**Literature Review**

**On- Device Machine Learning**

S. Dhar, J. Guo, J. (Jason) Liu, S. Tripathi, U. Kurup, and M. Shah, ‘A Survey of On-Device Machine Learning: An Algorithms and Learning Theory Perspective’, *ACM Trans. Internet Things*, vol. 2, no. 3, p. 15:1-15:49, Jul. 2021, doi: [10.1145/3450494](https://doi.org/10.1145/3450494).

Inference – a model which is trained in the cloud and executed on the device

The biggest constraint with having the client handle all the operations of Machine Learning is the device’s resources. This article aims to survey the current situation of Machine Learning on edge devices. It highlights how the present scenario is to resort to a cloud-based system where the edge device will collect data and conduct inference, send the data to the cloud, where the model is trained and re-trained with new data from the devices. It mainly discusses the notion of having the edge devices conduct the training of the model and no cloud implementation at all for handling the model. Whilst this is an alternative approach for implementing the FAIMS application system, which solves the issue of using machine learning to classify images whilst offline, it also has some cases that need consideration.

Cloud-based systems also have issues regarding security and privacy surrounding users’ data. Data loss, compromised cloud network.

Deep Neural Networks like CNN are the ideal algorithm for image classification as they have high accuracy for classifying the image. Since classification accuracy is a focus for the FAIMS system, a CNN algorithm is desirable. However, deploying CNN models would not be ideal due to edge device resource constraints. Training of the model needs MBs or GBs of memory for the weights, activations, gradients, and data batches, making it very difficult to be deployed and train on a mobile device. Thus, having a purely on-device image classification system would not be the best approach.

Requiring the user’s device to connect to the server to transmit data that may be stored raises concerns surrounding data security and whether a user can use the client system in an offline environment which is an essential requirement for this project. The connection could be interfered with and result in data loss, or the user has to rely upon the ML service securely storing their data so that it doesn’t become compromised by an unwanted party \cite{dhar\_survey\_2021}. This typical architecture, whilst prominent in the current environment for on-device ML, does not mean that it is flawless in design. Research suggests that on-device systems incorporating the cloud lose a sense of uniqueness\cite{dhar\_survey\_2021}. Since these systems re-train the model with data from various users, it will not provide a tailored experience to a singular user’s data. Alternatively, a pure client-side design would excel in comparison.

This design aims to provide an approach where the computational power of cloud resources can be used for training and re-training the model and have the mobile device load the model to conduct inference locally \cite{plieninger\_deep\_2016}. Further exploration into this design suggests that a lot of research incorporates this method due to its nature of offloading calculations from the cloud, allowing offline ML \cite{}. This method is not free from design concerns like the other presented designs. It shares the same issues raised for the exclusively server-side approach, costs, data security and usability. Additionally, the system must ensure that a user’s device has enough storage to load the model onto the device, which could be substantial in size \cite{dhar\_survey\_2021}. The ability of this design to provide multiple users ML solutions regardless of their device’s hardware in either online or offline fashion makes this solution accessible.

V. Ganesan, ‘Machine Learning in Mobile Applications’, *International Journal of Computer Science and Mobile Computing*, vol. 11, pp. 110–118, Feb. 2022, doi: [10.47760/ijcsmc.2022.v11i02.013](https://doi.org/10.47760/ijcsmc.2022.v11i02.013).

Implementing machine learning in mobile applications is made more straightforward due to the variety of frameworks in the current environment. TensorFlow, Keras, Scikit, Pytorch, ML Kit, Core ML, IBM Watson, Microsoft Azure Cognitive Services, Amazon Web services, Google Cloud Machine Learning. They aim to simplify the process of creating and implementing machine learning models as they remove the need to understand the underlying principles thoroughly. Each of these frameworks varies in capabilities; it provides many options for implementing a system into any application.

Due to the unique nature of the FAIMS application being a web application that is generated into native applications for both IOS and Android, selecting the tools that effectively cater to the already existing design is desirable.

The machine learning process for the edge device will initially extract features, where information from the data is taken to allow the model to classify the image correctly. Classification is the next step, where a formula is generated based on the information from the feature extraction that aims to be able to take new data inputs and classify them. Finally, prediction or scoring evaluates how the trained model performs on new data, specifically using the metric accuracy to see how the model predicts/classifies the expected results. These initial steps of the process are essentially the training process.

Regarding mobile applications and machine learning, it seems that there are predominantly two common architectures, a server-side architecture and a client-side architecture. The server-side architecture utilises a server to handle all the critical operations of image classification. The mobile device will interact with the server by sending requests and the evaluated result whilst receiving responses to the request from the server. The application and the server interface use web services to conduct essential operations, which means the application will continuously require a connection. Meanwhile, the client-side architecture aims to alleviate some processes by loading the model onto the mobile application to conduct inference. This architecture design aligns closer to the use case of the FAIMS application’s ability to perform ML whilst the application is offline from the server.

Conducting ML through deep learning techniques is a vastly growing area of interest due to the limitless possibilities and usability such solutions provide \cite{lane\_squeezing\_2017}. The machine learning process for the edge device will initially extract features, where information from the data is taken to allow the model to classify the image correctly. The next step is classification, where a formula is generated based on the feature extraction information that aims to classify new data inputs. Finally, prediction or scoring evaluates how the trained model performs on new data, specifically using the accuracy metric to see how the model predicts/classifies the expected results. These initial steps of the process are essentially the training process.

Regarding mobile applications and machine learning, it seems that there are predominantly two common architectures, a server-side architecture and a client and server architecture \cite{ganesan\_machine\_2022}. The server-side architecture is essentially responsible for hosting the ML model as well as running the model. With inference conducted on the server, edge devices provide access to the system by requesting answers to problems and receiving responses. The difference with the client and server architecture is that it only uses the cloud for some of its processes. With the model loaded onto the mobile device via the application, inference is conducted locally with no dependency on a cloud back-end besides updating the model with new user data and retrieving the copy of the latest model update. Additionally, a new approach to the current environment will be discussed, a pure client-side ML where both the model’s training and inference occur on the client’s device. This variation dramatically deviates from the prior mentioned architectures, as there is no longer any need for a server to handle any process of on-device ML.

