start bouncing up and down between iterations, it is said that gradient descent is regressing, and cannot find a new minima for the cost function. Thus the model will either start overfitting or completely not work at all and make wrong predictions.

Gradient descent is a mathematical approach to minimising any function. What it does is take a function and find the local minimums of that function. The objective goal of learning however is to find the global minimum of the function to make learning as efficient as possible and have a very low loss. The equation below is the mathematical definition of GD and is crucial in learning Deep Learning models. Various optimisers are used in training to make Gradient Descent faster. For this study, SGD and Adam were used interchangeably between model training for experimental purposes. However, after seeing that the Adam optimizer performed better and faster, it became the default for all models.

Equation 1: Stochastic Gradient Descent and Adam Averaging Summations From Sums of the Loss Function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(y_i, f(x_i;\theta))$$

To train these models, code about setting up the environment and loading data as well as training the model was split into cells and running them independently within a Jupyter file. Jupyter files can handle chunks of code in a cell one at a time,so that if one cell crashes, data from the previous cells is retained. This helps in reducing the time to run previous cells again. The efficiency of the Jupyter kernels helped us train multiple models on a more powerful machine.

The model architectures in this study were of 3 types, a shallow network consisting of only 5 valid layers, a middle 16-layer VGG16 network as well and a large Resnet50 with 50 layers that had skip connections. Skip connections allowed the model to train one layer and then skip a couple of layers, and then reintroduce previous layers in future untrained layers. These network architectures would consume a lot of resources and would make it almost impossible to train on a personal computer. Testing and experimenting on a personal computer was possible with very small image sizes between 16 and 32. However, this is not enough in production prediction. Images of these sizes tend to be very blurry and carry little to not enough data about the intricate details of parasites. To combat this and use large model architectures as well as large image sizes, it was required that we train the models on a server that had powerful resources. Training for the all model was only tested over 4 input feature size sets. These input feature sets were:

- 32 x 32 -- base model
- 64 x 64
- 128 x 128
- 256 x 256

However, for the final choice, only the 128x128, was the one used for all models. It gave the most optimum size for accuracy ratio. This also gave the models a fair comparison between each other.

Models were trained on a combination of GPU and CPU, depending on what was free on the server, as it was training them in parallel. This helped reduce the time it took to train the models from days to just minutes. The server that was used was from Paperspace, and it greatly reduced the amount of training time. The server was a Linux machine running Ubuntu 22.04 with dual 16GB GPU cores, 16 CPU cores and 96.6 GB RAM. The 2 cores were run separately to help train2 models in parallel over 2 different Jupyter files under the same kernel. This approach of not using a PC to train utilises the resources of servers without all the bloat software already running on a personal computer. When the 32x32 model was first trained on a personal computer for experimental purposes, it was consuming all the RAM of the computer and took over 7 hours to train. However, after moving to the server, it only took about 20 minutes to reach the same accuracy and loss. After booting the server, Transmit (an FTP application) was used to move the data sets, A and B and Jupyter scripts to the server. All training, validation and testing happened on the server to not have any biases within the models.

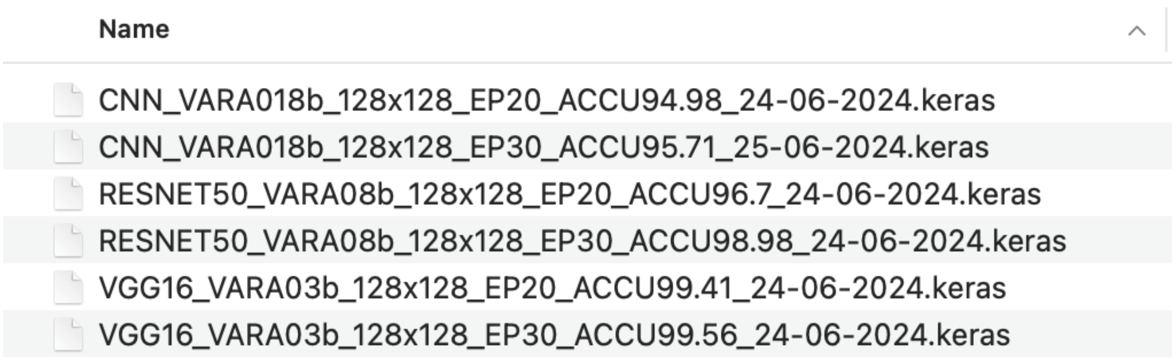
Models were trained using 2 epoch numbers. The first was 20 epochs and then 30 epochs. Some architectures responded better to less epochs and produced better results. The rest required a bit more training and performed better with 30 epochs.

After the models were trained, we utilised Transmit to fetch all the models as well as the metrics from the server back to the personal workspace and then tested over the web UI locally. Models were built using the TensorFlow framework. This is a powerful artificial intelligence codebase built by Google for AI engineers to utilise in building models. Using its sequential API, we can attach the model with specific layers one by one in order. The model can then be reviewed with the summary function to showcase the layers and their respective input and output parameters.

Training all these models will help compare if the input size matters in producing higher accuracies, as that is our optimising metric for this paper. The aim is to reach an accuracy higher than 95% for at least 3 models with a low error rate of under 5 per cent and a loss metric of 0.1 or lower. At model training and writing of this paper, the human error rate was estimated at around 5%, giving us a wide window to narrow into the Bayes error rate.

In statistical classification, Baye's error rate is the lowest available error rate that any classifier can reach. This error rate can vary for different classifications, but it is known that no classifier should be able to reach Bayes error rate. The error rate is usually compared to human abilities to predict and classify problems. Finding the human error rate can help determine how close Baye's error rate is.

Models are named with a very verbose structure as can be seen in Figure 9. This naming scheme helps in identifying models pretty quickly and choosing the one with the required accuracy. Metadata like dates and epoch runs are also saved onto the name to keep some history of the model for future comparisons to other versions if the model gets improved. This model had an accuracy of 99%. All model accuracies were written to the closest significant number. Details of exact accuracies can be viewed under the results section for both dataset A and B. Finally, the training date is at the end of the file name, to keep track of when the model was trained. All model files were saved with a .keras extension, supporting the latest 3rd version of Keras as of 2024.



Name
CNN_VARA018b_128x128_EP20_ACCU94.98_24-06-2024.keras
CNN_VARA018b_128x128_EP30_ACCU95.71_25-06-2024.keras
RESNET50_VARA08b_128x128_EP20_ACCU96.7_24-06-2024.keras
RESNET50_VARA08b_128x128_EP30_ACCU98.98_24-06-2024.keras
VGG16_VARA03b_128x128_EP20_ACCU99.41_24-06-2024.keras
VGG16_VARA03b_128x128_EP30_ACCU99.56_24-06-2024.keras

Figure 9: Naming Scheme of Models starting off with the type, variant, input feature size, epoch count, accuracy and date.

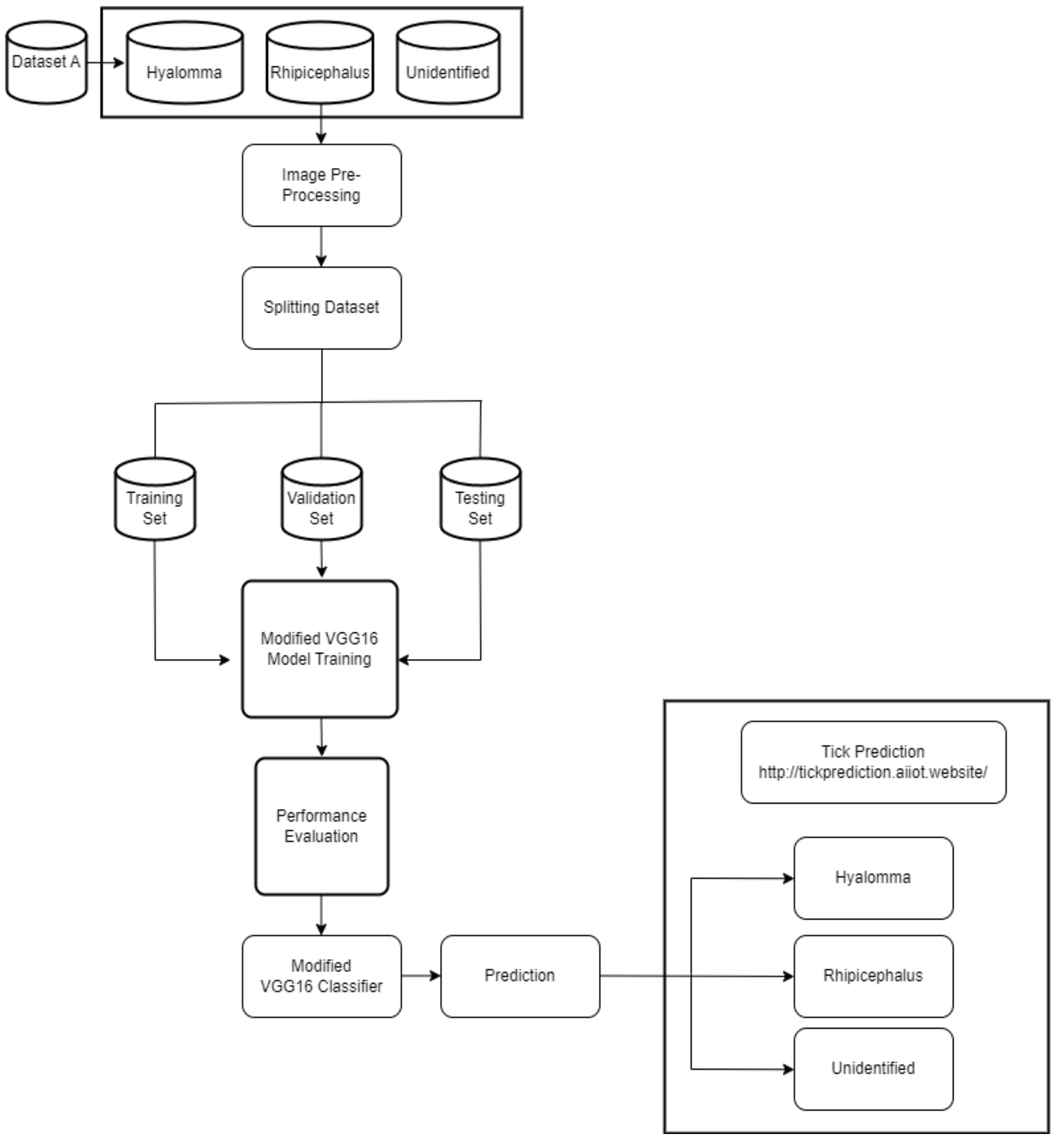


Figure 10: Flowchart depecting the processes of this study making a model with dataset A.

Figure 10 is a flowchart detailing the full process from training the models using dataset A, to using the models and outputting the correct class. The model is fed data in the form of images listed in the 3 subclasses, data is preprocessed, split and kept into 3 sets for training, development validation and testing. The model is trained, evaluated and saved. From then the model is kept onto a WebUI for prediction. The results will be seen visibly by the user from the respective image uploaded.

Similarly, Figure 11, details the same processes with minor changes, where dataset B was used with 7 classifiers. The images go through the same processes. At the end, when the image is predicted, the tick is identified as well as it’s positions, whether it’s HeadShot, CenterShot, CornerShot for each bug respectively or Unidentified.

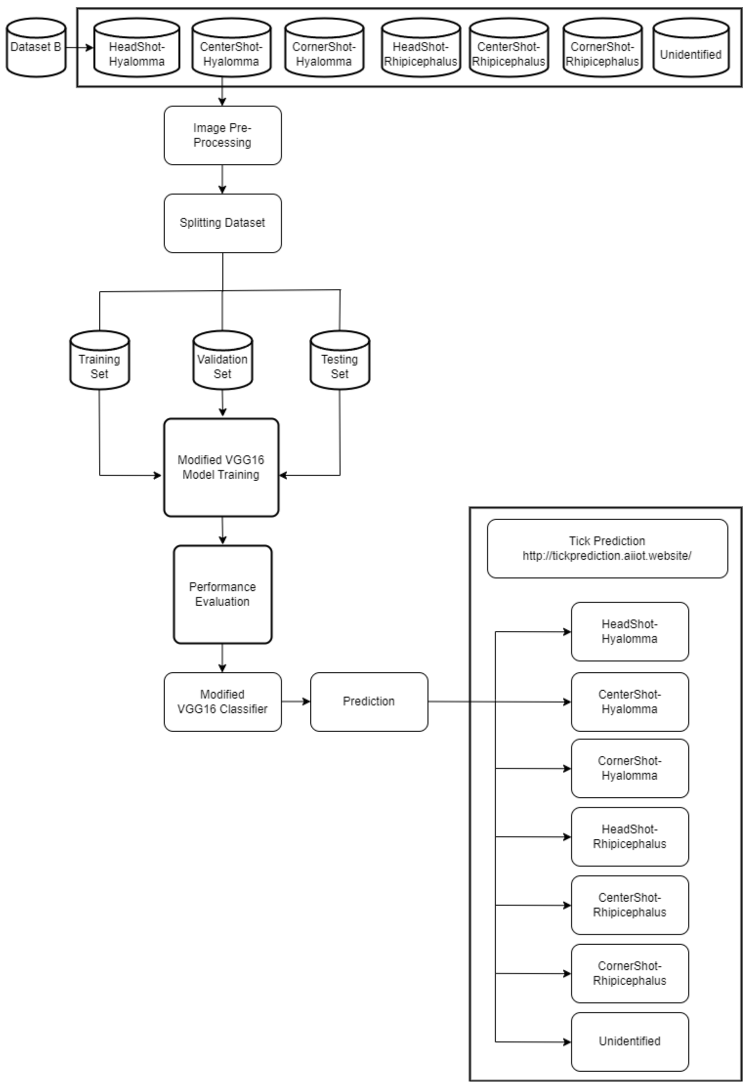


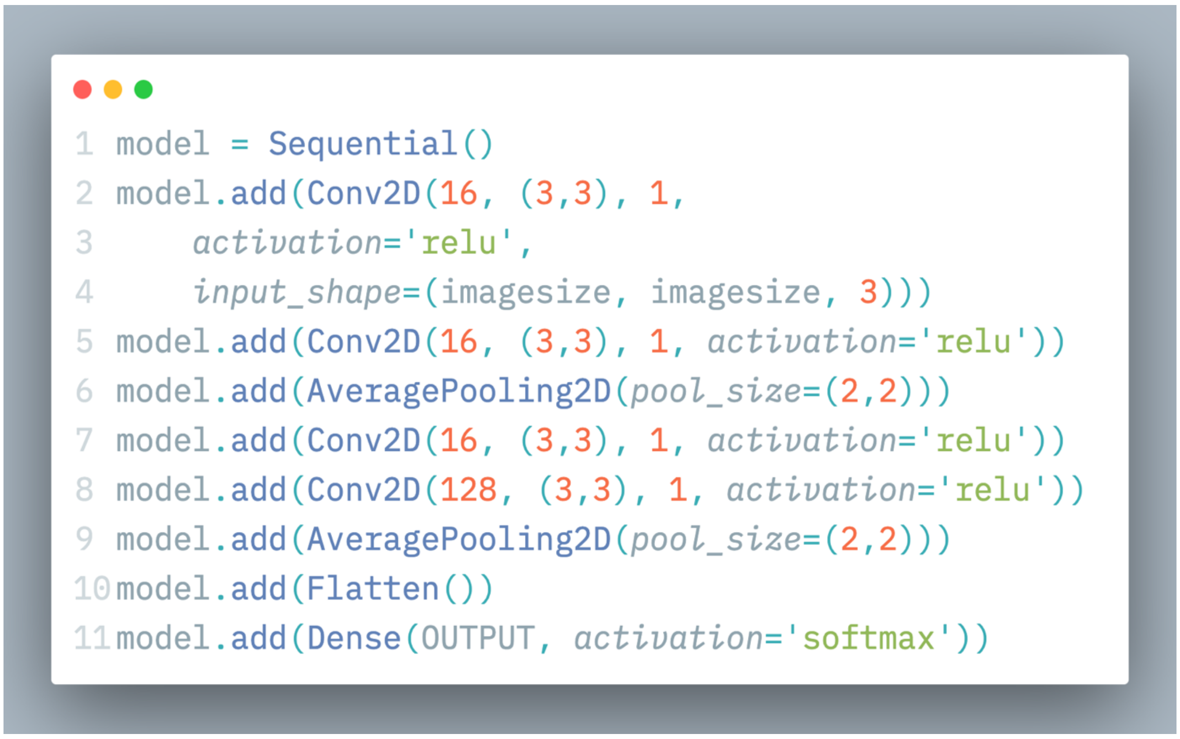
Figure 11: A flowchart depicting the process of making a model with dataset B.

