|  |
| --- |
| EPQ Project Report |
| Determining the best AI algorithm to accurately check if a person is wearing a face mask, and to assess the usefulness of such a program. |
| Mahmoud Hussein |

Introduction

During the COVID-19 Pandemic, masks were required in most public settings. From trains and buses to shops and restaurants masks were required. Changing regulations with new variants mean that at any instance masks requirements can be reinstated. In fact, masks are still encouraged if not required in hospitals. For example, St George’s University Hospitals requires facemasks as of October 2022. Through my own experiences in hospitals, however, I noticed that many people still weren’t wearing masks. Dr Serife Mehmet, a doctor working at Queen Elizabeth’s Hospital, confirmed my suspicions when I discussed the project with her, saying: “Many people forget that masks are still encouraged or even required in hospitals. This often creates an awkward situation for workers who must remind people to wear masks. In A&E, workers simply don’t have the time to keep reminding people to put on their masks.”. This presented a clear use case for an application that can detect whether someone is wearing a mask and provide a visual prompt to wear a mask. Leading me to teach myself to programme in order to write and develop the application. Said application also means that, if in the future mask mandates become a reality again, they can be enforced without a need for extra staff – just a webcam and computer.

With the use case of this application in mind I set out the following criteria for the success of this project:

* The program needs to be accurate. It needs to be both accurate on a data set and also accurate when tested on live volunteers.
* The program needs be user-friendly. This is so that it easily accessible to the general population.
* The program should minimise how much memory it needs and should be as efficient as possible.
* The program’s code should be clean and be able to be understood by another programmer.

Programming Language

There are many different programming languages which all have different pros and cons as well as different individual use cases. For my project which is based around machine learning, my initial research immediately pointed me in the direction of Python being the industry standard for 57% of data scientists and machine learning developers(Voskoglou, 2017) . Furthermore, Python is one of the most popular languages for general use that isn’t related to machine learning. For example, Python was the second most popular language on Github, a code hosting platform used in industry, as of 2021 (Github, 2021). The popularity of Python meant there would be a lot of support, documentation and courses that I could draw upon if needed when creating the program. In addition, its use for machine learning in particular meant that I had an abundance of different machine learning modules available each with large amounts of documentation and support, whereas other programming languages may only have 1 or 2 if any that I would be forced to use with likely little support available as a result of a lack of popularity. However, the major downside to Python was its speed. Python is an interpreted language which must be “parsed, interpreted, and executed each time the program is run” which greatly adds “to the cost of running the program” (IBM, 2010). This contrasts with compiled languages, like C or C++, where the code is translated by running the source code through a compiler creating one executable file which can be run any number of times without retranslation (IBM, 2010). This, therefore, means that “the overhead for the translation is incurred just once, when the source is compiled; thereafter, it need only be loaded and executed” (IBM, 2010) . Essentially, complied languages only need to be translated once which, therefore, means they only require the computational resources for this once. On the other hand, interpreted languages need to be translated every time the application is used, thus incurring extra memory costs and computation time when running the application. Therefore, interpreted programs are usually less efficient than compiled programs (IBM, 2010). Despite this, however, I still decided to use Python thanks to it being the language of choice for machine learning in industry.

The main resources I used to learn Python was using a beginner course on the language’s fundamentals (Codecademy, n.d.) and then progressed to a YouTube course that covered TensorFlow (freeCodeCamp, 2020), the main module I was using for machine learning.

Why TensorFlow?

I chose to use Google’s TensorFlow for the machine learning aspect of the project thanks to its popularity within the field, meaning there is lots of documentation and tutorials based around it. My use of Tensorflow was also supported by transition to Google Colab, as further discussed later in the report. Colab best supported Tensorflow as they were both made by Google and thus designed to work together. Furthermore, TensorFlow works very well in tandem with other modules, in particular NumPy which is a module that makes array handling more efficient in Python and thus helps meets one of my criteria for success.

Dataset

“Garbage in, garbage out” is a common adage used in the context of data for machine learning. This is because the success of your machine learning model largely dependent on said dataset. “Simple models on large datasets generally beat fancy models on small data sets. Google has had great success training simple linear regression models on large data sets” (Google, 2022), thus indicating the importance of the dataset’s size. Quality is also important. The quality of a dataset can be broken down into reliable data and data that minimises skew (Google, 2022). A reliable dataset is one you can trust is correctly labelled and has relevant images. Minimising skew simply means to ensure the dataset is general and isn’t skewed towards something, like a particular feature or image type. With these criteria in mind, I decided to use the MaskedFace-Net dataset. This dataset was the largest in size I could find (133783 images) and had been used in projects in the past meaning I could be sure of its ‘reliability’. I did notice during my project the images weren’t as varied as they could’ve been, skewing the machine learning model towards a specific scenario. I attempt to address this in the model later.