**Review of the Image Classification Problem**

Image classification is a primarily studied computer vision problem with many use cases and years of research supporting it. The CNN architecture is a more recent, popular network for image classification problems, a type of DNN \cite{ding\_decode\_2019}. It aims to provide machines with a function simulating optical systems found in living creatures, the ability for machines to learn from seeing and make classifications through its multi-layered network \cite{sultana\_advancements\_2018}. CNN’s will be further investigated in a later section of the paper. Image classification is essentially classifying what you see into a standard label. This function that we as humans frequently infer with daily observations of objects can be easily overlooked as just human nature. However, when considering the FAIMS application, a user must classify each fieldwork sample manually. The heart of this problem becomes apparent, a tedious process of reviewing each image of fieldwork and labelling it based on the environment or type of fieldwork conducted. Thus, it is desirable to have a solution that simplifies this process, makes it more efficient by automating it and, most importantly, makes accurate classifications \cite{nath\_survey\_2014}. It is evident that an image classification task has steps associated with it to ensure peek performance

TBF

Steps of image classification

Following this process will ensure a robust performing image classification task. As stated before, aspects can vary depending on the particular problem, and the viability of following this process may not be feasible. The question of data requirements can vastly differ depending on the chosen model, quality of data, and availability of data. These aspects concerning a lack of data availability are referred to as limited data \cite{maharana\_review\_2022}. Research suggests that the more data available to train the model, the more accurate its classifications will be \cite{cho\_how\_2016}. These clear studies do put pressure on smaller teams to deliver similar results. Though each workflow will vary, having the most data to ensure the best accuracy is not necessarily realistic. Depending on the client or stakeholders, slightly lower accuracy results may be acceptable after evaluating the performance, as it comes at the cost of sourcing less data and a quicker turnaround time for delivering a system. Alternatively, working with the limited data by utilising methods that ensure the most features can be extracted will improve the accuracy of classification \cite{chaki\_beginners\_2018}. Investing in image pre-processing techniques and other methods will make a limited data set more useful. Whilst collecting data, it is essential to have a robust classification system to assign an appropriate label to each sample image. Whilst no classification system is perfect with inaccuracies and inconsistencies, a sophisticated design will benefit the ML model used \cite{armitage\_development\_1999}. This classifying system should be built based on the requirements of the user, client and other stakeholders; it should be informative enough to satisfy the use cases of the system\cite{lu\_survey\_2007}.

Rather than sourcing an ML model to reuse, creating an ML model is an alternative path raised prior that requires additional steps. Whilst it would be ideal not to rely upon a third-party tool to achieve system functionality depending on the project scope, building one may not be feasible due to the increased steps and time needed. Furthermore, you wouldn’t be restricted by potential licensing issues of using the ML models in certain use cases and have complete transparency over how the algorithm performs. However, these benefits would be surrounded by a layer of complexity which may hamper the project and lead to scope creep. Regardless, with either a sourced or created ML model, its performance on how accurate it is with its classification task it must be evaluated. The evaluation of the model is dependent on the metrics of \cite{landau\_developing\_2022}:

Being aware of these metrics throughout the development of the model would ensure the best performance. Creating an ML model also emphasises the concerns surrounding data that were discussed. Utilising existing methods for using limited and poor quality data would be restricted as some are intended to re-train an existing model. Collecting the data would require a focus on ensuring that all potential biases have been accounted for, or potentially the model would be ineffective for the whole target use case and effective for a small sample of the use case \cite{landau\_developing\_2022}. Accounting for all the edge cases of the use case would require a larger dataset to ensure the model is adequately trained. Existing ML models exemplify this challenge as they have many parameters that allow them to achieve the results on the target problems they have specified \cite{beam\_challenges\_2020}. Evidently, creating an ML model would require more time than repurposing a model. Finally, considering the ML model will handle lots of data, there will be standards the ML system has to adhere to, to ensure the integrity of all data used. Important bodies like the IEEE, ISO and IETF have regulated appropriate approaches to systems management of data \cite{matus\_certification\_2022}. Ensuring data handling, specifically the data collection for the model, is conducted responsibly is an example of a standard that a team must adhere to as it concerns data privacy \cite{matus\_certification\_2022}. Even though many ML resources such as datasets, frameworks, and open source code have adopted the principles of open science to provide transparency, they are still scrutinised for their improper practices in managing data. This issue is further emphasised by the obscurity of the inner functionality of public ML models, which only the team behind the model exclusively understand \cite{beam\_challenges\_2020}. A purposely built ML model will have challenges that require a necessary amount of time for the project to address. This additional time focused on minor issues will only slow down progress toward satisfying the true intentions of the ML system and ultimately may be disappointing if the model results do not perform as planned due to these presented issues.

The advancements in computer vision problems and the determination to create the best DNN algorithm that caters to these issues effectively led to the rise of the CNN algorithm. Research still regards the CNN algorithm as the state-of-the-art mark for image classification problems \cite{sultana\_advancements\_2018}. Whilst evidently effective and still supported by researchers as the backbone architecture for many image classification models. However, this has not restricted studies and research from expanding the CNN algorithm further to achieve more extraordinary results. A paper by Ding presents an algorithm called the DEep COnfiDEnce network that addresses current DNNs constraints requiring cleaned and pre-processed data to function \cite{ding\_decode\_2019}. DECODE can detect data that has been mislabeled, thus noisy data that will only negatively impact the accuracy results of the model. This advancement is very beneficial in speeding up the procedures involved in an image classification task, as steps that involve enhancing data will no longer need as much dedicated time or any time at all. Another algorithm DEFEATnet by Gao addresses reducing the number of parameters required to train a model, which is commonly found in DNNs \cite{gao\_defeatnetdeep\_2016}. Through each layer of the network, crucial information is carried over, which helps minimise the number of initial parameters needed. This development would benefit smaller teams with limited data and data quality issues raised earlier in this paper when building an ML model with limitations. New advancements and research will continue to arise due to the interest in computer vision problems and DNNs like CNN, which solve these problems. CNN’s continued preference for still being the backbone architecture for models and solutions exemplifies the optimal nature of this network and its effectiveness in creating workflows for these issues. It is clearly the tried and tested approach for computer vision problems.