3.7.1 CNN

3.7.1.1 Introduction to CNN

Convolutional Neural Networks (CNN) are at the heart of most deep learning networks. They perform convolutions over matrices that include the data that we are training with to extract features that we can use to identify or predict a solution. In this particular case, when classifying images, we can use CNN models to be able to extract features that we may not be able to manually engineer at hand and use them to automatically classify the tick bugs. Features closer to the input of a CNN model are very primitive, including strokes, likes and edges of objects. In the middle of the CNN network, the features would include shapes, portions or shapes or shadows of the image. In the final layers, the model will be able to fully understand the image to reveal, segmentations, object boundaries or the object itself. This helps in creating an appropriate cost function and respective weight parameters that can be utilised to predict new images that a model has not seen yet.

Deep learning models require that we create more than 3 of these layers. This model was created with a 5-layer network that shall be referred to as our basic CNN model. The 5 layers included only valid convolutional layers. Average Pooling and Flattening layers were not counted. All the deep learning models in this paper are versions of CNN. Apart from this basic CNN model, the rest are advanced networks with very nuanced architectures. Figure 12 showcases the structure of layers of this model in code form.



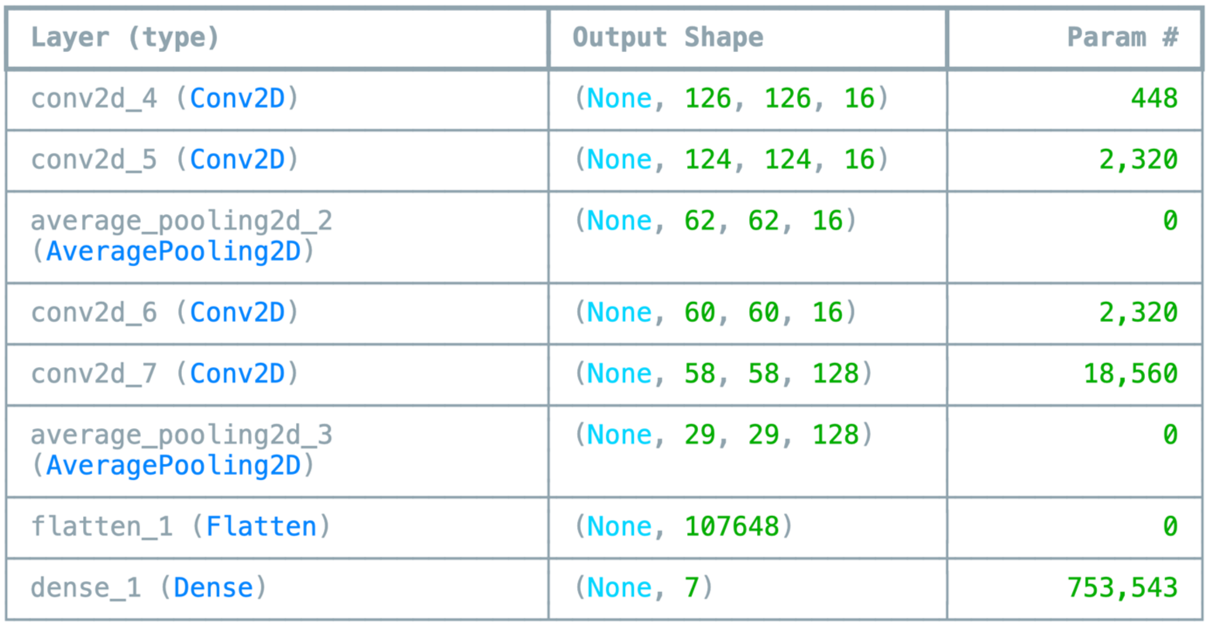
```
model = Sequential()
model.add(Conv2D(16, (3,3), 1,
    activation='relu',
    input_shape=(imagesize, imagesize, 3)))
model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(AveragePooling2D(pool_size=(2,2)))
model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(Conv2D(128, (3,3), 1, activation='relu'))
model.add(AveragePooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(OUTPUT, activation='softmax'))
```

Figure 12: The layers of the basic CNN model. Only Convolution layers and fully connected layers were counted.

The choice of using a shallow model first is to test out if the objective is feasible. Can the pictures actually produce any form of classification? If so, how accurate are the classifications and what problems is the model facing that we can solve to make it better? This basic CNN was the base research for most of the other models. From this model, we were able to figure out the range of epochs the models would run for, the range of learning rates that would be suitable as well as batch sizes.

The major disadvantage of this model is that it's shallow, it doesn't have enough layers to be able to fully comprehend all the features of the images. We cannot tell for sure if it can understand the legs, heads or bodies of the bugs. However, we know that it can get some basic shapes of the bugs. The layer summary of this model can be seen in Figure 13.

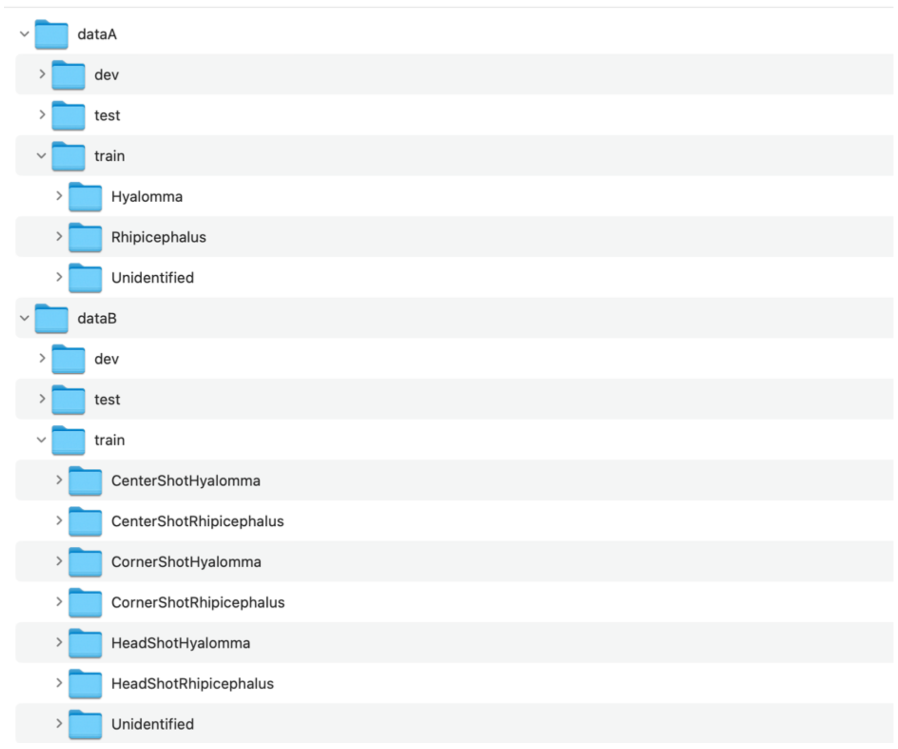
One advantage is that the build for the model is very small. The size of the model is measured at 4.2MB for a 128x128 input feature size. This can be particularly useful in a memory constraint system where we have to load a very small model to be able to make predictions. It was observed during training and deployment that RAM was a huge resource the models seemed to consume. This model consumed the least amount of RAM and disk space as it weighed around the same as the images themselves.



Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 126, 126, 16)	448
conv2d_5 (Conv2D)	(None, 124, 124, 16)	2,320
average_pooling2d_2 (AveragePooling2D)	(None, 62, 62, 16)	0
conv2d_6 (Conv2D)	(None, 60, 60, 16)	2,320
conv2d_7 (Conv2D)	(None, 58, 58, 128)	18,560
average_pooling2d_3 (AveragePooling2D)	(None, 29, 29, 128)	0
flatten_1 (Flatten)	(None, 107648)	0
dense_1 (Dense)	(None, 7)	753,543

Figure 13: Parameter list of basic CNN model with 32x32 feature input size set.

Training for the models was done over 2 sets of data that are denoted in Figure 14 as dataA and dataB. Dataset A included all the images at once, whilst B was subclassed further to reduce bias.



```
dataA
  dev
  test
  train
    Hyalomma
    Rhipicephalus
    Unidentified
dataB
  dev
  test
  train
    CenterShotHyalomma
    CenterShotRhipicephalus
    CornerShotHyalomma
    CornerShotRhipicephalus
    HeadShotHyalomma
    HeadShotRhipicephalus
    Unidentified
```

Figure 14: Dataset folder structures of original major classes and new subclasses.

3.7.1.2 CNN-A Model

This was the first variant of the basic CNN model. It included the 3 classes, "Hyalomma", "Rhipicephalus" and "Unidentified". The images were shuffled and passed through the training session inside a Jupyter file.

The model was trained with an input feature size of 128x128. It was started off with an epoch count of 20. The data was saved and then it was retrained with an epoch count of 30.

3.7.1.3 CNN-B Model

The second variant of the CNN model used the same architecture as the first variant, we only had 2 differences with this model. Firstly, the dataset changed from A to B and some of the hyperparameters changed so as to increase the accuracy of the model.

3.7.2 VGG16

3.7.2.1 Introduction

VGG16 is a very robust Deep Neural Network that can accurately classify images into the correct class. The network was of interest due to transfer learning as it would take less time to train the model from start to finish. This is a 16 Convolutional Neural Network layer model that extracts features from images with high accuracy. The layers can be observed below in Figure 15:

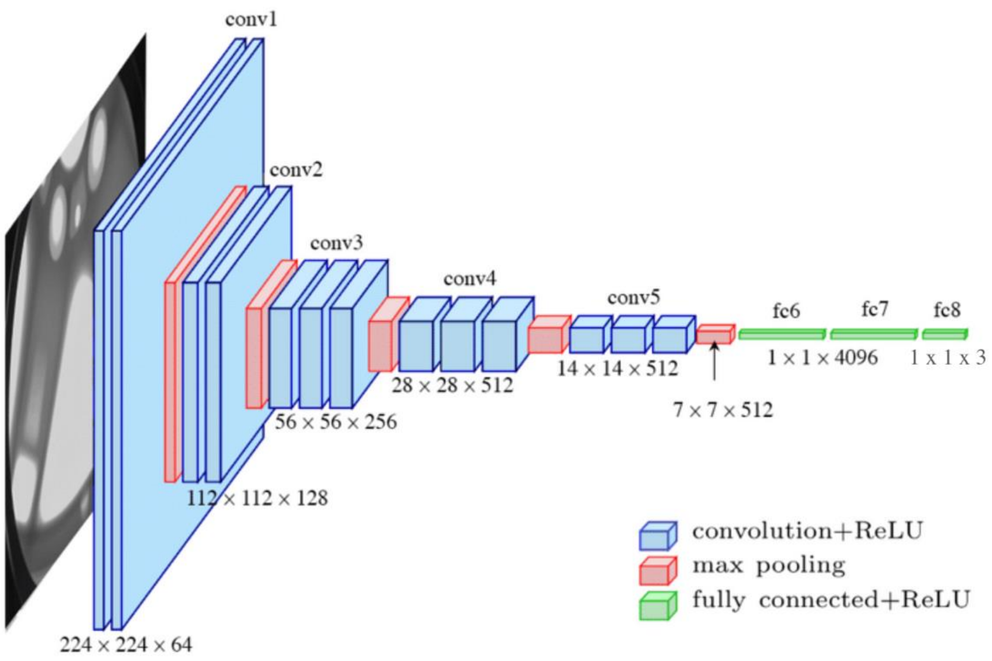


Figure 15: Layer overview of the VGG16 network. The model's output is 3 for dataset A and 7 for dataset B.

Another major advantage was due to the limitation of data we had to begin with. With around 15k worth of data, it was clear that we needed a boost from some architecture to be able to accurately classify the images correctly. VGG16 contains enough layers to be able to extract the base features from the image as well as extract some nuanced features in its later layers to be able to group certain qualities of the bugs. These qualities, such as rounded edges, longer legs or deeper centres could be features that distinguish the ticks from one another. These features are all clumped up within the black box of the VGG16 model and it is hard to almost impossible to see what the model is doing behind the curtain.

Because of the robustness of the network, we could freeze the early layers of the VGG16 model to stop them from training to keep the early weights and biases intact. This means that we can extract the earlier features such as edge detection and horizontal and vertical lines of the Hyalomma and Rhipicephalus tick bugs. To be able to fine-tune the network, we unfreeze the later fully connected layers and add them to the model so that they can be retrained. These later layers will take on the shape, weights and biases of the new images to comprehend the nuanced features of the bugs. The final layer, the softmax layer, had different unit numbers depending on the folder structure that was passed in. We had 2 almost identical model architectures, that had only this one minor difference. The first model which is referred to as VGG16-A had 3 softmax units which were trained on the dataset where there were only 3 classifiers, "Hyalomma", "Rhipicephalus" and "Unidentified". The second model which is referred to as VGG16-B had 7 softmax units which were trained on the second dataset that had extra subclasses. There were a total of 7 subclasses that are later enumerated in detail.

Transfer learning was tested with both variants of this model on a personal computer, however, after multiple experiments, the decision was made to rebuild the network architecture from scratch and train the weights from the beginning. The biggest issue was that gradient descent seemed to be going in the wrong direction and producing high values for loss and stagnant values for accuracies. What transfer learning does is use weights from past models, freeze them in place, and let only the final layers of the model get trained. This increases the accuracy of the model as well as reduces the time it takes to train the models. One major downside is the fixed size of feature sizes that come with retrained models that we are trying to transfer learn or fine-tune. It is common to find only one size of feature input set. Changing those sizes for a model means that the old weights associated with that model will not work, as they will differ. For this study, we were tested to see if the image size increased the accuracy of the model in any way. To achieve this, new builds of the model with respective feature input sizes had to be initialised before training.

Both variants of this model, named VGG16-VARA denoting the A folder variant of the VGG16 architecture network and VGG16-VARB denoting the B folder variant used 4 different input feature sets to test out their accuracies and performance. The choice was to double the powers of 2 from 32 to 1024. However, 512 and 1024 were not an option as some of the images could not scale down to this size. Secondly, images of this size caused memory to run out extremely quickly when training because the kernels had to multiply and convolve really large matrices together. Most of the runs resulted in crashes within the first epoch.

3.7.2.2 VGG16-A Model

The VGG16-A Model which is the first variant of the two available VGG16 models for this report, houses a 16-layer architecture that can accurately predict one of the 3 given classes, "Hyalomma", "Rhipicephalus" and "Unidentified". The main difference between variants A and B of these VGG16 models is the dataset structure from which it pulls the data as well as the training to validation split.

The data for this model was housed within 3 subfolders, Hyalomma, Rhipicephalus, and Unidentified. All the images were manually shuffled before the data was placed into the working directory for model training. This ensured that the images of the augmented bugs were not clumped together as shown in the Figure 16.

Figure 16: A folder view of re-shuffled images.

Data was loaded through a flow generator from Keras to not cause any memory issues when loading large images. This enabled efficient memory use especially while training the data on paperspace's gradient ML platform. Models with an input feature larger than 512 pixels for their height and width more than 2 GB worth of model size which required over 60 GB of RAM to train their hyperparameters and a larger GPU core than 16GB. This was the breaking point for the images as well, as the unidentified portion of the data included images that had heights less than 500 pixels. Scaling images higher than this to 1024x1024 seemed impractical and would have tripled or quadrupled the RAM required and could have crashed the server.

At first, it felt right to only train models using higher pixel counts on the server as their input features were exponentially large. Training on Paperspace with a lot of RAM makes this process a breeze. However, in the end, to reduce bias, all the models were trained on the server to level the playing field between them and produce fair results for discussion.