Google Colab

I had initially planned on training the model on my personal computer. I quickly realised this wasn’t feasible and decided to switch to Google Colab, which is a cloud-based programming environment that allows anyone to run python code from their browser. Colab is especially suited to machine learning thanks to it giving you free, virtual access to googles Graphics Processing Units. GPUs are what are generally used for training machine learning models thanks to them being “a single-chip processor used for extensive Graphical and Mathematical computations” (Dsouza, 2020). GPUs are much faster at performing calculations than what was readily available on my home computer, as illustrated in this diagram comparing GPUs to CPUs:

Chart, line chart

Description automatically generated

(Shaw, 2021)

This diagram shows GPU’s consistently outperforming CPUs, being able to complete around 10 times more calculations than CPUs in the same period of time as of 2016, demonstrating why GPUs are preferable to CPUs.

GPUs are specifically preferable for machine learning thanks to them being able to process multiple computations simultaneously unlike CPUs (Dsouza, 2020). In fact, a neural “network’s size is limited mainly by the amount of memory available on current GPUs” (Krizhevsky, et al., 2012). Therefore, getting access to the best GPUs available and speeding up the training process were the key benefits of Google Colab. This is further illustrated here, where machine learning training times are compared on different machines:

Chart, bar chart

Description automatically generated

(Radečić, 2020)

The above graphic demonstrates that Google Colab takes the shortest amount of time, 523 seconds, when training a dataset in comparison to different laptops and computers.

Loading Data

A key issue encountered near the beginning of the project was loading the data. When training machine loading models one must load the data onto their random-access memory, RAM for short, to be used by the program. Google Colab gives you access to 12 GB of RAM while my dataset was nearing 40 GB. Thus, indicating it wasn’t feasible to load all the data all at once. I, therefore, opted to load my data in batches and successively train the model. This would mean I loaded 2000 images at a time and train the model on those images, then delete the images from the RAM and load another batch for further training. This is an example of an ‘input pipeline’, which is a series of processes which takes the data from in your hard drive to being fed to the machine learning model. An input pipeline allows the handling of the large amounts of data I planned on using by loading the data in “chunks” which minimises how much memory I use (Shukla, 2021). Although an input pipeline can be implemented through TensorFlow, I opted to make my own from scratch as it gave me greater control and meant I didn’t have to make any significant changes to how my data was stored and labelled. A course on input pipelines was especially helpful in teaching me the key principles (Codecademy, n.d.). The code for this follows below:

Text

Description automatically generated

Line 33 lists all the file paths within each of the directories listed in line 25 and temporarily stores two of them within a variable, one file full of correctly masked images and one with incorrectly masked images. The lines 35 to 39 then iterate through said files. First the function in lines 49-51 occurs which resizes the images to 224 by 224 pixels. The images are then stored in the RAM as an array. An array is a type of data structure that contains a collection of elements. Each element contains the numerical colour data for each pixel, therefore one element in the array corresponds to one pixel of the image. I ensure to also store the correct label, masked or unmasked, in a separate array where the position of a label corresponds to the same position of its respective image. In lines 41-44 I then randomly shuffle the images. This is done so that the order of how the images is fed to the machine learning model doesn’t have any influence. When shuffling the images, I ensure the labels are also shuffled in the same way. This is made possible using a ‘seed’. The random function in python uses a complex algorithm that takes some number as an input and returns an output. If you input the same number, you will receive the same output. Normally the function takes the date and time on your computer as the input, however I can set a seed before the each, as done in lines 41 and 43. I set the seed as the same number which means the list of images and list of labels will be shuffled in the same way; this means that each label will still correspond to its image. The images and labels can now be used for training.

The Model

The model will have the general structure of a neural network as show in the following diagram:

Diagram

Description automatically generated

An Input layer, several hidden layers and then an output layer. In the input layer, each node should represent one pixel. A node can be thought of as a container which stores a number. Therefore, since my images are 224 by 224 pixels – the input layer would have 50176 pixels. On the other hand, the output layer would only contain two nodes: One will be on if the person is masked while the other will be on if the person isn’t masked. The hidden layers perform different calculations and apply different weights and biases to help the neural network conclude.