**Review of the Object Counting Problem**

Object counting is another ML computer vision problem that also has many use cases in a variety of different industries. Like the image classification problem, it highlights the tedious nature of counting the number of objects in an image sample when the data input size is significant. It also raises the complexity of this problem compared to an image classification problem. The model must be able to detect many objects in a sample that may be overlapping, vary in size, and the perspective of the sample may not be clear and then still count them \cite{marsden\_people\_2018}. Thus, having an automated solution achieved through ML is desirable. Research indicates that initially approaching the object counting problem follows a similar set of steps to the image classification problem \cite{pandit\_literature\_2014, khule\_automated\_2015, baygin\_image\_nodate}. So adapting the knowledge from the image classification problem and applying it to the object counting problem, the steps to ensure peek performance would be:

The similarity between the general steps for each problem is due to object counting and image classification being sub-problems of object recognition \cite{russakovsky\_imagenet\_2015}. The shared nature of both issues suggests that methods and approaches for one problem can be adapted for the other problem. Additionally, respective research for each situation will provide some alternating use to each other. However, it is essential to highlight that it will also share the same challenges as an image classification problem, as research has suggested \cite{marsden\_people\_2018}. These similarities are further emphasised by literature indicating that ML models that use a CNN as the network, which performs object detection, and segmentation, are the preferable technique for this problem as they detect objects based on classifications passed into the parameters \cite{kilic\_accurate\_2021}. Research into these problems has led to advancements in CNN which aim to reduce the runtime of an object counting task; enhanced CNN algorithms like RCNN and YOLO are the results of this research which will be discussed in a later chapter \cite{ren\_faster\_2016 ,redmon\_you\_2016}. These advancements in CNN algorithms come under object-detection-based methodology, which is one of four methodologies surrounding the object counting problem \cite{moon\_smart\_2022}. These four methods are:

Regression-based

Density-map-estimation-based

Heatmap-estimation-based

Object-detection-based

Within the regression-based models that count objects, there are sub-methods that researchers have proposed. Wang developed a deep end-to-end CNN regression model that counted the number of people in sample images \cite{wang\_deep\_2015}. Xue developed a model that calculated the number of cells in a section of the cropped input image \cite{xue\_cell\_2016}. Regardless of the difference in these approaches to developing regression-based models, they still achieved the same result of being able to count objects in an image. Additionally, they shared the same issues of counting clustered objects or when the object’s scale was distributed. These models suggest that they are suited to a specific use case or scenario.

The density-map-estimation-based method essentially wraps the object counting problem in its parent image classification problem, where the image, which is a density map, is classified and then used for object counting. The MCNN is a sub-method that Zhang presented, which consists of three columns that compute density maps for different sizes of objects \cite{zhang\_single-image\_2016}. This approach to object counting seems more appropriate for clustered data inputs like images with crowds and dense traffic.

Heatmap-estimation-based models essentially use heatmaps on an image to detect and count the objects in it. The HLCNN is the predominant model utilised for this type of methodology. Being a newer approach to this problem, it may not be a thoroughly tested and robust approach. However, research indicates it provides an accurate result by being able to detect objects effectively.

The object-detection-based method is the state-of-the-art and robust approach to the object counting problem. The objects in a sample are detected, and the number of instances is instantaneously designated and summarised \cite{moon\_smart\_2022}. Various sub-models are available due to the interest in this approach and the longer timeline compared to newer techniques. Research results already indicate strong performance, ease of use for users coming from an image classification problem and an abundant amount of support for developers due to its rooted history.

**Frameworks/Tools**

Current research into the available frameworks/tools for on-device ML will be reviewed in this section to understand the environment surrounding this field and provide the reader with an idea of the best framework/tool for undertaking on-device ML.

**TensorFlow**

TensorFlow is an open source ML software library developed by Google focusing on deep learning solutions to classification, regression, detection, transcription, and various other ML problems \cite{goldsborough\_tour\_2016-1}. Research highlights the framework’s flexibility in being deployed, being able to run on a single CPU machine, GPUs, edge devices, and multi-node large scale distributed systems \cite{nguyen\_machine\_2019}. The platform interfaces quite easily with existing systems to provide efficient ML developments by giving access to multiple ML models and tools through APIs. Services like Keras can be accessed, and integration with numerous workflows like Python, C++, Java, GO, R, JS and Haskell are all achieved using these APIs \cite{nguyen\_machine\_2019}. Additionally, it provides a workflow solution for on-device ML through TensorFlow lite by accessing the native code to deploy a compressed TensorFlow model in an application or TensorFlowJS using cross-platform development frameworks such as React Native to deploy a compressed TensorFlow model \cite{nguyen\_machine\_2019}. TensorFlowJS also allows for JS based application development and provides the tools for developing ML models like Tensorflow. TensorFlow models can ultimately be used and deployed in everyday applications like cloud applications, servers, mobile/edge devices, and web applications \cite{ganesan\_machine\_2022}. The capabilities and flexibility of this library, with constant support from the established company Google, make it a convenient development tool desirable among developers.

**Keras**

Keras is a python wrapper library that targets backend deep learning tools. Essentially, this API aims to make it easy to use an ML model’s abilities without focusing on the associated complexities on the backend. This framework follows four development principles of *“User friendliness and minimalism, Modularity, Easy extensibility and Work with Python”* \cite{nguyen\_machine\_2019}. Research highlights that this tool is not preferable for those looking to design new ML models, as its principle of Modularity means it focuses on more straightforward implementations of ML solutions \cite{nguyen\_machine\_2019}. This tool is powerful for those designing ML solutions with existing models on the backend.

**Scikit**

Scikit or scikit-learn is an open source framework that provides access to a vast selection of data mining and ML algorithms. This framework’s generalised approach to the field of ML makes it adaptable for various scenarios, and being built upon existing solid libraries ensures a lot of resources are available for development using this tool. Research alludes to the popularity of this tool due to its integration with the Anaconda environment making it favourable to statistic/analyst and data scientist teams \cite{nguyen\_machine\_2019}. This tool’s extension of NumPy and SciPy libraries makes it superbly functional for various ML problems that are exclusive and non-exclusive of deep learning.