After the images were loaded, they were transformed by reshaping them into various input sizes such as 128 by 128 pixels. We chose this number as most images were fairly large and of high quality, and the assumption that they would still stay that way after reducing a 1/8 of their size was not wrong. The images retained their quality and features of the bugs. The images were then normalised by 255 to uniformly range the values of the pixels from 0 to 1. This helped in performance to train the model quickly and effectively as gradient descent moves down its plane a lot quicker when the values take on shapes that are closer to each other than when they are larger and sparsely correlated.

The images were not augmented with code due to 2 reasons specifically for this variant of the model. Firstly, the model was tested against its ability to use its transfer learning capabilities to use new images from a different dataset to retrain the network. Secondly, the images had already been manually augmented from the lab by shifting the position of the bug, rotating the images, and zooming them at different positions. This meant that we were trying to reduce the effect of this manual augmentation as well as use it to our advantage.

This variant of the VGG16 model split the data between training, development and testing sets with a 50/40/10 ratio split respectively. This split resulted in the image value mentioned before in Table 2. The choice of using both a development and testing set helped in reducing overfitting on both the training set and the development set. This in turn increased the accuracy and reduced the training loss in the final testing set. This model is midway between being shallow and too deep. This split was a safe bet knowing that the test set would be able to handle shaping the model to not overfit.

Larger input sizes have the benefit of using more of the data and producing higher quality predictions, but become less performant and can utilise a lot of RAM and CPU power. On some machines, these large models may crash the system/server or may not run at all. On the other hand, using the smaller input feature size models results in a tinier model, sometimes only kilobytes in size. These models have the benefit of using fewer computer resources to make a prediction, they take up less space, thus can be put on smaller-sized machines and take less time to produce results. However, they may not be as accurate and may have misleading results.

We set out to only use one input feature size to make the comparisons fair. 128x128 was chosen as the default as well for VGG16 models. This model took 8 hours to train on a personal computer, but only 30 minutes on the server. The model was first trained over 20 epochs, and then again at 30 epoch. The results for this model are shown in the results and discussion section under Table 3 and Table 7 for variants A and variants B respectively.

3.7.2.3 VGG16-B Model

The VGG16-B model was the second variant for the VGG16 network architecture. This model utilised the same hyper-parameters as the first variant with only folder structure changes. The hypothesis for this model was to try and reduce bias over the images as well as the manual data augmentation performed in the lab. Images that were processed under the microscope in the laboratory were cropped, zoomed in, rotated and Z-flipped (mirror-flipped). The purpose of the data augmentation was to increase the image count of the tick bugs to improve the model accuracy. However, this can increase the bias in the model, as we are using the same bug over and over again. On top of this, the position of the bug changes within the photos. By changing the structure from dataset A to dataset B defined in Figure 10, we change the final output layer result count from 3 classifications to 7 classifications.

Thus, the data for this variant is housed under the datasetB folder, previously denoted under Figure 14. By using tensor flow data flow generators, the images were passed through into the VGG16 architecture, under the same ratio split of 60/25/10 split for the training, development and testing sets. Because the images are still in the same amounts, we had the same amount of images for all three ratios. However, the images under each folder for the subclasses changed. Counts can be revised again in Table 2. This creates a disadvantage over the first variant, as it has more images on the test set. The test set balances out the training set and the dev set from overfitting. But now the set is spread over 7 classifications instead of 3. Hypothetically, this variant should have performed better and had major advantages over the other variant. However, it can only work with a large dataset. By reducing the data count over a class into subclasses, we have reduced the count by almost 30%. If the count of these images were increased to the same size as the first variant, it would be possible to reach a high accuracy.

3.7.3 ResNet50

3.7.3.1 Introduction to ResNet50

Resnet is a large CNN architecture of a model that contains 50 layers stacked together efficiently to be able to produce accurate results despite the depth in layers. ResNet is a model built by Microsoft in 2015 and utilises residual blocks that capture information in one spot and send that data forward one to two layers, creating skip connections. This helps retain older information that could have been lost when traversing down really large models. This is a solution to training deep networks, as when models get deeper and deeper, their accuracy starts to vanish and the loss degrades. Optimisers find it hard to figure out the correct path down gradient descent.

Skip connections, are just regular layers, but they don't connect directly to the following layer. They use their output and push it further as the input to another layer. By utilising convolutions within a network, features that are extracted in layer 2 can be re-fed into layer 4, then layer 6 and so forth. Thus this makes the model extremely robust. One of the largest downsides of using models this big is that you need a lot of data and a lot of training time to be able to reach the accuracies required. The limitation facing this project of not having enough images makes it challenging to reach our optimising metric. On top of this, transfer learning won't work because we have multiple sets of feature input sizes.

The models were thus built from scratch, creating each layer one by one. Figure 17 shows the structure of layers for the Resnet50 model.

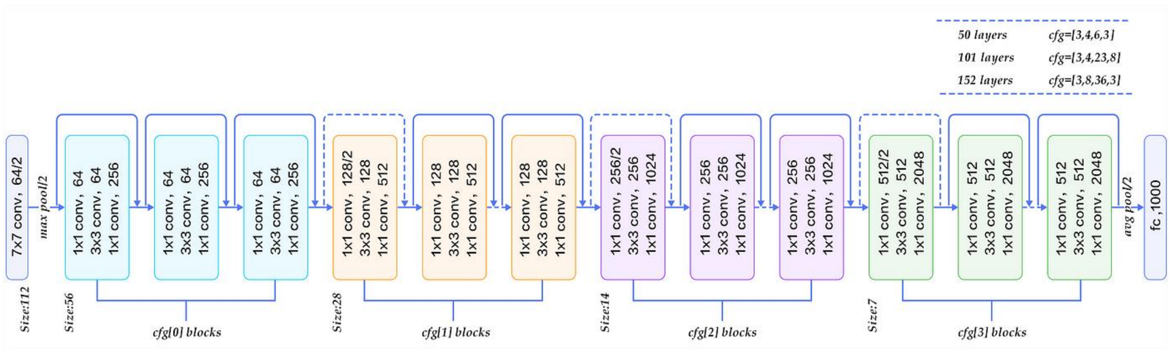


Figure 17: The layer structure of the Resnet50 model.

By combining these blocks, we can then form the whole network. This monolith is built to take on any challenge given the right amount of data. The model was set by generator images split into the same ratio of 50/40/10.

The model was fed the same two datasets, A and B, and training was tried on a personal computer and then sent to the server. Because of the nature of the model, 50 layers were quite a challenge to train on the server. Convolutions would almost always fail on the 256x256 model when run on a single-core GPU. The model would eat up all the RAM, and all the GPU resources, and almost freeze the entire session. We settled again only using 128x128 input feature size for ResNet50 models.

3.7.3.2 ResNet50 A

This is the first variant of this model that uses the first dataset. All the images were passed through the model. What was different this time was that the model was extremely large and our server was running under heat. It was extremely overloaded performing arduous convolutions and multiplication of matrices. On top of this, the learning rate was set very low and that slowed things even further. To combat this, the image generator had to be batched. The batch size was 32. This sped up the epoch running rate and quickened things. Almost every single run tried without batch sizes caused errors and crashed the Jupyter kernel. The model would train between 30 minutes to an hour and at times, they had to be retrained because of detection of overfitting.

3.7.3.4 ResNet50 B

The second variant of the ResNet50 model utilised the second dataset. Dataset B was now further skewed in size and increased the output count from 3 to 7. This majorly made training on the ResNet model harder. The parameters increased exponentially. Rather than training on one GPU core, we had to combine all the power from the 2 cores to be able to allow convolutions to happen in between epoch runs. This model was also trained for both 20 and 30 epochs.

3.7.4 Web App

3.7.4.1 Introduction to The WebApp

A web application was made to complement the models created in this paper. Predicting images through the CLI can be extremely cumbersome, as you have to type commands to run each prediction. Using an application that can communicate with models on the backend and produce a visual look of the results that are generated by the model equips users to predict images of different kinds through different models to compare results. These results are automatically saved for future review and comparisons.

The web application is built using an array of languages that weld together a coherent experience for predicting tick bugs. The concept of the application is to use an API to communicate with the model backend and return results to the frontend.

3.7.4.2 The API

The model was built using Python and a popular framework known as TensorFlow. It's only fitting that the API follows suit as well. For compatibility reasons, none of the models we compressed, converted into other versions or saved into other formats. All models were saved as keras files that ensured compatibility for calling the API. The API was built with Flask, a very popular tool for building sites and APIs using Python. It was served with a WSGI framework named Waitress over a safe port.

The API lives on a different server from the frontend UI. This is to ensure that the models can take full advantage of the resources in the system and not cause any performance bottlenecks when performing predictions. API calls are produced using a URL that is passed from the front end to the backend. On a successful result, the server returns a 200 status response with a valid result. The front end picks up the results and saves all the metadata into the database. This process varies with internet speeds as the image needs to move from the frontend server to the API server. The larger the image that is uploaded, the longer it will take. 128x128 input sizes made the whole process a lot smoother and quicker. Results came back in a snap.

3.7.4.3 The Frontend

The front end is the view of the web app. This is what the user is seeing, including, buttons, images and text. By manipulating the DOM using JavaScript, it was possible to render dynamic content on the application that was generated from predictions that arrived from the API. The front end loads on the login page where the user is required to authenticate himself to gain access. Authentication is required so that people who do not have access cannot tamper with the system and cause harm to the API calls and models.

After authentication, the user is greeted with a dashboard that includes quick links for predictions, a small list of recent past predictions and a list of top accurate available models. This is where the user can perform quick clicks to reach a required objective.

The menu on the side also provides quick access to other pages. Notable mentions for this paper is the knowledgeable section where normal users and scientists can view the human error of predicting tick bugs and take a 5-10 minute test to prove the human error of predicting tick bugs. This will help set a bar for future research and model training as we can tell when the model is overfitting or underfitting. By using the human error rate, we can assume that we are getting closer to Baye's theorem. This can help us close in on the avoidable bias of the model.

The prediction feature on the web application utilises multiple HTML elements that guide the user on how to predict an image. Firstly, a model is preselected from the available list.

The preselection is automatic and is filtered by descending order of accuracy of the models. The next step is to select an image from the user's device and start the prediction. This is where the front end calls an asynchronous function that communicates with the API server. The API server pulls the image that has been uploaded and starts the predictions. A few seconds is all that is needed to perform a prediction. The server spits out results and wraps them as a response with appropriate headers to ensure integrity. The result is unwrapped from its headers and parsed to reveal the final result. The data is saved into the database, triggering a new popup that showcases the final result of the prediction with metadata details of the model used along with its accuracy. Figures 18 and 19 showcase the frontend view of the application

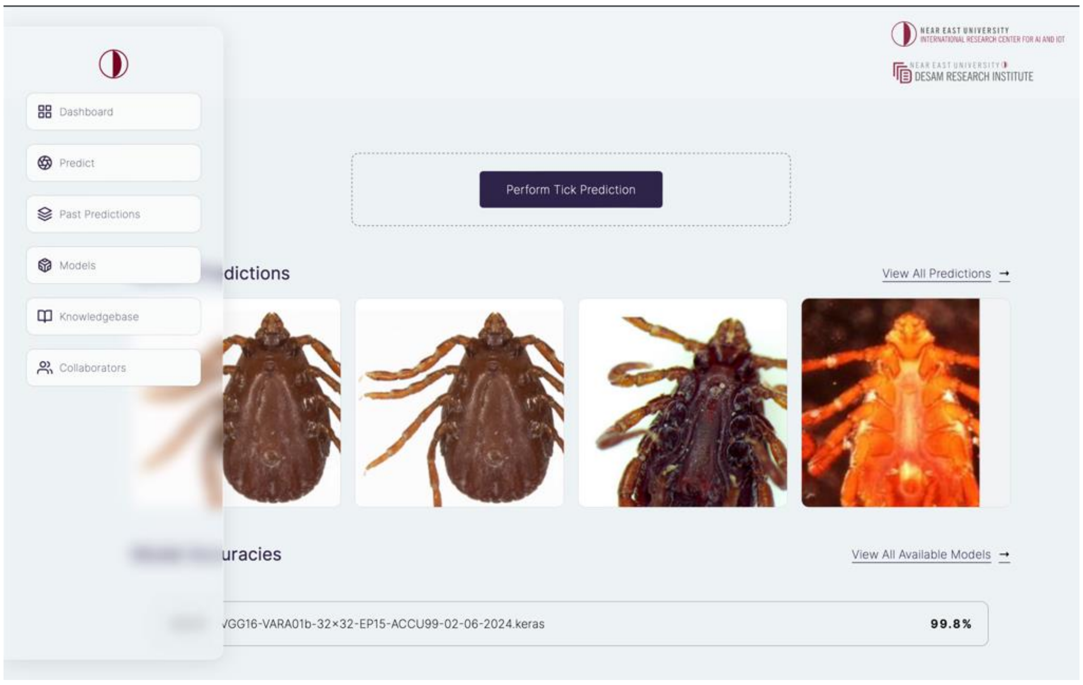


Figure 18: Tick bug prediction web application dashboard page.

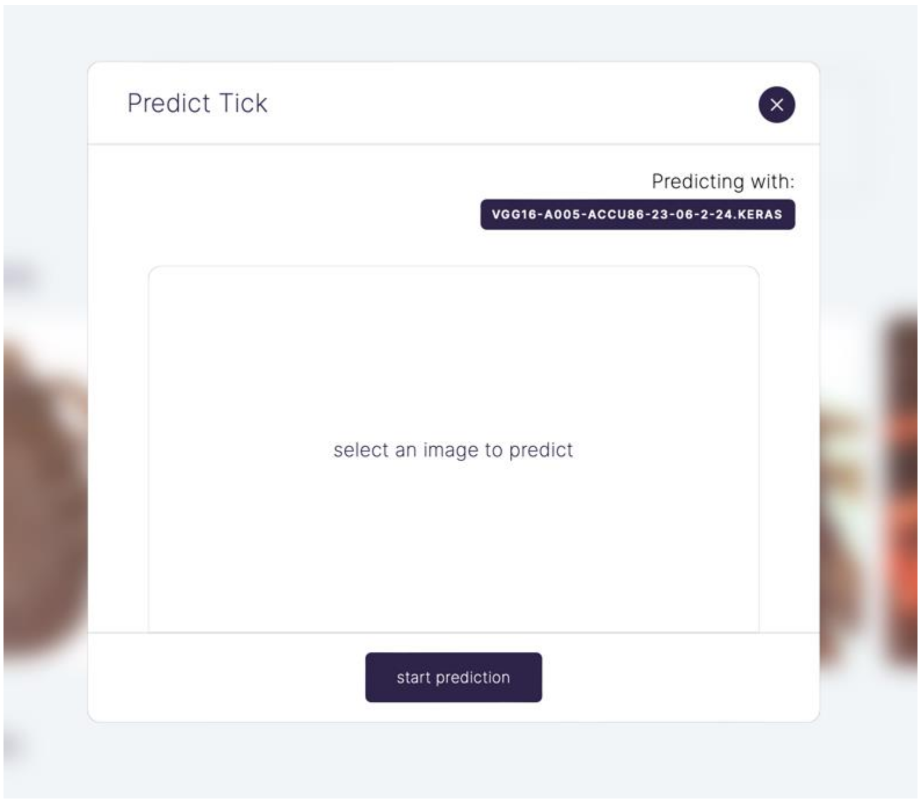

Predict Tick
×
Predicting with:
VGG16-A005-ACCU86-23-06-2-24.KERAS
select an image to predict
start prediction

Figure 19: Image upload popup to send photo to server.

3.7.4.4 The Database

The database chosen was a SQL database known as MySQL. This enables performant search queries and powerful join calls through tables to make the web application function. Multiple tables are housed in this database. A notable mention of the database is the way the predictions are saved. Each prediction is linked with an id the model that was used to predict the image along with the date of prediction. This helps in linking back each image to its parent model. Further development of the application will help predict the same image with multiple models and compare multiple results at once. The database also enables us to distinguish which predictions belong to which user. Each user will only be able to view their predictions and have separate model prediction settings.

CHAPTER IV

Results Analysis and Discussion

Deep learning models all come with accuracies and loss metrics to represent how well they can perform in prediction both in the training sessions as well as test sessions. The test session is more crucial as this is our target to be able to make real-world predictions. The accuracy that comes from the test set can make or break a model. Various model architectures are known to reach extremely high values of accuracy. Because deep learning models differ from machine learning models, we do not have MSE, as this only works with continuous data prediction. With classification-type problems, models take on a simpler error rate of subtracting 100 per cent out of the total accuracy. The models were all tested on the same folders of test sets to ensure integrity. Their corresponding correctness can be seen in confusion matrices that display how many images the model got right, and how many images the model classified as other terms.