The first iteration of my model had this code:

Text

Description automatically generated

Line 55 above creates the input layer and indicates that I am using a convolutional neural network – a “form of deep networks widely used in computer vision” (Srinivas, et al., 2016), which are able to achieve record-breaking results on even the most challenging datasets (Krizhevsky, et al., 2012), thus indicating why I used it. The next lines create hidden layers of varying sizes. The 65th and 66th line of the model applies an activation function to the results. “The activation function is used to transform the activation level of a unit (neuron) into an output signal” (Sibi, et al., 2013). For example, the activation function I used in this iteration of the model was Rectify Linear Unit (‘relu’):



This function takes some numerical input and returns the same number if its positive or returns zero if its negative. The outputs are therefore between zero and infinity. My initial reasoning for choosing this activation function is that the function being linear makes it very easy to compute – again speeding up the model and training times. Line 63 creates the output layer. Line 65 then compiles the model, setting its metrics for success as its accuracy.

Although this model had a high accuracy on the training and test data set, it performed poorly on my tests from my own webcam as shown later in the report when I tested and compared the different iterations of my program.

A person wearing a mask

Description automatically generated with low confidence A person wearing a mask

Description automatically generated with low confidence

The images here show the poor accuracy when further away from the camera. The first image I am further away, and the model incorrectly thinks I am not masked mask, while in the second image I am closer, and this is no longer the case. This indicated to me that the program wasn’t general enough, which is again further supported by my tests later in the report. I attempted to address this in my next iterations of my model. Due to this poor accuracy and a large memory size, I opted to use a pre-built model as the base and then train and build upon said model in the next iterations of my model. Pre-built models are publicly saved machine learning models which have been trained on a massive dataset of different images (Google, 2022). I would then implement transfer learning, which is the further training of the pre-built model on specific images (Google, 2022). This will fine tune the model towards the specific task of detecting whether someone is wearing a face mask or not. Transfer learning on a pre-built model brings the advantages of saving resources and being more efficient (Sharma, 2022). I opted to use MobileNetV2 thanks to it being designed to maximise accuracy while being mindful of the restricted resources on some devices. MobileNetV2 massively reduced the size of the model and thus removing a key barrier of entry to using the app, not everyone has a big hard drive for an app. My initial model had a size of 173 MB while the model implementing MobileNetV2 had a size of 11 MB, a size reduction of over 10x.

The machine learning model now had this code:

Text

Description automatically generated

Line 53 sets the base of the model as MobileNetV2, setting the input size of 224 by 224 which are the dimensions of the images. I then added lines 57-59, with the purpose of helping the model generalise better. These lines apply random transformations to the images when training the model. These transformations include randomly zooming and randomly rotating the image, which is the “easiest and most common method to reduce overfitting on image data” (Krizhevsky, et al., 2012). Overfitting happens when a statistical model is too closely tailored or fitted to its training data. (IBM, 2021), this means that the machine learning model isn’t general and versatile enough. Thus, augmenting the data in this way will increase the variability of the dataset to make the model more accurate on real world data. Through lines 61 – 65 I added the hidden layers. Another key difference from the previous model is the addition of a dropout layer in line 65. This layer randomly drops a node out, which means temporarily removing it from the network (Nitish Srivastava, 2014). This will help to prevent overfitting by discouraging the neurons from relying too much on each other. By randomly removing connections, each neuron is forced to learn more general and robust features that are useful in conjunction with many different combinations of other neurons.

(Krizhevsky, et al., 2012). Essentially the model will randomly deactivate nodes in the neural network during the training process. This will mean that the machine learning model cannot be overly dependent on a single node which would act as a possible point of failure, making the model more robust. The only other notable difference with this model is the changing of the activation function to ‘sigmoid’:



Through further research I found that this function was better suited to my problem. “The sigmoid function will only produce positive numbers between 0 and 1 which, therefore, makes it most useful for training data that is also between 0 and 1 (Sibi, et al., 2013). Therefore, since my data also falls between 0, not masked, and 1, masked, I opted to use ‘sigmoid’. However, the same study also concludes that, although the “activation function is one of the essential parameters in a Neural Network”, “the performance evaluation” of their own tests showed that “there is not a huge difference between them” (Sibi, et al., 2013). Meaning that their research indicates the activation doesn’t play a huge part in the accuracy of the model. Thus, I doubt that the change in activation function was a large contributor to any changes in accuracy or performance.