**Pytorch**

The PyTorch framework is an interface of the C libraries found in Torch developed by Facebook to provide GPU-accelerated solutions to deep learning ML problems. Research emphasises the feasibility of quickly creating complex ML model architectures with techniques like *“reverse-mode auto-differentiation”* simplifying the process of changing network behaviours, which would generally require time \cite{nguyen\_machine\_2019}. With solid support from industrial and scientific communities and now versions of this framework designed for creating on-device ML solutions with PyTorch Mobile in beta development, this is a robust option for an ML workflow. Though its mobile device version is still a beta release, it may not be the best choice for a production build implementation for on-device ML. There may still be bugs surrounding performance and other vital concerns that may hinder system progress.

**ML Kit**

ML Kit or Firebase ML Kit is an SDK developed by Google to host and deploy ML models to edge devices. It aims to alleviate the pressures of developing ML by having APIs handle frequent use cases for on-device ML, like facial recognition \cite{singh\_mobile\_2020}. The APIs cater to any of the approaches for system architectures raised in the section \textbf{\ref{sec:appOnDeviceMl}} with any cloud approaches being hosted on the Google Cloud Platform. Alternatively, custom models built with TensorFlow Lite can be hosted and deployed using Firebase with ML Kit as an API layer for developers with specific use cases \cite{singh\_mobile\_2020}. ML Kit provides a strong option for a backend solution for easy on-device ML with pre-set APIs available with an array of deployment options. This platform can also be adapted for custom ML problems; however, this can only be achieved by utilising TensorFlow lite to develop the model, which restricts the choices available for the workflow.

**Core ML**

Core ML is a framework for on-device ML exclusive to IOS systems. It aims to solve ML problems like computer vision, natural language recognition, numeric relationships and many other ML problems. A benefit of this framework is the native support it receives from IOS, with Core ML being built into the operating system to allow optimal use of the device’s hardware \cite{ganesan\_machine\_2022}. Core ML has a significant focus on deployment options using Apple Cloud. Ease in deployment allows ML models to be developed, deployed and updated to the application as soon as changes are made. A robust opportunity for IOS development if the developer has an OSX machine, it limits any consecutive AOS development unless either AOS is excluded or a separate development stream is created.

**IBM Watson**

The IBM Watson platform is an ML tool primarily designed for server-side applications focusing on problems like chatbots, understanding natural language, language translation, computer vision and numerous other ML problems. Research indicates that ease of use makes it accessible to various parties ranging from those with limited ML experience to those looking to develop intense ML solutions. Being an established framework in the on-device ML environment has allowed it to collate a variety of packages for ML problems and easy to set up pre-made models \cite{ganesan\_machine\_2022}. Developers looking for a quick solution or who lack experience with ML principles would benefit from designing an implementation using this framework.

**Microsoft Azure Cognitive Services**

Microsoft Azure’s platform is an ML framework that was designed to target ML on server-side applications for problems like understanding natural language, computer vision, audio recognition and many other ML problems. Research highlights that the framework was aimed at usability for those with limited knowledge of ML to individuals with a greater understanding of the core concepts of ML \cite{ganesan\_machine\_2022}. Additionally, it can integrate easily with many applications and services by modifying settings or using supported APIs. Again this option for ML implementation is preferable for those with a smaller timeframe or limited ML experience due to its user friendly choices.

**Amazon Web Services**

Amazon web services is a cloud infrastructure aiming to solve a particular problem for various workflows \cite{ganesan\_machine\_2022}. Amazon Sagemaker is the framework that provides solutions to ML problems. It focuses on the issues of training models with large data streams by having a scalable and elastic system to handle the inputs \cite{}. The use case of this platform is more tailored to sizeable industrial use cases where data is a core consideration of the application.

**Google Cloud ML**

Apart from the Google infrastructure, it is a cloud ML framework with the purpose of training and deploying large scale ML models \cite{}. By storing your TensorFlow, Keras, Scikit-Learn or XGBoost model on Google Cloud, you can train or deploy the model. Deploying the model and using a server-side application through Google Cloud allows for inference on TBs of data \cite {}. This tool would be viable for a server-side architecture to ensure good performance. It also integrates easily with existing tools for ML. It may be slightly over-ambitious depending on the project size, as research indicates it targets industrial-size amounts of data, but like most cloud services may offer flexible options depending on project constraints.

**Performance**

As already discussed, the performance of each of these frameworks/tools would vary depending on the deployment options invoked. The execution of the client-side design and client-server design, where the model is loaded onto an ideal device, would be much faster than a server-side design, as inference doesn’t consider latency or network connectivity issues. Alternatively, non-ideal devices which don’t meet the hardware requirements for client-centric approaches would ultimately see better performance from a server-side system as all hardware dependencies are designated to the server. These frameworks/tools do add another layer of design concern when building an on-device ML workflow; as research highlights, each agency may utilise techniques that target the hardware of the device (dedicated GPUs, CPUs, memory, architecture design), software approaches (use of particular libraries, methods to improve the efficiency of libraries, algorithms employed in the framework) and theoretical methodologies (theoretical approaches for handling resource limited environments, algorithm design targeting performance) \cite{dhar\_survey\_2021}. A comparison of the TensorFlow framework with other popular libraries, Theano, Torch and Caffe, that are a part of or have correlations to the discussed frameworks was conducted. The research suggested that Tensorflow performed relatively well, coming second to Torch in each benchmark except for the GPU criterion, which came last. However, this result was attributed to the GPU configuration using a cuDNNv2 library compared to its competitors utilising a cuDNNv3 setup \cite{goldsborough\_tour\_2016-1}. TensorFlow, however, did optimally perform the best in word processing tasks using a miniature model against the Penn Treebank dataset. Considering that this research was conducted relatively early against the timeline of TensorFlow, it’s understandable that its performance is not truly optimised. The analysis suggested that it was not tested using its unique feature of parallel distributed computing, which was designed to handle large scale ML models \cite{goldsborough\_tour\_2016-1}. Utilising this feature could have altered its results on testing for the large scale model. As the study indicated, with its strong support, this framework still has plenty of potential for growth \cite{goldsborough\_tour\_2016-1}. TensorFlow and the other comparisons are viable options, with this growth seen with its expansion into edge device ML, suggesting that performance on various devices has been a critical concern.