The results are grouped by models and then discussed together to compete for the best 3 models that were featured on the front end of the web application. All models were available on the UI to test with, and they were all saved for future comparison.

4.1 Dataset A

Models created from dataset A produced very streamlined results. They incorporated 3 output classifications, "Hyalomma", "Rhipicephalus" and "Unidentified". The models trained pretty quickly due to less parameters in their layers. Results for the accuracies of all the models can be seen in Table 3. VGG16 topped the charts with an accuracy of 99.57% for the 30 epoch run during the testing phase.

Table 3: Accuracies for training, validation and testing sessions respectively for all models with 20 and 30 epochs using the first dataset.

Model Architecture	**Training Accuracy %**		**Validation Accuracy %**		**Testing Accuracy %**	
	20 epochs	**30 epochs**	**20 epochs**	**30 epochs**	**20 epochs**	**30 epochs**
VGG16	99.75	99.93	99.75	99.92	99.42	99.57
RESNET50	99.47	98.88	99.47	98.88	96.70	98.98
CNN	97.15	99.23	97.15	99.23	94.98	95.71

Table 4: Loss values for training, validation and testing sessions respectively for all models with 20 and 30 epochs using the first dataset.

Model Architecture	**Training Loss**		**Validation Loss**		**Testing Loss**	
	20 epochs	**30 epochs**	**20 epochs**	**30 epochs**	**20 epochs**	**30 epochs**
VGG16	0.0072	0.0026	0.0073	0.0025	0.0186	0.0175
RESNET50	0.0183	0.0303	0.0183	0.0303	0.1006	0.0297
CNN	0.0879	0.0329	0.0879	0.0329	0.1390	0.1336

Table 4, shows the losses of the models for both 20 and 30 epochs. VGG 16 had the lowest losses from the testing phase with a value of 0.0175. This meant that the model was getting close to the global minimum of the cost function for identifying tick bugs.

Table 5: Evaluation metrics for all 3 model architectures including precision, recall, F1 score, sentisitivity, specificity and AUC percentages for the 20 epoch run.

Model Architecture	**Class**	**Precision %**	**Recall %**	**F1-Score %**	**Sensitivity %**	**Specificity %**	**AUC %**
VGG16	Hyalomma	99.16	98.95	99.06	98.95	99.88	100
	Rhipicephalus	98.83	99.41	99.12	99.41	99.49	100
	Unidentified	99.78	99.52	99.65	99.51	99.70	100
RESNET50	Hyalomma	90.75	98.54	94.48	98.54	98.61	100
	Rhipicephalus	94.74	95.21	94.97	95.21	97.71	99
	Unidentified	99.15	97.10	98.11	97.10	98.86	100
CNN	Hyalomma	80.35	96.65	87.75	96.65	96.74	99
	Rhipicephalus	94.75	92.44	93.58	92.44	97.79	99
	Unidentified	98.91	95.96	97.42	95.96	98.56	100

Table 5, lists the evaluation metrics for the 20 epoch run model for the three model architectures. We can see that the models perform well with high precision values and F1-scores. The table lists the values for all three classes respectively. Out of all the results, the Hyalomma class had the lowest precision for the CNN model with an 80.35% and an F1 score of 87.75%. VGG16 took the spotlight with a precision score of 99.83% and an F1 score of 99.12% for the Rhipicephalus class.

Table 6: Evaluation metrics for all 3 model architectures including precision, recall, F1 score, sentisitivity, specificity and AUC percentages for the 30 epoch run.

Model Architecture	**Class**	**Precision %**	**Recall %**	**F1-Score %**	**Sensitivity %**	**Specificity %**	**AUC %**

VGG16	Hyalomma	99.37	99.58	99.48	99.58	99.91	100
	Rhipicephalus	99.08	99.58	99.33	99.58	99.60	100
	Unidentified	99.87	99.56	99.71	99.56	99.82	100
RESNET50	Hyalomma	97.68	97.07	97.38	97.07	99.68	100
	Rhipicephalus	99.07	98.24	98.65	98.24	99.60	100
	Unidentified	99.21	99.78	99.49	99.78	98.92	100
CNN	Hyalomma	92.97	88.49	90.67	88.49	99.08	100
	Rhipicephalus	91.99	96.47	94.18	96.47	96.37	99
	Unidentified	98.35	96.83	97.59	96.84	97.78	100

Table 6, lists the evaluation metrics for the 30 epoch run models architectures. All the models had precisions and F1 scores higher than 90%. The top 2 contenders where VGG16 and ResNet50 with approximate scores of 99% for the precisions as well as the F1 scores.

4.1.1 CNN-VAR A Model

Models created from the basic CNN architectures had surprisingly great performance as well as accuracy. Different hyperparameters were passed in to try and improve the accuracy, to reach testing accuraries higher than 95%. However, this was only possible with higher epoch counts. The 30 epoch count pushed to reach an accuracy of 95%. A shallow model was capable in this case of reaching high accuracy with the dev test set using this small split of data. It met the optimizing metric that we set before.

This is the first variant of the CNN model that utilised dataset A. It performed a lot better than the second dataset, but not as well as the deeper models. However, because of it's small size, it becomes a great contender in situations where network is slow, where we need a very small sized model that takes up less resources and can predict very quickly.

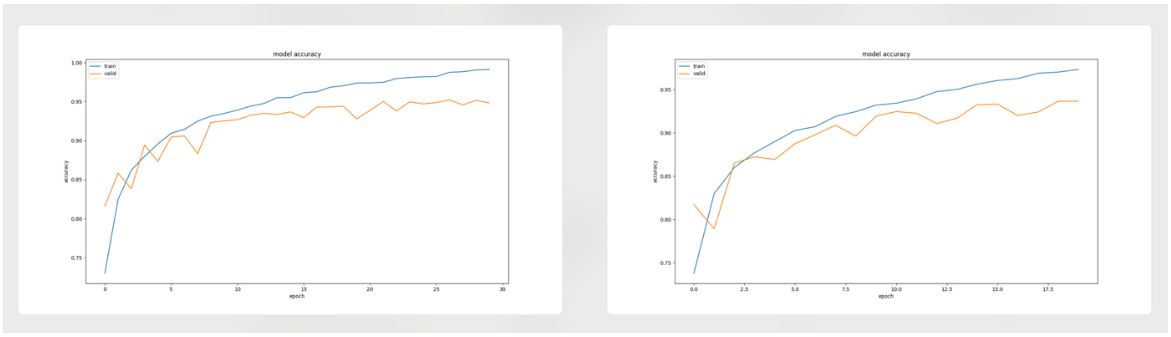


Figure 20: Accuracies for the CNN variant A models for both training and validation sessions. The graph on the left represents the 20 epoch run and the right represents 30 epoch run.

The graphs from Figure 20, show the accuracies for both training and validation sessions for both 20 and 30 epochs. During training, the CNN models would reach final accuracies close to 97% but then fell short to around 95% during validation.

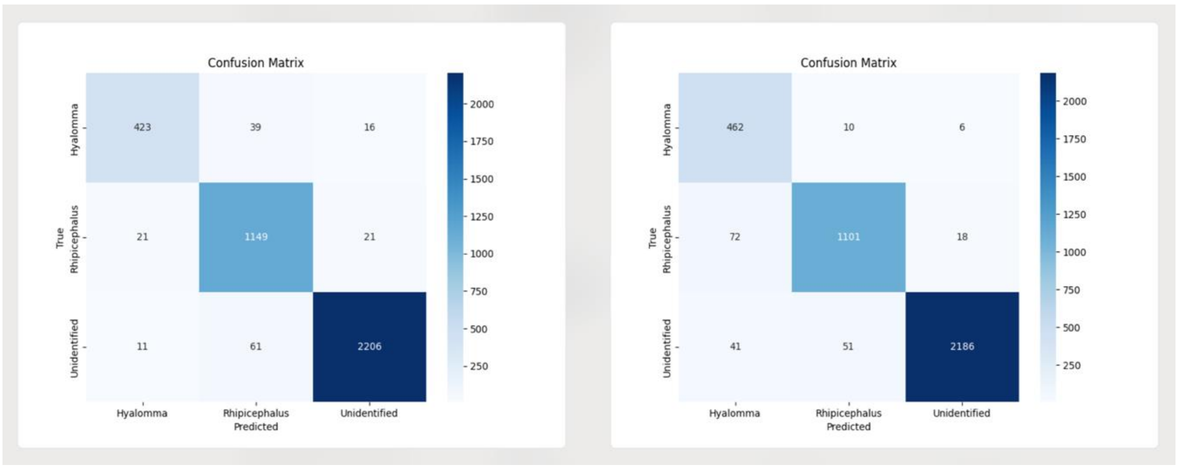


Figure 21: Confusion Matrices for the CNN variant A models for 20 epoch runs on the left and 30 epoch runs on the right.

The confusion matrices from Figure 21, showcase how many images the models got correct and ones they mislabeled. CNN produced accurate results for most of the images in the test set, but mislabled upto 113 incorrect labels for the Hyalomma class, 61 incorrect labels for Rhipicephalus and 24 for Unidentified. Further improvements are needed for this model to reduce these numbers and get them close to zero.

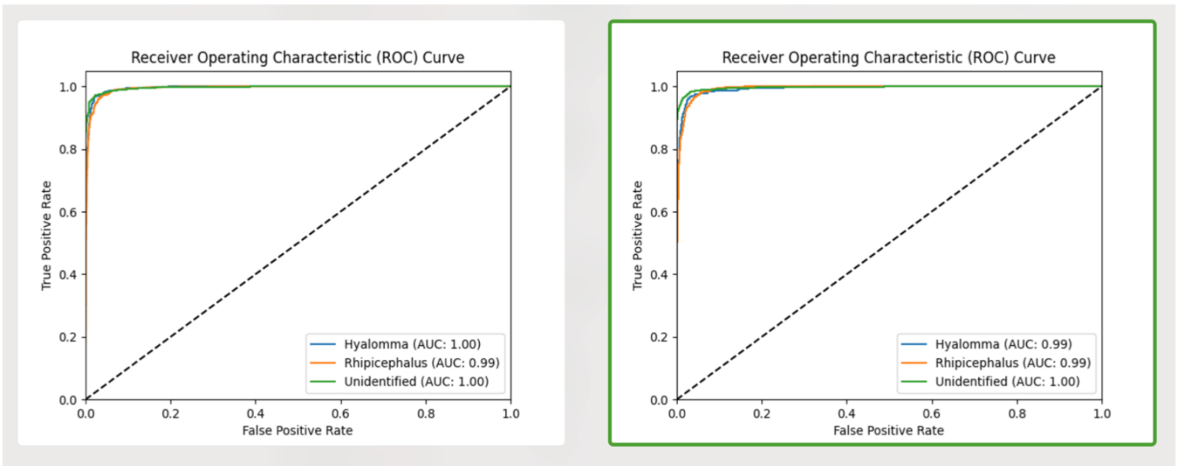


Figure 22: ROC curves for the CNN Variant A models for the 20 epoch runs on the left and 30 epoch runs on the right.

The ROC curve signifies stable predictions for the model under different thresholds. The AUC for the classes are approximately 1.

4.1.2 VGG16-VAR A Model

VGG16 models exhibited extreme robustness to the images, being able to reach very high values of accuracy as well as low values of training and testing loss. The models would train from a minimum of 30 minutes to an hour and only some had 2 varying epochs sizes. The difference in accuracy between the 2 epoch sizes of 20 and 30 wasn't massive. But it still pushed it by 0.1%. The accuracy for the 30 epoch count managed to reach 99.57%.

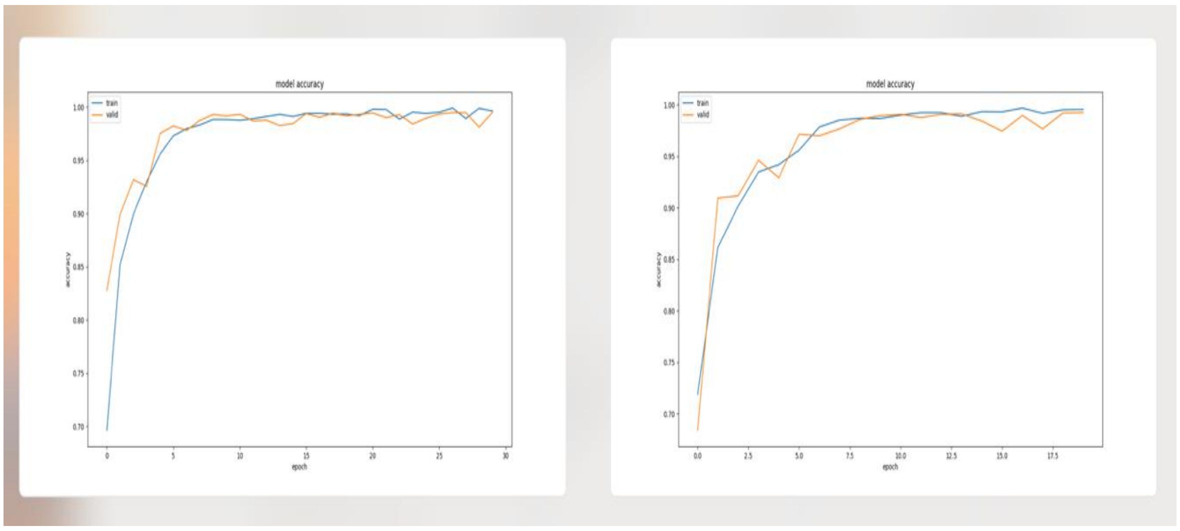


Figure 23: Accuracies for the VGG16 variant A models for both training and validation sessions. The graph on the left represents the 20 epoch run and the right represents 30 epoch run.

From Table 3, we can see that the 30 epoch count outperforms the 20 epoch run model by a small margin of 0.15% for the accuracies during training and validation sessions. The VGG16 models performed outstandingly and produced accurate results almost all the time.

Figure 24 showcases confusion matrices for both 20 and 30 epoch runs Here we can observe, how the model is performing on the classification keys. Surprisingly, the 20 epochs run model had fewer incorrect values with only 3 incorrect for Hyalomma, 11 incorrect for Rhipicephalus and 3 incorrect for Unidentified.

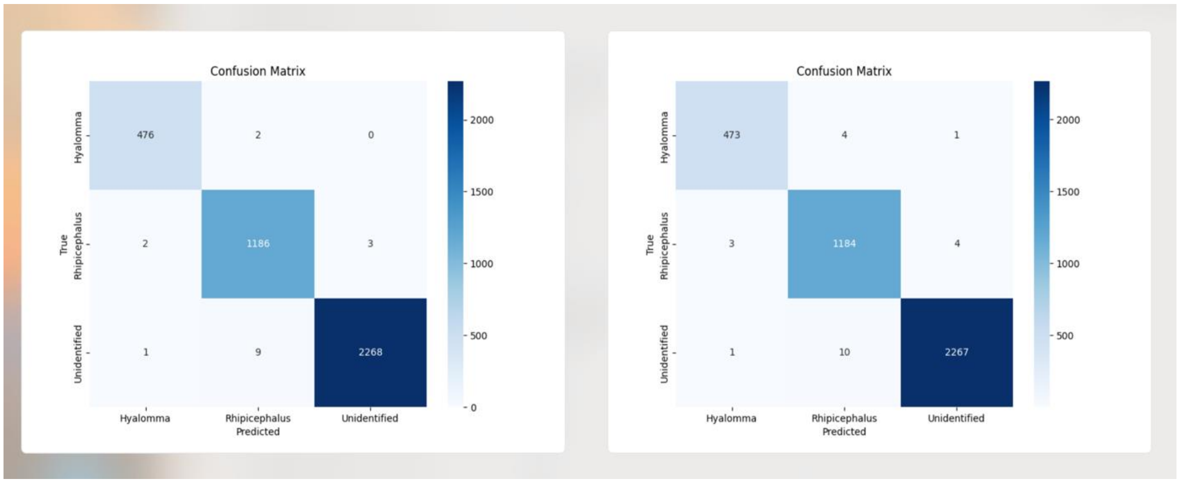


Figure 24: Confusion Matrices for the VGG16 variant A models for 20 epoch runs on the left and 30 epoch runs on the right.