GUI

The Graphical User Interface was another important part of my project as it is what makes the program usable for the general population. With my goal of user-friendliness in mind, I aimed to have the program need as little user input as possible to work – minimising room for error and confusion. I achieved this, having the program detect your webcam and start displaying whether someone is wearing a mask automatically. I utilised the Python module ‘Pygame’ – a module that allows rendering of graphics to the screen. I chose this module over alternative modules as it was the easiest to implement and gave me the greatest control. This was made especially clear when experimenting with ‘PyQt5’, an alternative module, where I had to go through several steps to simply display an image to the screen, overcomplicating what should have been something simple. In addition, ‘Pygame’ is built upon C while ‘PyQt5’ is built upon C++. C is the predecessor to C++ and has less features. Studies have shown that “C++ programs apart from being slower than their corresponding C versions, consume significantly more energy" (Chatzigeorgiou, 2003), indicating that C++ is more intensive than C and programs built in it take longer to compute. Therefore, for the simple task of displaying images I opted to go with ‘Pygame’ thanks to the fact it is built upon C and, therefore, would be less performance intensive which meets one of my goals of making the program as lightweight and efficient as possible.

Feedback on code

I do not study Computer Science A Level, but I asked a Computer Science teacher to give me some feedback on the code. She was able to give me feedback to help improve my code and come closer to one of my goals of making the code as efficient and clean as possible. One piece of feedback she gave was in reference to the program’s maintainability, which is the process of ensuring that “it is easy for other programmers to understand what the code does” (CGP, 2017). She suggested that I improve the maintainability of my program. In response to this, I research and implemented ‘comments’ into my program. These are written within the program to explain “what the key features of the program do” and are “fundamental for helping other programmers understand your programs” (CGP, 2017). In Python, comments are prefixed with a ‘#’ and are ignored by the program. Examples of how I implemented comments are below:







The Computer Science teacher also raised the issue of compatibility with different devices. In response to this I compiled my final product into a singular ‘.exe’ file. This means all the different python files, modules and the machine learning model are compiled into a single application. This removes the need for the user to install Python and all the modules separately for the application to work.

Finally, the teacher commended my use of subroutines that make the program efficient as well as the effective use of different python modules.

Comparing the models

By the end of the project, I had two machine learning models: One built from scratch and another that builds upon the prebuilt model ‘MobileNetV2’. To compare the models’ effectiveness, I tested them in two ways: Firstly, on images from the same dataset that the models hadn’t seen yet and secondly, I conducted my own test using real world data. To test the models in the first way I used the TensorFlow ‘evaluate’ function as demonstrated below:



This takes the test images and test labels as inputs, tests the models, and then returns the accuracy. I repeated this with 8 batches of 2000 images. The results of this can be found below:

*Chart, line chart

Description automatically generated*

The solid line represents the model built from scratch, which I will henceforth refer to as Model1, while the dashed line represents the model using ‘MobileNetV2’, Model2. The above graph demonstrates that in my testing, Model1 has a consistently higher accuracy than Model2. Model1 hovers around 99% accuracy while Model2 has around 92%-93% accuracy. Therefore, suggesting Model1 is more accurate. Although Model1 is more accurate, both models have above 90% accuracy. Therefore, I wouldn’t say these results are significant to say that one model is better than the other. Instead, I would argue the real deciding factor would be which model performs better on real world data as this better reflects the situations in which the program will be used.

To test the program on real world data I tested to see if each model would be able to correctly identify whether someone was or wasn’t wearing a mask. I would wear a mask and check at a close (0.5m), medium (1m), and far (2m) distances. I would then check again without a mask. I repeated these two more times and then repeated the whole thing with a different person.

I started with Model2. Throughout the entire test, Model2 never incorrectly said someone was wearing a mask when they weren’t. When someone was wearing a mask, the model was very accurate at identifying that when close to the camera – correctly identifying it 13/15 times or with an accuracy of 87%. The results were similar at a medium distance with Model2 correctly identifying someone was wearing a mask 11/15 or 73% of the time. However, at the far distance the model almost never correctly identified someone was wearing a mask – only 3/15 times meaning an accuracy of 20%. Thus, when someone is wearing a mask, the model was able to identify it 60% of the time. This jumps to 80% of the time when only accounting the close and medium distance, however. In these tests, however, the person’s face was always central and looking straight at the camera. When the person looked to the side or wasn’t centred, the model almost never correctly identified whether they were wearing a mask – always saying they weren’t. This indicates the model has been overfit to the dataset which was made up of ideal photos where they are centred and looking at the camera.

Testing Model1, I found that, unlike Model2, there was inaccuracy when someone wasn’t wearing a mask. In fact, it would only correctly identify when someone wasn’t wearing a mask 34/45 times- giving an accuracy of 76%. When wearing a mask, the model was able to identify it correctly 14/15 times when close. However, at the medium distance it could only correctly identify someone was wearing a mask 8/15 times and was never correct at the far distance. This gives it an overall accuracy of 49% as well as the accuracy of 76% when identifying someone isn’t wearing a mask.