**ML Models for Image Classification and Object Counting**

As stated, models already exist for relevant ML problems, which professional teams have sophisticatedly developed. Understanding the current environment of on-device ML requires an overlook of the existing models and the data requirements surrounding them. Using an existing ML model could be more effective than developing one.

**Image Classification Models**

The image classification problem has been extensively researched and resulted in open-source implementations developed. These discussed models are a selection of numerous effective models to solve the image classification problem.

**Sequential**

The sequential model is a fundamental Keras model designed on the premise of being a sequence of layers. As such, the research evaluates the model as being a linear stack of layers \cite{manaswi\_understanding\_2018, ketkar\_introduction\_2017}. Essentially the benefit of this model compared to other models is the attribute of having the creative freedom to add one or more layers to suit the needs of your task. The variety of functions that the sequential model can adapt to through manipulating its layers are \cite{ ketkar\_introduction\_2017}:

Whilst having this unrestricted design over the model does allow for catered solutions with optimal performance, this requires additional steps of concern. The critical question of how many layers are needed to provide a viable solution was answered using the heuristics approach of trial-and-error until generating a desirable result \cite{ manaswi\_understanding\_2018}. Employing a brute-force approach to creating a suitable solution with desired results can be risky as it could be uncertain how long this approach will take, with no guarantee that expected results would be produced. This problem could be enhanced further by the obscurity of the ML task being solved. The unique nature of an ML problem would restrict the number of resources to help with the problem, thus extending the process of brute-forcing a solution as there is limited direction. Documentation also suggests not using this model if it requires multiple inputs or outputs, as each layer has one input tensor and one output tensor \cite{team\_keras\_nodate}. Depending on the system’s design, having a batch of input images would not generate a collection of classified pictures. The system would have to classify each image systematically.

**MobileNetV3Large**

MobileNetV3 is an update of the MobileNetV2 model, which comes packaged as two sub-versions, MobileNetV3Large and MobileNetV3Small. Research suggests that a primary focus of this version of the MobileNet architecture is to target mobile devices’ CPUs to achieve the best performance through the design incorporating hardware awareness and efficient algorithms \cite{howard\_searching\_2019}. Thus, the model sub-versions target scenarios that may need more or fewer resources. This model performs strongly in the classification problem against its previous version and competitors. MobileNetV3Large has a 75.2% Top-1 accuracy with 5.4 million parameters, and MobileNetV3Small has a 67.4% Top-1 accuracy with 2.5 million parameters on the ImageNet dataset \cite{howard\_searching\_2019}. These are a respective 3.2% and 6.6% increase in accuracy performance from MobileNetV2 with similar parameter setups. The increased accuracy results of the model and focus on developing a robust approach to handling high or low resource devices make this an optimal choice for developing an on-device ML workflow. Being an established model in ML and on-device ML also ensures plenty of documentation and support resources available for assistance on implementation \cite{team\_keras\_nodate-3}. This model’s emphasis on providing an infrastructure targeting on-device hardware for performance may not translate to workstations, web applications and other device hardware constraints meaning that the same results may not be achieved. Regardless of this model’s catering to variating resource devices, the research only presents testing on Google Pixel devices for the classification problem \cite{howard\_searching\_2019}. It could be assumed that this was to show the unique nature of this model’s pure on-device ML ability and that performance will be consistent across the board. Though not all workflows may want a pure client-side approach, testing will indicate how it performs in these use cases.

**InceptionV3**

The IncpetionV3 model is Google’s current evolution of the 2014 GoogLeNet model, which made a significant entry into the deep ML environment. GoogLeNet was initially designed with resource constraints in consideration, only needing 5 million parameters whilst still being able to perform accurately \cite{szegedy\_rethinking\_2015}. Whilst significantly lower than its competitors at the time, it seems this design philosophy has somewhat been unprioritised as the current version, IncpetionV3, requires 23.9 million parameters for a Top-1 accuracy of 77.9\% on the ImageNet dataset \cite{team\_keras\_nodate-4}. The difference in the number of parameters is a 4.78x increase which ultimately means a more considerable strain on memory and computational costs. Whilst comparatively still a minor hit to performance compared to other networks, models like the MobileNet series were able to maintain a lesser number of parameters. However, the boost in performance of the MobileNet models does come at the cost of marginal differences in accuracy. InceptionV3 accuracy surpasses MobileNetV3Large in the Top-1 category by 2.7%. Considering use cases like on-device ML, though still achievable through InceptionV3, knowing that another network performs relatively similarly at a significant computational cost reduction raises the query of the accuracy of results being prioritised or achieving optimal performance. The research alluded to the increased parameters resulting from scaling the network to meet the needs and expectations for new use cases \cite{szegedy\_rethinking\_2015}.

**VGG16**

The VGG16 model was a network designed by the VGG team and was initially released in 2014. Whilst adding to the development of deep learning models for ML, the results of this model were second to the GoogLeNet model. Regardless, the advancements made were significant compared to the entries of 2013, seeing improvements in the error rates of the model \cite{simonyan\_very\_2015}. The current versions of the model are still relatively impressive regarding the accuracy of their classifications, with a Top-1 accuracy of 71.3% on the ImageNet dataset and low error rates \cite{team\_keras\_nodate-4}. These results come with an extensive cost to computational performance, with this model having one of the most significant amounts of parameters needed, 138.4 million. This impact on performance renders the viability of implementation for some workflows like edge device ML as not recommended, as the strain on those devices’ already limited hardware would be extreme compared to other models. Though the actual accuracy of the model is sufficient when considering all networks and VGG16 being established in the community ensures plenty of supporting resources, making this model a substantial option for a limited set of use cases. The nature of this model requiring many parameters does hinder the usability of VGG16.

**On-device ML Relevancy**

Whilst, these models achieve their intended ML purpose, they must be reviewed in terms of what this paper is trying to present, a survey and workflow of on-device ML. Fortunately, since frameworks/ tools exist like TensorFlow and Keras, as discussed prior, they provide options that allow for all these ML models to be compressed and exported so that they can support different application setups like mobile devices. Even though, theoretically, all these ML models can be implemented on-device does not mean this is the optimal approach. Some ML models were designed with limited hardware configurations in mind, thus the preferable choice to use in an on-device workflow. The determined ranking of ML models to use on-device are:

MobileNetV3 is undoubtedly the best option for an on-device ML implementation. With its significantly smaller number of parameters and competitively desirable accuracy results, it would not be overly taxing on the edge device’s computational resources whilst still functioning desirably. Comparatively, VGG16 would not be a recommended option as even though compression techniques would make the model more viable on mobile devices, the sheer size of the parameters would still noticeably impact the system’s performance due to the device’s limited hardware.