The ROC curves for the VGG16 models can be seen in Figure 25.

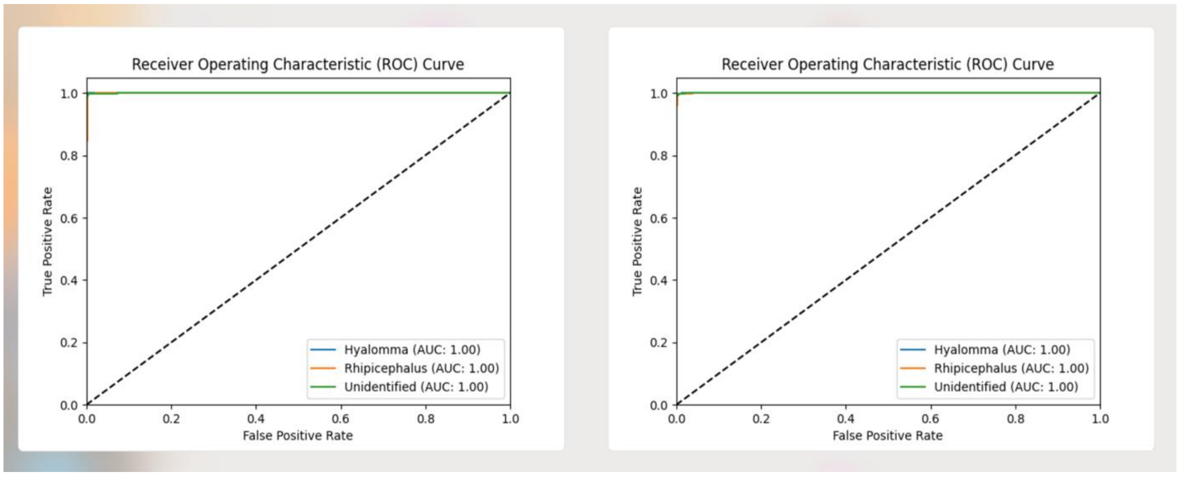


Figure 25: ROC curves for the VGG16 Variant A models for the 20 epoch runs on the left and 30 epoch runs on the right.

This is a model with high stability. It does not seem to make random predictions. The near perfect ROC curve from the beginning to end signifies a nearly perfect model.

4.1.3 ResNet50-VAR A Model

Resnet50 models were large in architecture but produced mid-sized models. The models achieved high accuracies in training sessions and similar scores in testing. The architecture is well-built and has more than enough layers to be able to predict the correct classification. However, the models fell short and could not compete with VGG16 accuracies or losses. The base reason is that when using deeper models, a lot more data is required. ResNet50 would work a lot better if we had 10 times the amount of images that we currently have. Its robustness in being able to predict the correct result should have been unparalleled.

Figure 26 showcases the results of the accuracies during training and validation sessions. It performed well and reached a final accuracy score of 98.98% during the testing phase.

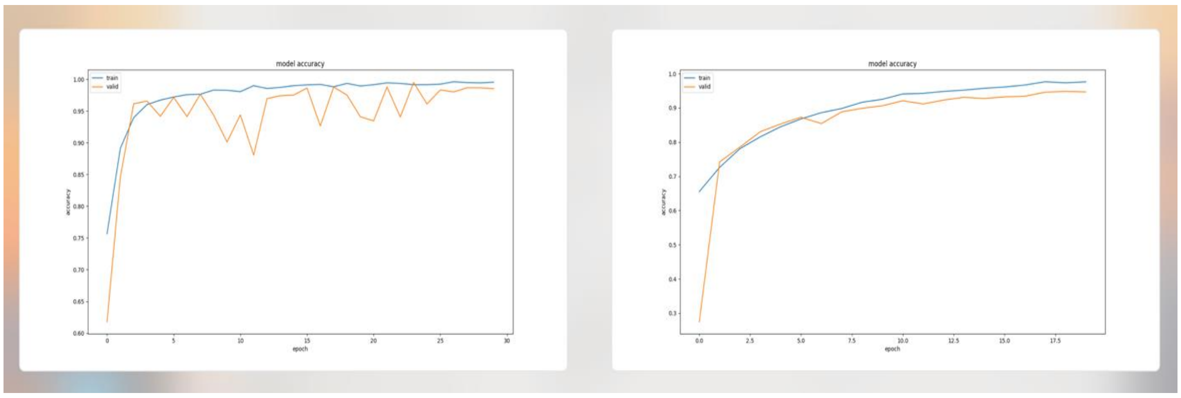


Figure 26: Accuracies for the ResNet50 variant A models for both training and validation sessions. The graph on the left represents the 20 epoch run and the right represents 30 epoch run.

Figure 27 showcases the confusion matrices for the first variant of the ResNet 50 models. Here we can see how the models are predicting different classification from what it's supposed to. ResNet50 models seemed to be hallucinating results even when they had high accuracies.

For the 30 epoch run, the model seemed to be making a lot of mistakes, with 48 misclassifications for Hyalomma, 63 for Rhipicephalus and 19 for Unidentified.

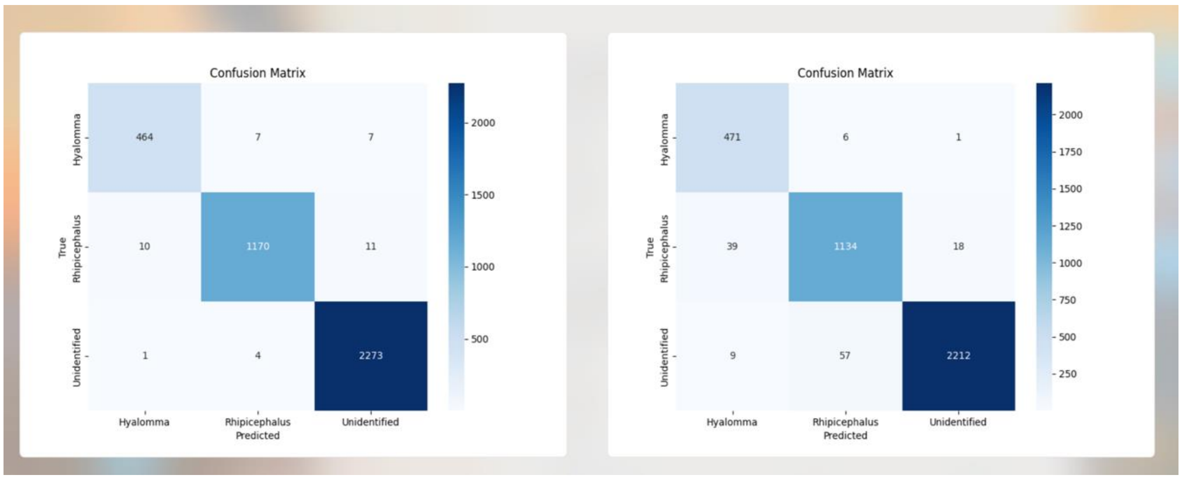


Figure 27: Confusion Matrices for the ResNet50 variant A models for 20 epoch runs on the left and 30 epoch runs on the right.

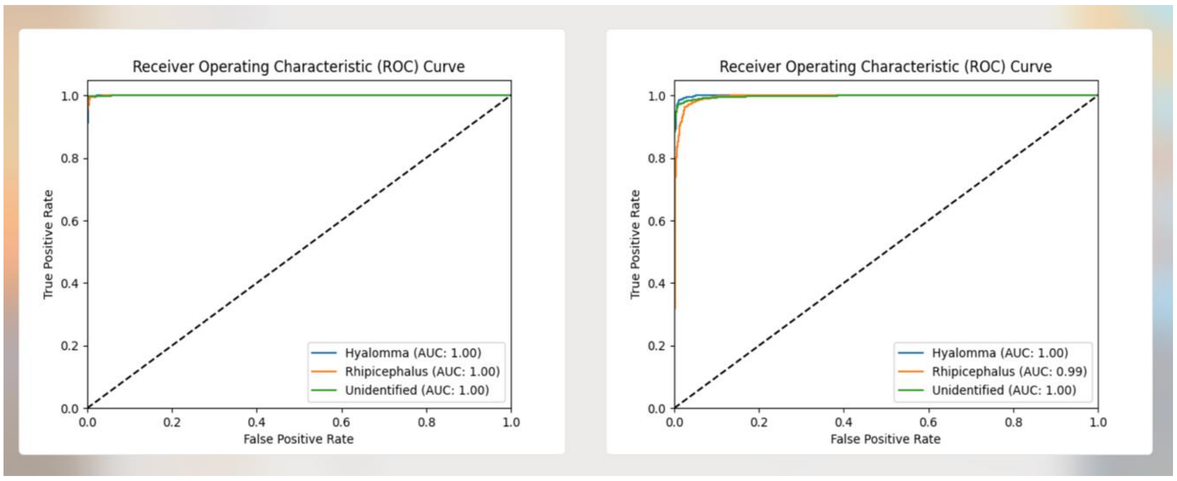


Figure 28: ROC curves for the ResNet Variant A models for the 20 epoch runs on the left and 30 epoch runs on the right.

VGG16 carried this dataset well with 99.57% accuracy and 0.0175 loss for the 30 epoch run. It was stable and produced correct results and had fewer mistakes made. Over the total 3,947 testing set images, it only got 17 wrong for the 20 epoch run and 23 wrong for the 30 epoch run.

4.2 Dataset B

Models created from dataset B produced very interesting results. They incorporated 7 total output classifications which were “HeadShotHyalomma”, “HeadShotRhipicephalus”, “CenterShotHyalomma”, “CenterShotRhipicephalus”, “CornerShotHyalomma”, “CornerShotRhipicephalus” and “Unidentified” in that order. Some results figures had these labels chopped off because they were too long. The models trained pretty slowly due to more parameters in their layers. Results for the accuracies of all the models can be seen in Table 7. VGG16 topped again the charts with an accuracy of 94.05% for the 30 epoch run during the testing phase.

Table 7: Accuracies for training, validation and testing sessions respectively for all models with 20 and 30 epochs using the second dataset.

Model Architecture	Training Accuracy %		Validation Accuracy %		Testing Accuracy %	
	20 epochs	**30 epochs**	**20 epochs**	**30 epochs**	**20 epochs**	**30 epochs**
VGG16	98.58	98.65	98.58	98.64	94.23	94.05
RESNET50	90.35	96.51	90.35	96.51	88.88	91.71
CNN	98.15	99.25	98.15	99.25	88.81	86.79

Table 8: Loss values for training, validation and testing sessions respectively for all models with 20 and 30 epochs using the second dataset.

Model Architecture	Training Loss		Validation Loss		Testing Loss	
	20 epochs	**30 epochs**	**20 epochs**	**30 epochs**	**20 epochs**	**30 epochs**
VGG16	0.0479	0.0433	0.0479	0.0433	0.2144	0.2858
RESNET50	0.3246	0.1183	0.3246	0.1183	0.4351	0.3272
CNN	0.0739	0.0318	0.0739	0.0318	0.4510	0.7019

Table 8, shows the losses of the models for both 20 and 30 epochs. VGG 16 had the lowest losses from the testing phase with a value of 0.2858. This meant that the model was getting close to the global minimum but could not reach the satisficing metric we set for loss. Models were to be 0.05 or lower.

Table 9: Evaluation metrics for all 3 model architectures including precision, recall, F1 score, sentisitivity, specificity and AUC percentages for the 20 epoch run.

Model Architecture	Class	Precision %	Recall %	F1-Score %	Sensitivity %	Specificity %	AUC %
VGG16	HeadShotHyalomma	92.90	96.05	94.44	96.05	99.74	100.00
	HeadShotRhipicephalus	91.30	92.07	91.68	92.07	99.10	100.00
	CenterShotHyalomma	92.73	85.86	89.16	85.86	99.59	100.00
	CenterShotRhipicephalus	86.05	88.00	87.02	88.00	98.21	99.00
	CornerShotHyalomma	83.72	88.52	86.06	88.52	99.15	100.00
	CornerShotRhipicephalus	87.36	87.60	87.48	87.60	97.90	99.00
	Unidentified	99.77	99.14	99.45	99.14	99.76	100.00
RESNET50	HeadShotHyalomma	93.89	95.48	94.68	95.48	99.78	100.00
	HeadShotRhipicephalus	98.61	59.29	74.05	59.29	99.91	99.00
	CenterShotHyalomma	88.96	92.26	90.58	92.26	99.30	100.00
	CenterShotRhipicephalus	82.43	79.13	80.75	79.13	97.89	98.00
	CornerShotHyalomma	87.77	82.38	84.99	82.38	99.43	99.00
	CornerShotRhipicephalus	87.42	74.80	80.62	74.80	98.22	98.00
	Unidentified	89.23	100.00	94.31	100.00	87.19	100.00
CNN	HeadShotHyalomma	82.11	88.14	85.01	88.14	99.32	99.00
	HeadShotRhipicephalus	80.64	89.56	84.87	89.56	97.80	99.00
	CenterShotHyalomma	81.11	88.22	84.52	88.22	98.75	99.00
	CenterShotRhipicephalus	81.31	77.91	79.57	77.91	97.76	97.00
	CornerShotHyalomma	79.17	70.08	74.35	70.08	99.09	98.00
	CornerShotRhipicephalus	79.04	78.61	78.83	78.61	96.55	97.00
	Unidentified	96.99	95.68	96.33	95.68	96.85	99.00

Table 9, shows the evaluation metrics for the dataset B trained models for the 20 epoch run. Unidentified was the most stable classification having accuracies ranging between 89.23% to 99.77%. The models seemed to be able to classify the HeadShots more accurately than any other part of the bug, with accuracies in the 90 percents. The lowest accuracies were for the CenterShots for both Hyalomma and Rhipicephalus. Some CenterShot images were close to CornerShots. The accuracies for these classes were in the lower 80 percents. CornerShots had ranging accuracies of 81% to 87% between all the model architectures.

Table 10: Evaluation metrics for all 3 model architectures including precision, recall, F1 score, sentisitivity, specificity and AUC percentages for the 30 epoch run.

Model Architecture	Class	Precision %	Recall %	F1-Score %	Sensitivity %	Specificity %	AUC %
VGG16	HeadShotHyalomma	91.26	94.35	92.78	94.35	99.68	100.00
	HeadShotRhipicephalus	92.98	91.23	92.10	91.23	99.30	99.00
	CenterShotHyalomma	90.78	89.56	90.17	89.56	99.45	100.00
	CenterShotRhipicephalus	81.35	90.26	85.57	90.26	97.41	99.00
	CornerShotHyalomma	88.03	84.43	86.19	84.43	99.43	99.00
	CornerShotRhipicephalus	89.42	84.06	86.66	84.06	98.35	99.00
	Unidentified	99.59	99.51	99.55	99.51	99.56	100.00
RESNET50	HeadShotHyalomma	92.67	78.53	85.02	78.53	99.78	100.00
	HeadShotRhipicephalus	87.63	90.19	88.89	90.19	98.70	99.00
	CenterShotHyalomma	89.71	82.15	85.76	82.15	99.42	100.00
	CenterShotRhipicephalus	85.79	81.91	83.81	81.91	98.30	98.00
	CornerShotHyalomma	90.12	59.84	71.92	59.84	99.67	99.00
	CornerShotRhipicephalus	75.73	88.83	81.76	88.83	95.28	98.00
	Unidentified	99.07	99.77	99.42	99.77	99.00	100.00
CNN	HeadShotHyalomma	82.68	83.62	83.15	83.62	99.38	99.00
	HeadShotRhipicephalus	82.96	84.34	83.64	84.34	98.23	98.00
	CenterShotHyalomma	74.32	82.83	78.34	82.82	98.25	99.00
	CenterShotRhipicephalus	73.38	78.61	75.90	78.61	96.43	97.00
	CornerShotHyalomma	75.78	69.26	72.38	69.26	98.90	98.00
	CornerShotRhipicephalus	72.16	84.06	77.66	84.06	94.63	96.00
	Unidentified	98.91	92.03	95.34	92.03	98.92	99.00

Table 10 lists the evaluation metrics for the 30 epoch run. The perfomance is a little less than the 20 epoch run. Only Unidentified performed the best in this set, reaching accuracies between 98.91% to 99.59%. HeadShots performed less well with accuracies ranging between 82% to 92% between the different architectures. CenterShots had accuracies ranging from 73% to 90%. Lastly, CornerShots performed the least well with accuracies ranging from 72% to 90%.

4.2.1 CNN-VAR B Model

Models produced from the basic CNN architectures had low performance in comparison to the first dataset. Models could not reach 90% accuracies during the testing phase. The highest achievable accuracy was 88.81% from the 20 epochs run. We deduced this was because of not having enough data for all the new subclasses.

This is the second variant of the CNN model that utilised dataset B. The accuracies can be seen for both training and validation sessions for the 20 and 30 epoch runs. The models would reach high values of accuracies during training and validation but less in testing. We deduce that the model might be overfitting to those two sets and producing poor accuracy for the testing set.

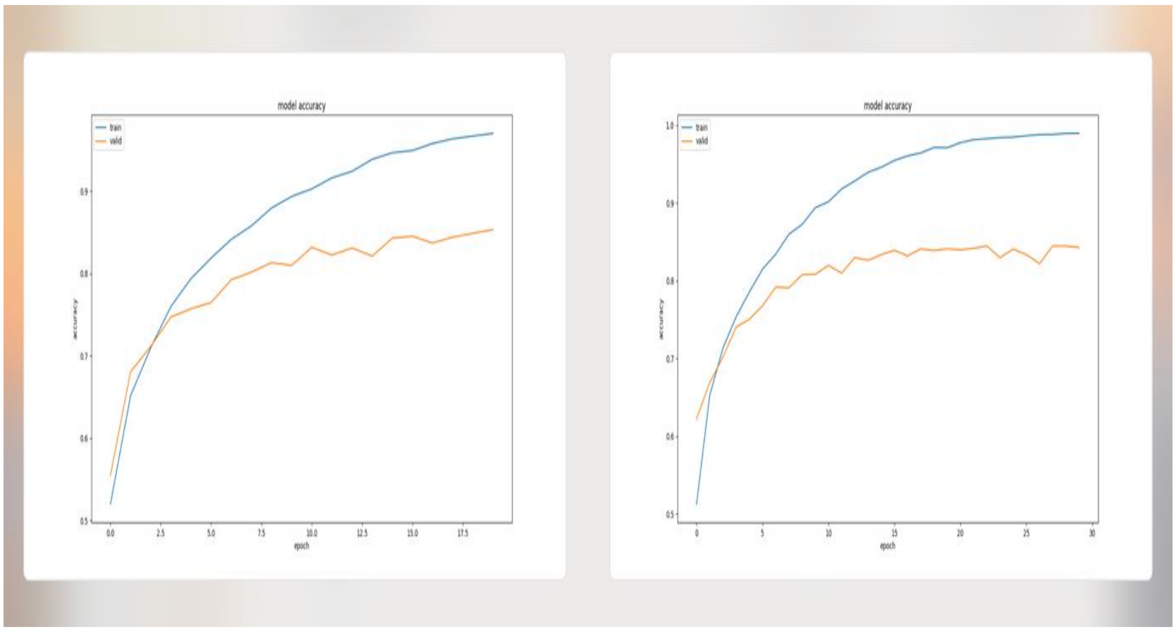


Figure 29: Accuracies for the CNN variant B models for both training and validation sessions. The graph on the left represents the 20 epoch run and the right represents 30 epoch run.

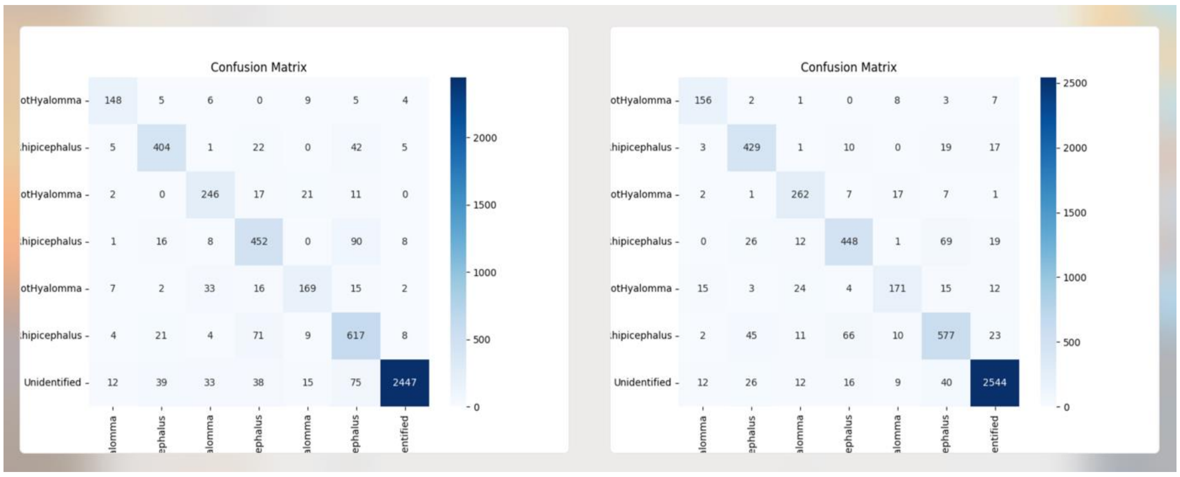


Figure 30: Confusion Matrices for the CNN variant B models for 20 epoch runs on the left and 30 epoch runs on the right.

The confusion matrices from Figure 30, showcase how many images the models got correct and ones they mislabeled. CNN produced inaccurate results for most of the images in the test set, but upto 148 incorrect labels for the CornerShotRhipicephalus class, Further improvements are needed for this model. It is over hallucinating results in each classification label.

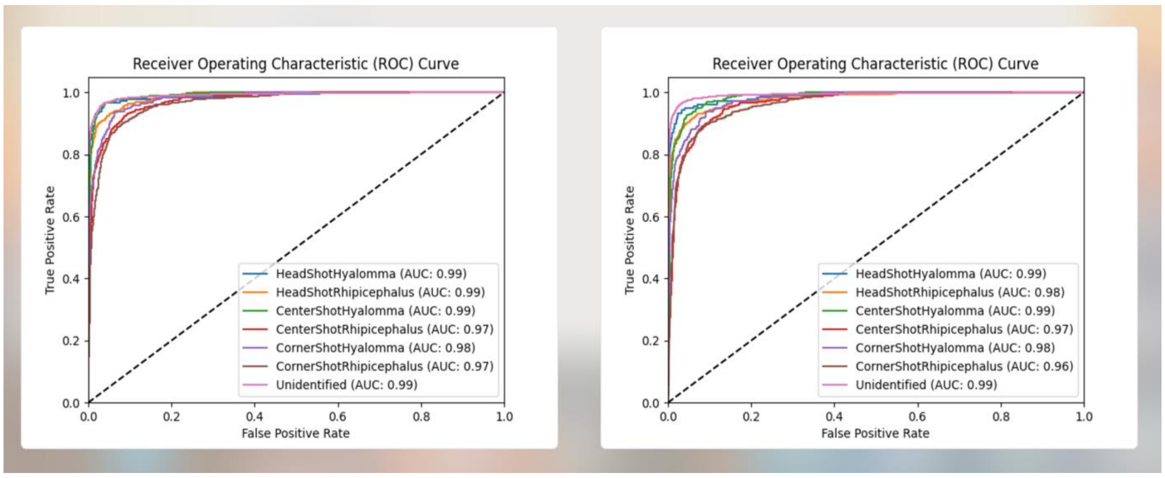


Figure 31: ROC curves for the CNN Variant B models for the 20 epoch runs on the left and 30 epoch runs on the right.

The ROC curve for this model isn’t as linear as the first dataset. It may have moments where it’s not predicting the correct values and making random predictions.

4.2.2 VGG16-VAR B Model

VGG16 tried again to be robust to the changes of dataset being able to reach very high values of accuracy as well as low values of training and testing loss. The models would train a lot longer, almost reaching 1.5 hours for the 30 epoch run. The difference in accuracy between the 2 epoch sizes of 20 and 30 was still minimal, only increasing about 0.18%

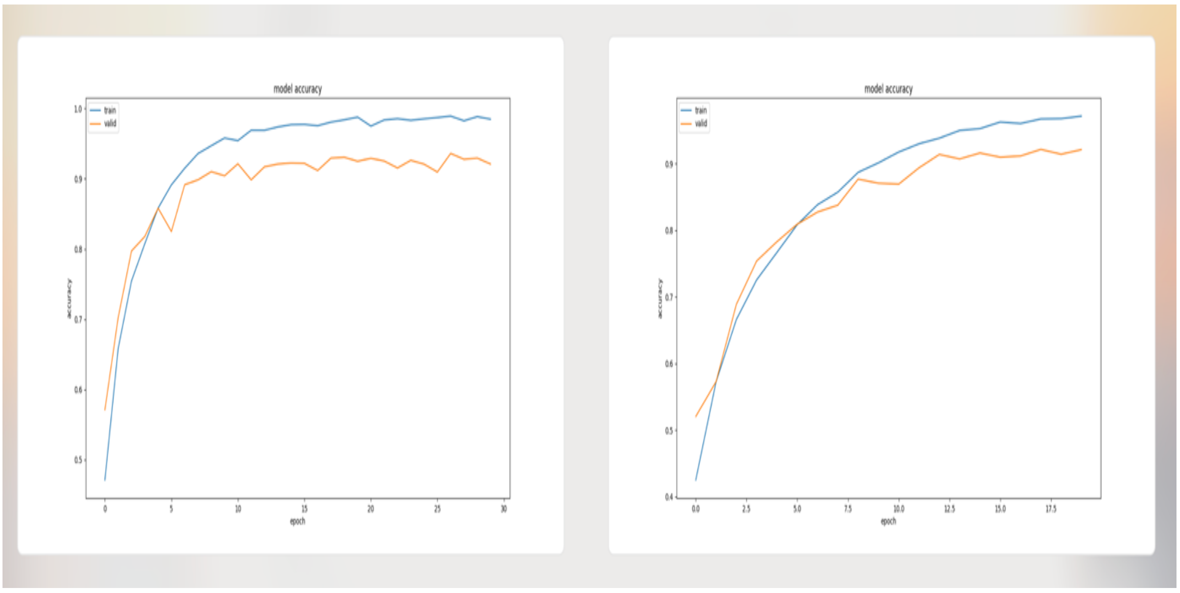


Figure 32: Accuracies for the VGG16 variant B models for both training and validation sessions. The graph on the left represents the 20 epoch run and the right represents 30 epoch run.

From Table 8, we can see that the 20 epoch count outperforms the 30 epoch run model by a small margin of 0.18% for the accuracies during training and validation sessions. The VGG16 models performed outstandingly again and produced accurate results almost all the time.

Figure 33 showcases confusion matrices for both 20 and 30 epoch runs Here we can observe, how the model is performing on the new classification keys. Surprisingly, the 20 epochs run model had more incorrect values, even though it had higher accuracy. The models seemed to be producing the most incorrect values for the Rhipicephalus subclasses, with up to 119 incorrect label for CenterShotRhipicephalus class for the 20 epoch runs model. Fewer mistakes were made for the HeadShot classes for both Hyalomma and Rhipicephalus as well as Unidentified

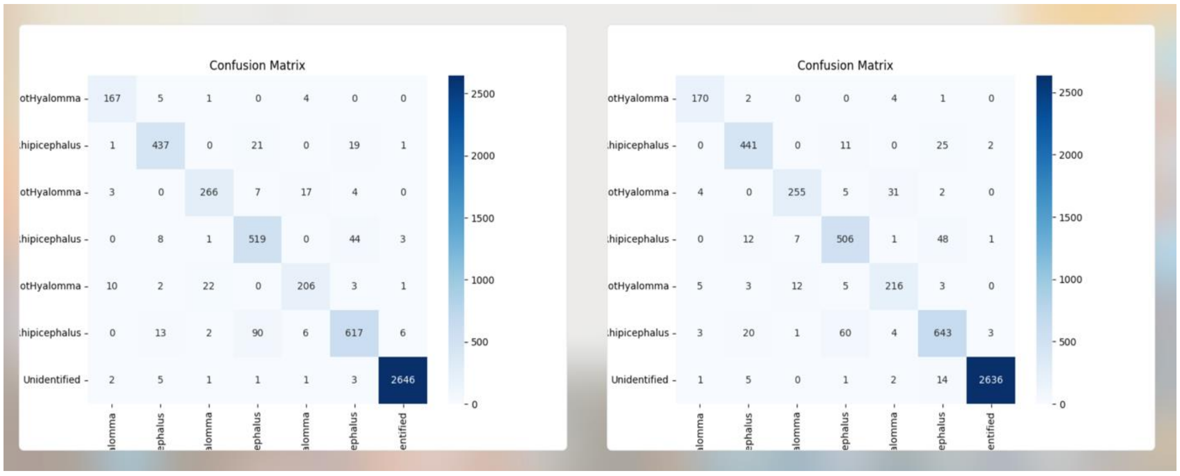


Figure 33: Confusion Matrices for the VGG16 variant B models for 20 epoch runs on the left and 30 epoch runs on the right.

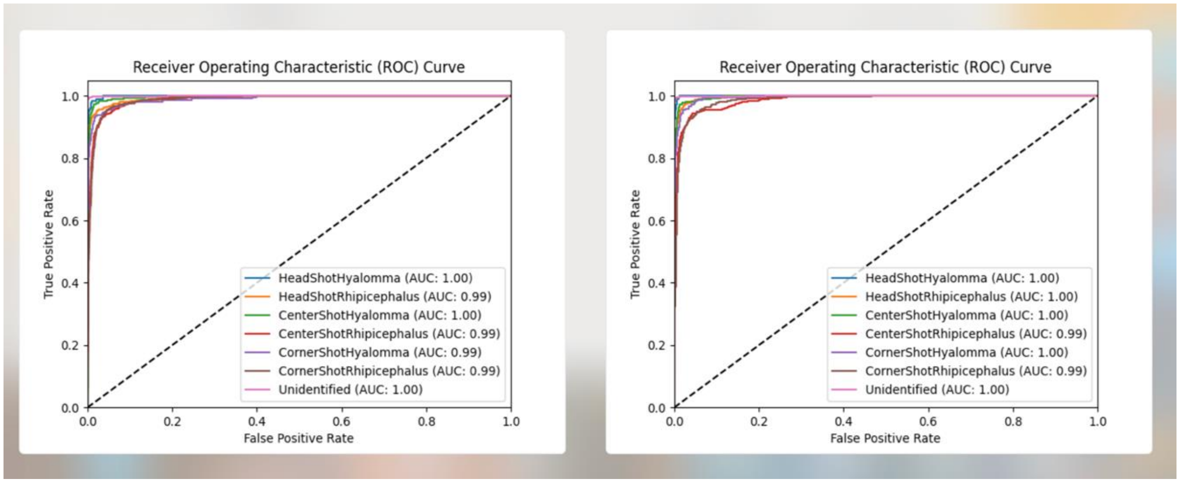


Figure 34: ROC curves for the VGG16 Variant B models for the 20 epoch runs on the left and 30 epoch runs on the right.

4.2.3 ResNet50-VAR B Model

Resnet50 models for variant B attempted to reach 95% accuracies, but fell short with only 91.71% for the 30 epoch run. We deduced again that the data count for the subclasses made this architecture ineffective.

Figure 35 showcases the results of the accuracies during training and validation sessions. It performed less than the first dataset and it had it’s accuracies regressing during the first epoch runs. Final values of accuracies were 88% and 91.71% for validation and training sessions respectively.