**Object Counting Models**

By extension of the image classification problem, this sub-problem has garnered much interest and developments towards solutions. These discussed models are also a selection of numerous effective models to solve the object counting problem. Additional evaluation criteria for these selected models will be whether they function with two necessary APIs, TensorFlow Object Detection API and TensorFlow Object Counting API, which would help build an object detection/counting workflow \cite{ozlu\_tensorflow\_2022,noauthor\_modelsresearchobject\_detection\_nodate}.

**Mask R-CNN Inception ResNet V2**

This model utilises the Mask R-CNN network architecture to detect objects. This extension of Faster R-CNN essentially allows the model to predict segmentation masks on objects and maintain the existing branches from Faster R-CNN for classification and bounding box regression \cite{he\_mask\_2018}. This additional branch does add an extra cost to computations meaning performance will be slightly compromised for increased functionality. This model does function with both APIs, implying that developing an object counting workflow can be done concisely. Use cases of this model are limited as studies indicate that the model is slow compared to other options \cite{ noauthor\_modelsresearchobject\_detection\_nodate}.

Regarding fitness measurement metrics, studies suggest that this model outperforms competitors like SSD MobileNetV2 in detection tasks having an average accuracy of 86.4\% compared to a 78.77\% average accuracy \cite{giron\_classification\_2020}. Documentation also states that for on-device ML, only SSD models are supported by TensorFlow Lite for box-based detection; however, this issue may be negligible due to its use of masks for detecting objects \cite{ noauthor\_modelsresearchobject\_detection\_nodate}. Since it employs an established network architecture, there are plenty of supporting resources for implementation and possible alternative ways of on-device ML. Ultimately, depending on the expected results of the workflow, an evaluation of whether the performance of hardware/resource utilisation is preferred or preference for performance in fitness metrics relating to object detection/counting tasks will need to be decided.

**YOLOV5**

The YOLOV5 is the current version of the model from the YOLO architecture. Being inspired by Google’s GoogLeNet model, it aims to be fast whilst also somewhat efficient \cite{redmon\_you\_2016}. YOLOV5 has evolved this inspiration into its design, improving the time needed for processing data \cite{kasper-eulaers\_short\_2021}. Systems handling large sets of data will benefit from a reduced timeframe. This model does not function with either APIs, which means alternative options or more time will be required to develop an object counting workflow. Additionally, not being an SSD model means its use cases are restricted from edge device ML using TensorFlowLite, which adds another constraint to designing a workflow. Different ways of implementing on-device ML will have to be sourced, potentially creating more issues depending on the obscurity of the framework/tool or approach.

**SSD MobileNetV2**

This model consists of the SSD network architecture, which employs the MobileNetV2 model as the backbone. The SSD network is a solid alternative in the environment of object detection/counting problems, as research indicates the ease of training SSD models and its robust ability with detection based problems \cite{giron\_classification\_2020}. The backbone of MobileNetV2 does not undermine these attributes of the SSD network. Though the supported version of MobileNet is not the most recent and does not receive the targeted resource benefits found in MobileNetV3, it’s still an efficient model for various workflows, specifically ones that aim for edge device ML. Studies highlight this model’s speed in detecting objects \cite{ noauthor\_modelsresearchobject\_detection\_nodate}. The responsive results of this model do come at a cost to performance in fitness metrics, where its average accuracy was 78.77\% \cite{giron\_classification\_2020}. The difference in metrics compared to a top-performing model like Mask R-CNN Inception ResNet V2 is very marginal for the trade-offs. Still, it all depends on evaluating the type of workflow needed. Additionally, this model does support both APIs and is even the suggested recommendation \cite{ozlu\_tensorflow\_2022}. Designing a workflow for these vision problems is quite viable using this model.

**SSD ResNet50 V1 FPN**

This model consists of the SSD network architecture, which employs the ResNet50 V1 FPN model as the backbone. As highlighted, the SSD network is a formidable option for designing an object detection/counting workflow. This model’s ResNet50 V1 FPN backbone also maintains the expectations of an SSD network by generating great results. Studies indicate that the model is relatively fast compared to other models in detecting objects \cite{ noauthor\_modelsresearchobject\_detection\_nodate}. Coupled with research suggesting that the model’s performance on a particular task was 65\% reliable \cite{fathabadi\_box-trainer\_2022}. Whilst not the best performing in resource utilisation and accuracy, it can interface with the APIs to help create an ML workflow. Notably, a desirable factor that will reduce the amount of time dedicated to implementing an ML workflow for an object detection/counting problem.

**On-device ML Relevancy**

Again to determine these object detection/counting models’ relevancy to the goals of this paper, they must be reviewed to determine their efficacy in on-device ML. As the beginning of this section highlighted, a critical piece of criteria for evaluating these models was whether they could utilise the APIs \cite{ozlu\_tensorflow\_2022,noauthor\_modelsresearchobject\_detection\_nodate}. These APIs are integral to building an on-device ML workflow for object detection/counting. The documentation for these APIs highlights the constraint that only SSD models are compatible with on-device ML if using the TensorFlow Lite tool part of TensorFlow frameworks to compress the models to make them more suitable for mobile devices. Ultimately, alternate options could be sourced to implement non-SSD models for on-device ML; the support these APIs provide to implementation makes SSD models more favourable. Thus the preference for models to satisfy on-device ML is:

SSD MobileNetV2 is the recommended on-device ML model for object detection/counting. Whilst not the most accurate or recent version of the MobileNet family of models it was still designed with mobile device usage and performance as a focus. The trade-off of a seamless implementation of on-device ML with marginal differences in accuracy is still a desirable outcome.