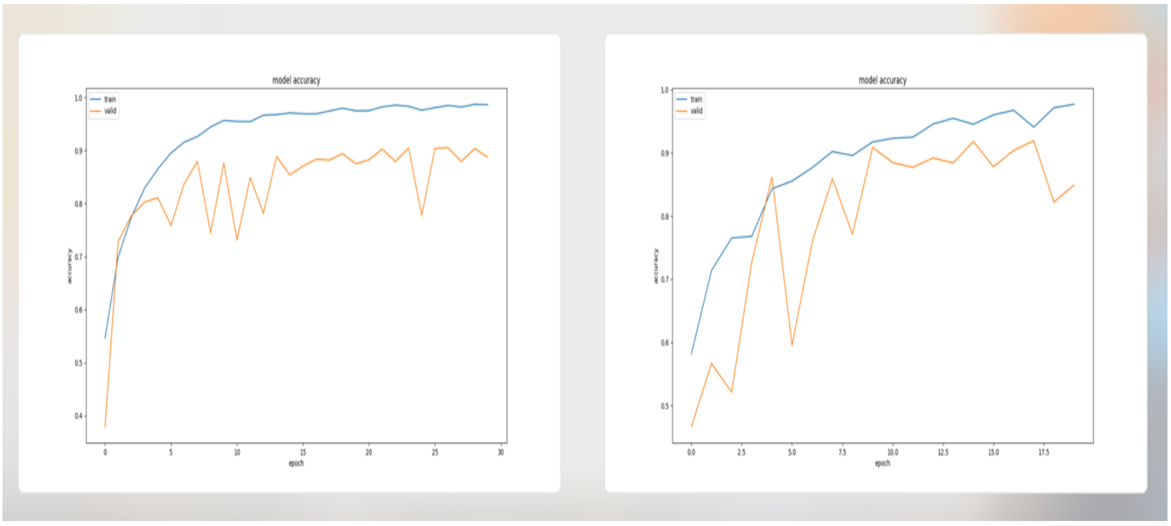
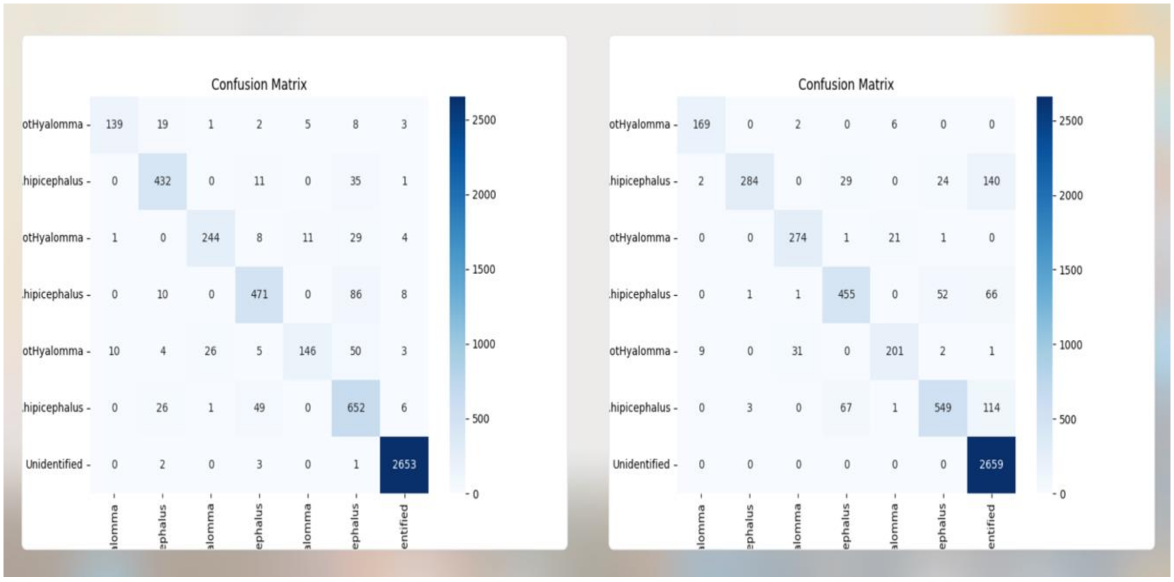


Figure 35: Accuracies for the ResNet variant B models for both training and validation sessions. The graph on the left represents the 20 epoch run and the right represents 30 epoch run.

Figure 36 showcases the confusion matrices for the second variant of the ResNet 50 models. Here we can see how the models are predicting results with a completely different pattern from the other models. The 30 epoch run model had a perfect score not predicting unidentified for all the other classes. The model seemed more stable at predicting the HeadShotHyalomma class with only 11 misclassifications and 4 wrong for HeadShotRhipicephalus.


Confusion Matrix

	alomma	ephalus	alomma	ephalus	alomma	ephalus	entified
otHyalomma	139	19	1	2	5	8	3
hipicephalus	0	432	0	11	0	35	1
otHyalomma	1	0	244	8	11	29	4
hipicephalus	0	10	0	471	0	86	8
otHyalomma	10	4	26	5	146	50	3
hipicephalus	0	26	1	49	0	652	6
Unidentified	0	2	0	3	0	1	2653

Confusion Matrix

	alomma	ephalus	alomma	ephalus	alomma	ephalus	entified
otHyalomma	169	0	2	0	6	0	0
hipicephalus	2	284	0	29	0	24	140
otHyalomma	0	0	274	1	21	1	0
hipicephalus	0	1	1	455	0	52	66
otHyalomma	9	0	31	0	201	2	1
hipicephalus	0	3	0	67	1	549	114
Unidentified	0	0	0	0	0	0	2659

Figure 36: Confusion Matrices for the ResNet variant B models for 20 epoch runs on the left and 30 epoch runs on the right.

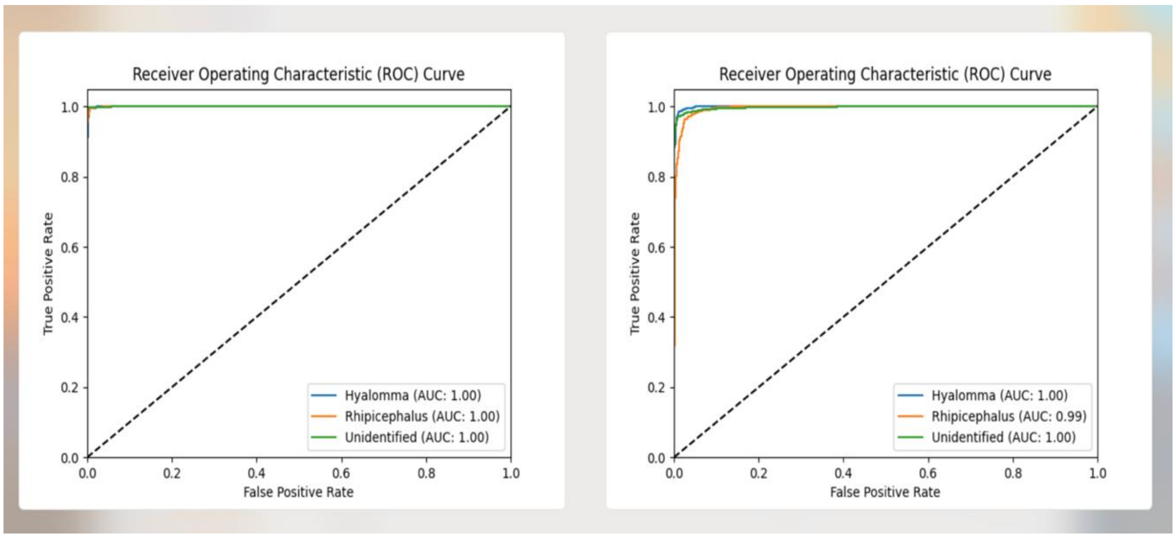


Figure 37: ROC curves for the VGG16 Variant B models for the 20 epoch runs on the left and 30 epoch runs on the right.

Models from dataset B performed less than dataset A. However, interestingly, the ResNet 50 seems to be giving hope of performing better in future trials, as it had less errors when it came to predicting HeadShots of the bugs. Perhaps, going though and segmenting the portions of the bugs tightly, might help the models perform better.

Overall, the models performed well and there was a lot to deduce from the training and testing sessions. The optimising metric being accuracy is how we find the best models to be able to represent this study and be used on the webUI. Performant and accurate models are required for user end use to reduce the likeliness of giving a false result. For the first dataset, A. The contender was the VGG16 model. It’s 99.57% accuracy was unparalled with fewer mistakes made during testing in comparison to ResNet50 and the basic CNN model.

4.4 WebUI Results

The web app also showed results of what was predicted in a GUI form. The image, along with the predicted class was highlighted. On top of this, an inner join to the model table pulled metadata associated with which model predicted the image’s class. Figure 38, 39 and 40 show the results of the classifications.

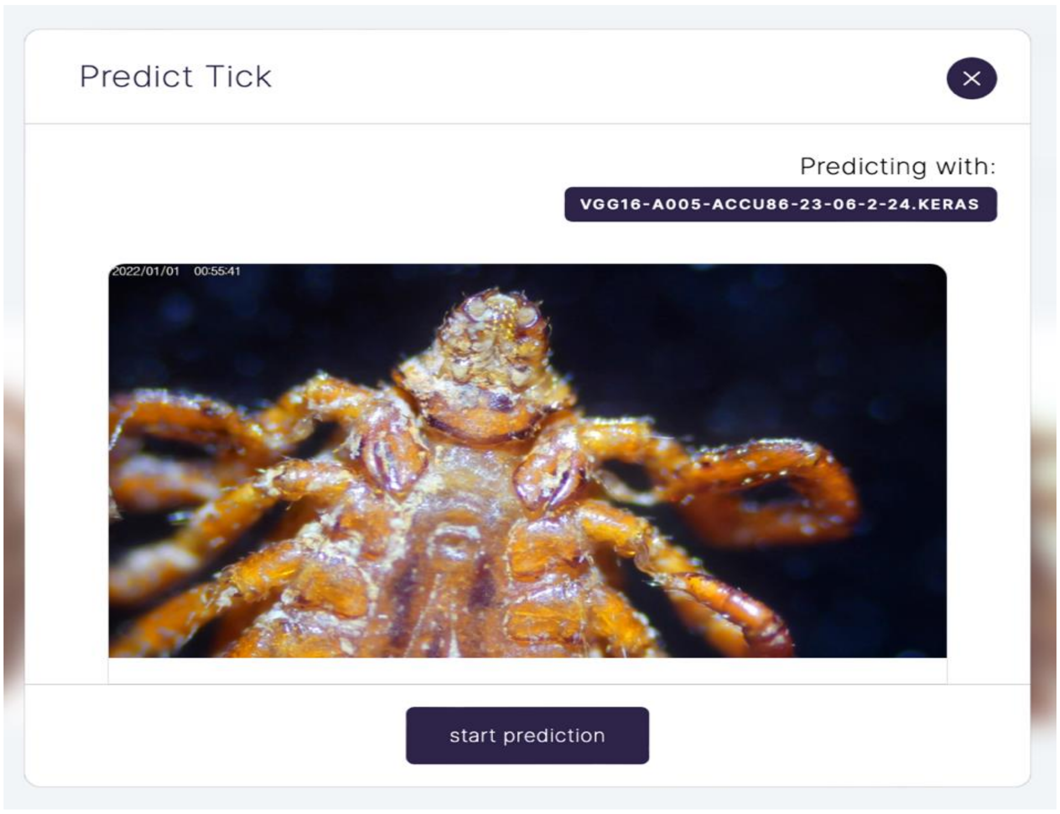
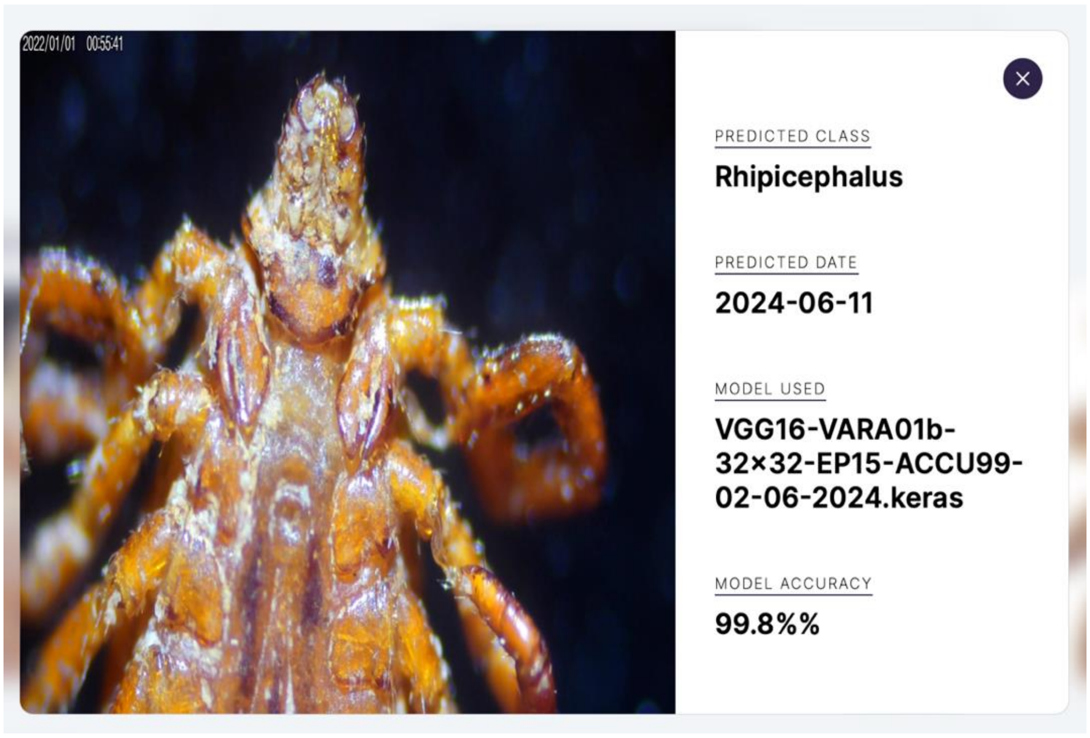


Figure 38: UI of uploading and image of a tick.



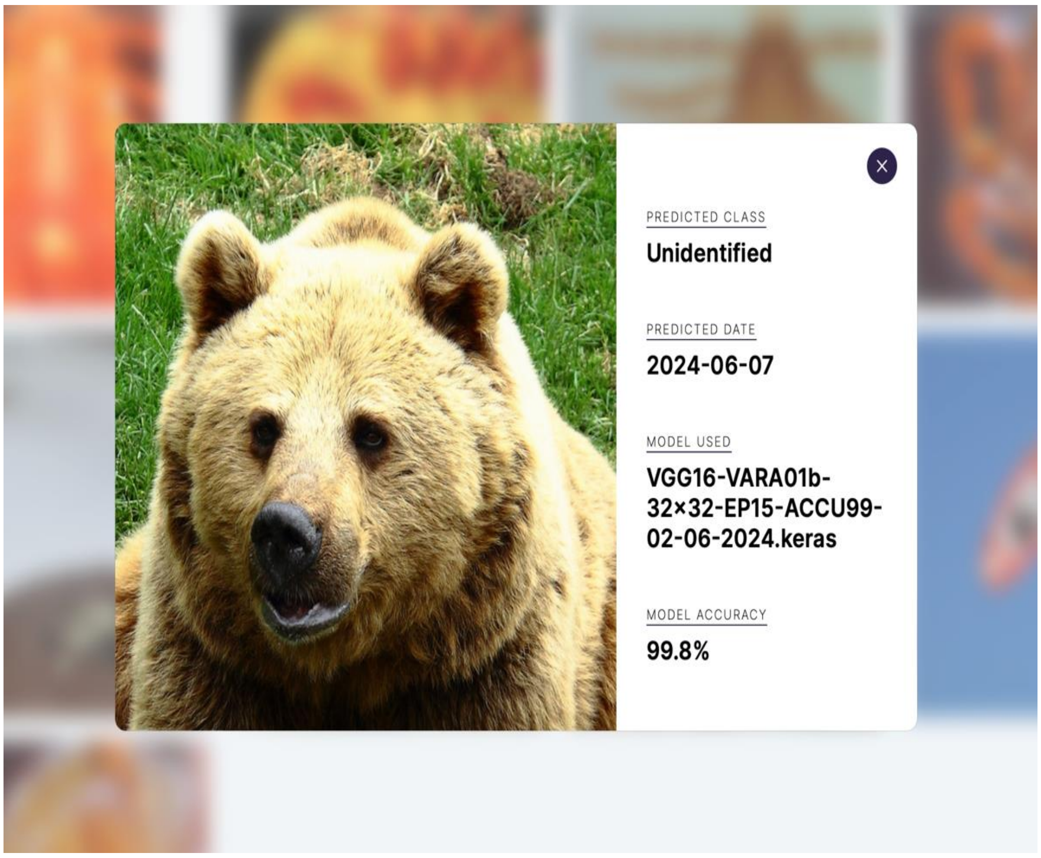
PREDICTED CLASS
Rhipicephalus

PREDICTED DATE
2024-06-11

MODEL USED
VGG16-VARA01b-32×32-EP15-ACCU99-02-06-2024.keras

MODEL ACCURACY
99.8%%

Figure 39: Classification of the uploaded image as Rhipicephalus.