**Data Requirements**

When designing a workflow for an ML problem, essential data considerations are needed before proceeding further in development. These discussed models were initially pre-trained on large datasets like COCO or ImageNet, consisting of hundreds of thousands or millions of images labelled and assigned to classes \cite{he\_mask\_2018, russakovsky\_imagenet\_2015}. Hence, for a minor team or workflow, building a model to solve a specific problem is not viable, as collecting the data to train and validate the model would be an enormous task. The optimum solution for this scenario would be to use existing models which have gone through the strenuous process with larger, dedicated teams and repurpose the models to suit the specific use case. Though this process of transfer learning which will be discussed further in the paper, does have some critical questions:

No quantifiable amount of data will provide coverage for every use case; however, research can give an idea of what is required from re-training a model on data specific to that problem and what the results were of that model. A study that employed Google’s GoogLeNet model to classify CT scans into body parts found that the smaller the training data size, the less accurate the classifications were\cite{cho\_how\_2016}. Alternatively, the larger the training data size, the more accurate classifications were made \cite{cho\_how\_2016}. These results could arguably be expected from a general perspective on the problem. However, what is interesting is the range of training data spanning from 5\sim200 samples, with the accuracy of classifications seeing a significant increase when the training data size is \geq 20 \cite{cho\_how\_2016}. However, the study suggested that the accuracy differences when the data size increased from 100 to 200 were not significantly different \cite{cho\_how\_2016}. This research emphasises the realistic proportions of data needed for re-training a model. Depending on the project expectations, satisfactory results were seen on smaller data sizes meaning teams and workflows limited on resources can still develop a somewhat accurate classification system. Unless the project aims to create the most precise classification system, the marginal differences in accuracy found when increasing the data size past 50\% may not warrant the expenditure of resources to achieve that result.

Additionally, techniques exist that aim to expand the utility of limited data for workflows that are highly constrained for data and will subsequently generate poor classification or detection/counting results. These techniques are referred to as a process known as data augmentation. This process takes the available data and performs transformations on samples to create some variance in the data \cite{shorten\_survey\_2019}. Some of these techniques are horizontally or vertically flipping the image, modifying the RGB channels, cropping the image, rotating the image, adding noise to the sample through Gaussian distribution, and many other techniques \cite{shorten\_survey\_2019}. Data augmentation also helps with the post-classification problem of overfitting, as the variance in data will help it handle unseen data not found exclusively in the training data.

Another technique or process of ML is setting the number of epochs. An epoch is the number of iterations over the training dataset \cite{brownlee\_what\_2018}. The training dataset is allotted into a number of batches depending on the initialised size of the batch, where each contains a sample of the training dataset. Each epoch will update or train the model with the number of batches containing some of the training dataset. Setting the size of epochs is a topical process in ML, as many studies and research pieces question the appropriate integer value. Like the size of training data and its impact on the model’s accuracy, it is proposed that increasing the number of epochs also has an effect \cite{hussain\_study\_2019}. The research tests did indicate that an increased epoch size did improve the classification accuracy, where 500 epochs had an average accuracy of 91\%, whereas 4000 epochs had an average accuracy of 96.5\% \cite{hussain\_study\_2019}. This increase of 5.5\%, whilst an improvement, is a negligible difference when considering the additional need for performance resources and computational strain, thus the heightened amount of time to train and update the model depending on the machine. Comparatively to increasing the training dataset size, the same concerns around the viability are present. Depending on the project, the team and the expected goals evaluating the trade-offs of either a less or more resource-intensive system should be conducted. Considering a focus of this paper is on-device ML, the marginal exchanges for accuracy at the benefit of increased performance are preferable for this form of ML. Though an increase in accuracy resulted from the increased number of epochs, further research indicates that the excessive size of epochs could lead to overfitting; hence utilising a trial-and-error approach to find the best fit and number of epochs is an optimal strategy \cite{brownlee\_what\_2018, komatsuzaki\_one\_2019}. These are essential considerations for the quantitative process of epochs.

**Datasets**

Data is a crucial component of ML as it can be the difference between a functional and non-functional system. The data is split between the training and validation sets, known as the validation split. The training set should have a more significant share of the data.

**Ibean**

This dataset consists of 1296 images split into three classes, a healthy class and two disease classes (Angular Leaf Spot and Bean Rust) \cite{noauthor\_ibean\_2022}. Though a smaller dataset, this is realistic and will not overwhelm a minor workflow.

**Tf\_flowers**

This dataset consists of 3700 images split into five classes, daisies, dandelions, roses, sunflowers, and tulips \cite{noauthor\_tf\_flowers\_nodate}. Whilst a slightly bigger dataset, this dataset has more variation creating a somewhat more challenging workflow.

**TRANCOS**

This dataset consists of 1244 images with 46796 annotated vehicles \cite{paredes\_extremely\_2015}. This dataset was designed with the object detecting/counting ML problem as a focus.

**COCO 2016/2017**

This dataset consists of 328000 images with 91 object types and 2.5 million image instances labelled \cite{lin\_microsoft\_2015}. Launched by Microsoft, this dataset was created with image classification, object detection and segmentation problems being targeted. Being a larger dataset, the amount of data may prove overwhelming for smaller teams.

**Provided Data**

Rather than sourcing somewhat similar datasets to the target use case, providing a dataset directly related to the problem domain would benefit the workflow for a more refined, relevant solution that saves time searching for or collecting data. Since the CSIRO is a prevalent stakeholder in this project with a clear vision for the intended use of this system, being provided essential data from this source would be exceptionally useful to the workflow of the project.

**Collecting Data**

As discussed, collecting data would require a sophisticated classification system to ensure the sample images can be assigned to their corresponding classes through labelling each image. A careful approach would provide quality data to help the model with the classification or detection/counting problem.

**Transfer Learning**

Transfer learning is a necessary process for handling new unseen datasets with existing ML models which were pre-trained on a large dataset. Essentially, the process involves adapting what the ML models learnt on their initially trained data and applying it to new related data. Research emphasises that this paradigm aims to use robust models that generate great accuracy results on their pre-trained datasets and get these same results on new datasets that the model is re-trained on \cite{hussain\_study\_2019}. Specifically, the process involves taking the weights from the model’s training on the original dataset and using them as the initial weights for the new task, whilst the rest of the model will respond to the latest data \cite{ shorten\_survey\_2019}. This concept benefits smaller teams or workflows with limited time constraints, as they do not have to build an ML model which consists of many intricacies like training it on an excessive dataset like ImageNet.

Additionally, smaller datasets can be used, which also benefits the workflow. Results of Transfer Learning also suggest the reliability of the process, with testing presenting average accuracy outcomes ranging from 65\%\sim71\% \cite{hussain\_study\_2019}. This study also highlighted a comparison made to research that showed a constructed model by a team generating classification accuracy results of 38\%{hussain\_study\_2019}. Considering the effort of building an ML model for a problem from scratch, this clearly shows the power of Transfer Learning. With a massively reduced workload surrounding the complexities of the model, generating a nearly double accuracy score for a specific classification or detection/counting problem is very promising.

**Overview of Computer Vision ML Models**

Computer vision based problems are so prevalent in research due to the unlimited potential of use cases these problems can solve in society. Understandably, studies have led to the refinement of networks and techniques that provide the core architecture of ML models.

**Deep Neural Networks**

Deep learning networks, or DNNs, are state-of-the-art designs for computer vision problems. Their ability to replicate human cognition by learning from what they see is unparalleled and highly effective for solving issues. With research continuing into DNNs, developments of the network have progressed with specific versions intended to specialise in particular problems.

**CNN**

CNN is a multi-layer artificial neural network inspired by time-delay neural networks and designed to manage input data that is two-dimensional \cite{al-saffar\_review\_2017}. Research suggests that the CNN network has a very flexible design by changing the network’s capacity by modifying the network’s depth and breadth, having fewer network connections and weight parameters \cite{al-saffar\_review\_2017}. A flexible design is beneficial as it allows for the tailoring of a tool for specific scenarios. CNN is very popular among image classification problems, as it was created to solve those kinds of tasks. The study also highlights the succinct functionality of a CNN to analyse an image, extract features from it, classify them and then output the classification result \cite{ al-saffar\_review\_2017}.

**RCNN**

The RCCN network is an extension of CNN to specialise in object detection problems. Being an early adaption of CNN, the RCNN network has become a series of iterations, each addressing performance issues and general improvements of the network \cite{he\_mask\_2018, ren\_faster\_2016}. The versions are in sequential order of release:

The Faster RCNN update sees the network consist of two core modules that aim to specify regions of an image or footage that will be classified into defined objects. Incorporating RPN functionality with Fast RCNN, Faster RCNN can effectively detect objects by outlining them in a rectangular box and providing a class label \cite{ ren\_faster\_2016}. Mask RCNN is the current network version, which expands upon Faster RCNN by incorporating object segmentation mask as a third output \cite{ he\_mask\_2018}. Along with its variability, the support this network receives emphasises the potential and possibilities that can be achieved by creating models using it.

**SSD**

The SSD network is an object detection based network that extends from CNN. This method for object detection achieves its results by using a single deep neural network \cite{ liu\_ssd\_2016}. Due to this design, SSD is easy to train and implement into systems. Additionally, this concept benefits its need for computational resources. Research highlights that compared to slower, more resource intensive networks, it’s still able to perform or outperform equally in speed and accuracy \cite{ liu\_ssd\_2016}. Due to the resource intensive nature of networks like RCNN and YOLO, implementing them on devices with limited hardware which can’t be changed easily would be difficult. A method that allows detection based computer vision problems to be solved on embedded devices, edge devices or any device with restricted hardware expands the usability and options for use cases.

**YOLO**

The YOLO network is another object detection method that expands from CNN. Like SSD, it steps away from the bounds of region based object detection that is employed in the RCCN network. The network defines object detection problems as a regression problem \cite{ redmon\_you\_2016}. Similarly to SSD, YOLO uses a single neural network to detect objects into boxes and classify them \cite{ redmon\_you\_2016}. This means the network’s performance will be fast and easy to work with, which benefits more use cases. Though research suggests that the accuracy of this method comparatively is lacking \cite{ redmon\_you\_2016}. Compared to SSD, this is true in both performance and accuracy.

**HLCNN**

HLCNN is another object detection network that builds upon and improves the CNN architecture. This network varies the general output of detecting an object and classifying it based on a label by generating a heat map of the targeted object \cite{kilic\_accurate\_2021}. Like previously discussed methods, this network also implements a single deep neural network for its object detection \cite{kilic\_accurate\_2021}. This suggests that this network will be generally effective in a variety of metrics, especially when compared to RCNN. Though this network is relatively new to the current environment, it’s assumed there is still a lot of room for growth before this network is truly viable.

**Traditional Machine Learning Techniques**

Though it is evident that using deep learning neural networks is the preferable solution for computer vision based problems, other techniques for solving these issues are practical. These methods should be discussed to deepen the understanding of approaching these problems and apply any lessons learnt.

**Decision Trees**

Decision trees are a supervised ML algorithm that classifies based similarly on if/else conditions. Whilst, not a neural network, decision trees still have a specific complexity that benefits image classification. The tree structure consists of one root node, internal nodes, and a set of terminal nodes \cite{otukei\_land\_2010}. The tree nodes are accessed recursively, passing the data into the conditional statements for classification. The study shows the results were relatively accurate for classifying images \cite{otukei\_land\_2010}. Though a decisive outcome, as other research indicates, traditional ML techniques usually had a higher error rate when compared to deep learning networks \cite{otukei\_land\_2010}. Even though accurate results can be generated, there are still significant considerations for all aspects of ML that need to be addressed.

**K Nearest Neighbours**

The K Nearest Neighbour is a straightforward and standard ML algorithm that can be used for various problems. The algorithm has proven itself useful in the space of computer vision problems, specifically for image classification. Classifying for this method involves the algorithm determining the closest target items to the closest training examples in the feature space \cite{nurwauziyah\_satellite\_2018}. The training process is also quite simple, only requiring the labels from the training data and storing of feature vectors \cite{nurwauziyah\_satellite\_2018}. The results of this research did yield accurate outcomes for the algorithm for this problem and performed better than the Decision Trees technique \cite{nurwauziyah\_satellite\_2018}. Whilst a solid alternative to a traditional approach, the same issue of all-around performance arises where deep learning networks are comparatively more rounded.