PREDICTED CLASS

Unidentified

PREDICTED DATE

2024-06-07

MODEL USED

VGG16-VARA01b-32×32-EP15-ACCU99-02-06-2024.keras

MODEL ACCURACY

99.8%

Figure 40: : Classification of an image of a bear classified as Unidentified.

CHAPTER V

Discussion

5.1 Related Works

From this study, Table 11, shows the comparisons with the previous literature comparing accuracies across the different model structures. This shows a realistic comparison of detecting tick bugs and which model architecture performed better.

The models from omodor et al 2021 built using ResNet50 as well as a custom built CNN model produced training accuracies of 99.7% and a final mean prediction accuracy of 80%. From this paper, the ResNet50 model produced a training accuracy of 98.88% accuracy during training and a 98.98% accuracy overall during testing. This gives us an 18.98% increase in ResNet50 accuracy for predicting tick bugs.

From Luo et Al, Identification of tick species, the resulting accuracy from a basic CNN model was 99.5%. We managed to reach 99.23% using 30 epochs for training and a stable 95.71% overall accuracy from testing.

Table 11: Comparison of studies along with their respective accuracies for classifying tick bugs.

Models	Accuracy %	Reference
VGG16	99.57	This Study
ResNet50	98.98	
CNN Model	95.71	
Random Forest (RF)	97.00	Pérez-Otáñez et al., 2024
ResNet-50, custom-built CNN	80.00	Omodior et al., 2021
TickIDNet (CNN Model)	87.80	Justen et al., 2021
CNN Model	92.00	Akbarian et al., 2020
CNN Model	99.50	Luo et al., 2022

5.2 Discussion

The results from this study show us that it is ideal to use deep learning models to be able to predict the class of a tick bug. We predicted 2 types of hard ticks, Hyalomma and Rhipicephalus. However, all the models are able to predict a third classification of Unidentified for any image given that is neither of the two. From the various studies, it shows that deep learning models are capable of extracting features from the images of the ticks and produce a cost function with weights and biases that are able to predict the correct class. This form of taxonomy can be able to help scientists in laboratories be able to classify bugs, especially if they are dangerous, in a very quick and efficient manner. Knowledge of features of these bugs become irrelevant. A scientist would only need to have an image of the bug and the model does the rest.

Being able to have this scientific tool in the form of a mobile application can help identify ticks in the wild, especially in a place where microscopes may not be available. (Pérez-Otáñez et al., 2024), we can see the distribution of bugs being in thriving and harsh areas. Being able to classify bugs where they are found in a quick and efficient manner, helps find results and take action quicker. By pushing average accuracies from 80% to 99.57% from the VGG16 model in this study. It will help scientists, farmers and vets classify ticks in a more orderly fashion with high accuracies and confidence.

CHAPTER VI

Conclusion And Recommendations

The research was very fruitful in digging down the architectures of the model to reveal the gemstone that shone when predicting the bugs. Out of the three models, VGG16 proved itself the contender in classifying the correct key value for the image. The model was very performant and could analyse hundred or thousands of images in a matter of seconds. It topped the charts with an astounding 99.57% accuracy. ResNet50 came in second with 98.98% accuracy and lastly the custom basic CNN model with an accuracy value of 95.71%. The models were trained with around 26,000 images. If the count was increased between 50,000 to 100,000. Surely, this model could improve the accuracies and keep them ranging over 99%.

6.1 Future Improvements and Ideas

- The models had really good accuracies, and some even tiering at 100%. However, there are some edge case scenarios that could throw a model off its weight. The models could improve by being fed random data from multiple data sets of bugs that resemble ticks but are not ticks to further expand the realm of unidentified objects.

- More data could be added overall to improve accuracy and stabilize the weights and biases of the model.

- Future models could utilize transfer learning by using a dataset of anything that contains 100,000 to 1,000,0000 images, train it with the feature set size required, then only exporting the weights. The weights could then be re imported and used with the bug images. Surely this task of training a million images could take a long time and might produce a very large model especially for the VGG16 model.

- Access to faster servers with more resources could help us train larger networks with deeper layers to be able to unlock new features of the bugs that we may not know off. There are deep learning models that have over 200 layers like the DenseNet264. This will greatly improve accuracy. With fast GPU cores and a lot of RAM, training a network like this would be a breeze.

- Expand the realm of tick taxonomy to more types like, Dermacentor and Ixodes.

- Use the models trained as bases, extract the weight and transfer learn for other microscopic classification like bacteria and amoeba.

- Make a phone application that can take a low resolution zoomed in image of a tick bug to predict its class.

- Use GANs (Zhang et al., 2023) to be able to pit models against each other to improve their accuracies. This method could have a new model that created tick bugs out of thin air and feed them to our models for prediction. From there, the models would improve accuracy and make the model that generates the bugs better. Eventually, the models will be good enough to not distinguish between a generated bug and a real one, giving high accuracies for predicting the classes and an almost infinite amount of data.

6.2 Conclusion

Overall, the study showcased how possible it is to classify the different types of ticks using Deep Learning models. This approach would not have been possible using base Machine Learning models. Deep Learning models are able to extract various features like edges and patterns of these bugs that ML models just couldn't. This will help future iterations of these models to achieve one that is able to predict images that are blurry, zoomed in and not as crystal clear as microscopic ones.

Each pandemic in the past had unique causes. However, they all underscored and underestimated the importance of public health measures and disease prevention. Artificial Intelligence methods offer a unique standpoint, at offering solutions with high accuracies and reliable result as well as unparalleled speed in comparison to human aided solutions. Being able to predict tick bugs at a glance is something that would take a human, years of scientific research to master and predict, but only a couple of hours for an AI method to learn and seconds to predict. This in turn can help scientists bypass years of unnecessary learning and pass the work to a model. In addition to this, the model can be used by regular people who are closely surrounded by tick bugs, such as farmers with herds of cows to predict if the bugs on cows are dangerous ticks, or dog owners who might be living with tick infested pets.

These historical pandemics shaped human history and our understanding of epidemiology, sparing many future lives. The outbreaks demonstrate the necessity of medical research and preparedness at all levels, including detections, innovation and prevention in the field of public health. The success and popularity of these artificial models, not just in the health sector but in other domains are helping humans further scientific advances for the welfare of humanity.

References

Akbarian, S., Nelder, M. P., Russell, C. B., Cawston, T., Moreno, L., Patel, S. N., Allen, V. G., & Dolatabadi, E. (2022). A Computer Vision Approach to Identifying Ticks Related to Lyme Disease. *IEEE Journal of Translational Engineering in Health and Medicine, 10*. https://doi.org/10.1109/JTEHM.2021.3137956

Amarathunga, D. C., Grundy, J., Parry, H., & Dorin, A. (2021). Methods of insect image capture and classification: A Systematic literature review. In *Smart Agricultural Technology* (Vol. 1). https://doi.org/10.1016/j.atech.2021.100023

Badirli, S., Picard, C. J., Mohler, G., Richert, F., Akata, Z., & Dundar, M. (2023). Classifying the unknown: Insect identification with deep hierarchical Bayesian learning. *Methods in Ecology and Evolution, 14*(6). https://doi.org/10.1111/2041-210X.14104

Christensen, C. (2015). Review of Alan Turing: His Work and Impact Edited by S. Barry Cooper and Jan van Leeuwen . *Cryptologia, 39*(2). https://doi.org/10.1080/01611194.2015.1009754

Estrada-Peña, A. (1999). Geostatistics and remote sensing using NOAA-AVHRR satellite imagery as predictive tools in tick distribution and habitat suitability estimations for Boophilus microplus (Acari: Ixodidae) in South America. *Veterinary Parasitology, 81*(1). https://doi.org/10.1016/S0304-4017(98)00238-6

Floréen, Patrik., Krüger, Antonio., Spasojevic, Mirjana., Liu, L., & Ratti, C. (2010). Pervasive computing : 8th international conference, Pervasive 2010, Helsinki, Finland, May 17-20, 2010 : proceedings. In *MIT web domain*.

Gill, K. S., Anand, V., Gupta, R., & Pahwa, V. (2023). Insect Classification using Deep Convolutional Neural Networks and Transfer Learning on MobileNetV3 Model. *2023 World Conference on Communication and Computing, WCONF 2023*. https://doi.org/10.1109/WCONF58270.2023.10235196

Justen, L., Carlsmith, D., Paskewitz, S. M., Bartholomay, L. C., & Bron, G. M. (2021). Identification of public submitted tick images: A neural network approach. *PLoS ONE, 16*(12 December). https://doi.org/10.1371/journal.pone.0260622

Kasinathan, T., Singaraju, D., & Uyyala, S. R. (2021). Insect classification and detection in field crops using modern machine learning techniques. *Information Processing in Agriculture, 8*(3). https://doi.org/10.1016/j.inpa.2020.09.006

Lahijany, G. M., Ohrndorf, M., Zenkert, J., Fathi, M., & Kelte, U. (2021). IdentiBug: Model-Driven Visualization of Bug Reports by Extracting Class Diagram Excerpts. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*. https://doi.org/10.1109/SMC52423.2021.9658989

Luo, C., Pearson, P., Xu, G., & Rich, S. M. (2022). A Computer Vision-Based Approach for Tick Identification Using Deep Learning Models. *Insects, 13*(2). https://doi.org/10.3390/insects13020116

Mahesh, B. (2020). Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet], 9*(1).

Malik, P., Singh, A., Gorai, C., Jha, I., & Paul, S. (2023). Insect Classification Using Pretrained Deep Neural Networks and Transfer Learning. *Conference Proceedings - 2023 IEEE Silchar Subsection Conference, SILCON 2023*. https://doi.org/10.1109/SILCON59133.2023.10404367

Nuttall, P. A. (2022). Climate change impacts on ticks and tick-borne infections. In *Biologia* (Vol. 77, Issue 6). https://doi.org/10.1007/s11756-021-00927-2

Omodior, O., Saeedpour-Parizi, M. R., Rahman, M. K., Azad, A., & Clay, K. (2021). Using convolutional neural networks for tick image recognition – a preliminary exploration.

Experimental and Applied Acarology, *84*(3). https://doi.org/10.1007/s10493-021-00639-x

Pérez-Otáñez, X., Rodríguez-Hidalgo, R., Enríquez, S., Celi-Erazo, M., Benítez, W., Saegerman, C., Vaca-Moyano, F., Ron-Garrido, L., & Vanwambeke, S. O. (2024). High-resolution prediction models for Rhipicephalus microplus and Amblyomma cajennense s.l. ticks affecting cattle and their spatial distribution in continental Ecuador using bioclimatic factors. *Experimental and Applied Acarology*, *92*(3). https://doi.org/10.1007/s10493-023-00883-3

Roberts, L. Gi. (1965). Lawrence Roberts Machine Perception of Three Dimensional Solids. In *PhD Thesis*.

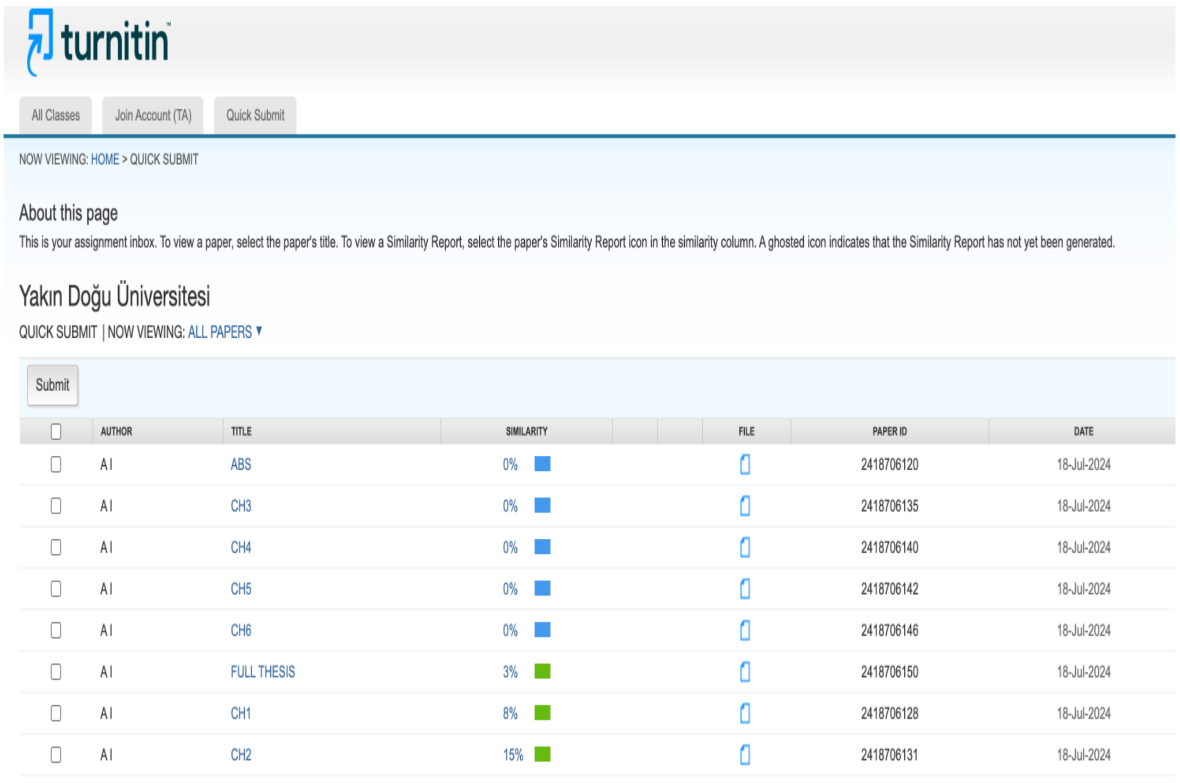
Teixeira, A. C., Ribeiro, J., Morais, R., Sousa, J. J., & Cunha, A. (2023). A Systematic Review on Automatic Insect Detection Using Deep Learning. In *Agriculture (Switzerland)* (Vol. 13, Issue 3). https://doi.org/10.3390/agriculture13030713

Yang, H., Liu, W., Xing, K., Qiao, J., Wang, X., Gao, L., & Shen, Z. (2010). Research on insect identification based on pattern recognition technology. *Proceedings - 2010 6th International Conference on Natural Computation, ICNC 2010*, *2*. https://doi.org/10.1109/ICNC.2010.5583156

Zhang, S., Qian, Z., Huang, K., Zhang, R., Xiao, J., He, Y., & Lu, C. (2023). Robust generative adversarial network. *Machine Learning*, *112*(12). https://doi.org/10.1007/s10994-023-06367-0

Appendix X

Similarity Report



All Classes | Join Account (TA) | Quick Submit

NOW VIEWING: HOME > QUICK SUBMIT

About this page

This is your assignment inbox. To view a paper, select the paper's title. To view a Similarity Report, select the paper's Similarity Report icon in the similarity column. A ghosted icon indicates that the Similarity Report has not yet been generated.

Yakın Doğu Üniversitesi

QUICK SUBMIT | NOW VIEWING: ALL PAPERS ▼

Submit

AUTHOR	TITLE	SIMILARITY		FILE	PAPER ID	DATE
A I	ABS	0%	🟦		2418706120	18-Jul-2024
A I	CH3	0%	🟦		2418706135	18-Jul-2024
A I	CH4	0%	🟦		2418706140	18-Jul-2024
A I	CH5	0%	🟦		2418706142	18-Jul-2024
A I	CH6	0%	🟦		2418706146	18-Jul-2024
A I	FULL THESIS	3%	🟩		2418706150	18-Jul-2024
A I	CH1	8%	🟩		2418706128	18-Jul-2024
A I	CH2	15%	🟩		2418706131	18-Jul-2024