These results therefore indicate that Model2 is more accurate on real world data than Model1.

Feedback from End User

I was also able to receive feedback on the application from Dr Serife Mehmet. She commended the user-friendliness of the application, approving of the lack of user-input required for the application to work and that it automatically detects the webcam. She does, however, comment on the accuracy of the program, specifically the model needing the person’s face to be centred and looking at the camera. Although this would work in slower moving parts of a hospital or GP surgery, it isn’t feasible for people to take time and space doing this in places like A&E.

Although I couldn’t improve the accuracy of the model with the resources, I had available, namely a much larger dataset with a greater variety of images, I added a graphic to my program that directed the users to position themselves in the centre, as demonstrated in the image below:

A picture containing text, indoor

Description automatically generated

This will direct users towards the centre where the model is most accurate in attempt to address the issue raised by Dr Serife Mehmet.

Conclusion

In this report, I have described the development and evaluation of an app that utilized machine learning to detect the presence or absence of a facemask in images of individuals. The project involved the development of two machine learning models, one created from scratch and one built on top of a pre-built model. Through testing and evaluation, it was determined that the model built on top of the pre-built MobileNetV2 model was more accurate when presented with real-world data. This was likely due to the pre-built model having already learned features that were relevant to the task, whereas the model built from scratch was overfitted to the limited dataset and not as effective at generalizing to new data.

While the chosen model performed well in most cases, it was found that the accuracy of the model decreased when the user was not positioned centrally and facing the camera. This limitation indicates that the dataset used to train the model did not have enough variation in terms of the positions and poses of the individuals. To address this limitation, artificial transformations were applied to the training images in an attempt to augment the dataset. However, it was noted that these transformations can only go so far in simulating real-world variations. As a result, it was concluded that a larger and more diverse dataset is necessary to improve the accuracy of the model when presented with new data.

To further improve the accuracy of the model, it is recommended that future iterations of the project focus on collecting a larger dataset with a wider range of poses and positions. This could be achieved through manual data collection or by utilizing techniques like data augmentation. It is also recommended that the model architecture be further optimized to better handle variations in poses and positions.

Overall, the project highlights the potential of machine learning in addressing real-world problems like mask detection. However, it also emphasizes the importance of high-quality data and the need for a diverse and representative dataset to train accurate models. If I were to continue with this project, future work should be done focusing on expanding the dataset and incorporating techniques to better handle variations in poses and positions, ultimately leading to more effective and accurate mask detection tools.

Glossary of Key Terms

Parsed: Analyse (a string or text) into logical syntactic components.

Compiler: A program that translates a programming language's source code into machine code, bytecode or another programming language.

Skew: The degree of distortion from a normal distribution. In the case of machine learning, the model incorrectly distorts the data as a result of a dataset that isn’t representative of the population.

Neural Network: A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain. Artificial Neural Networks are comprised of layers of nodes, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Array: A data structure that holds similar, related data. Each individual piece of data within an array is called an element.

Hidden Layers: Layers in a neural network that perform nonlinear transformations of the inputs entered into the network.Machine learning: a subset of artificial intelligence (AI) that focuses on developing algorithms and statistical models that enable computer systems to learn and improve from experience without being explicitly programmed.

Python: a high-level programming language used for various purposes including web development, data analysis, and machine learning.

Interpreted language: a programming language that is not compiled into machine code before it is executed. Instead, the code is interpreted at runtime, which makes the execution slower than compiled languages.

Compiled language: a programming language that is translated into machine code before it is executed. The resulting executable file can be run multiple times without retranslation, which makes the execution faster than interpreted languages.

Computational resources: the hardware and software required to run a program, such as the processing power, memory, and storage of a computer.

Memory costs: the amount of memory required to store the program and its data.

Computation time: the amount of time required to execute a program.

TensorFlow: an open-source software library for data flow and differentiable programming across a range of tasks. It is primarily used for machine learning applications such as neural networks.

GPU: The Graphics Processing Unit is a specialized electronic circuit designed to quickly manipulate and alter memory in order to accelerate the creation of images in a frame buffer intended for output to a display device. In recent years, GPUs have also been used for general-purpose computing tasks, such as machine learning and scientific simulations, due to their high computational power and ability to perform many calculations simultaneously.

CPU: The Central Processing Unit is the primary component of a computer that performs most of the processing of data and instructions needed to run software and applications. It is often referred to as the brain of the computer and is responsible for executing the instructions of a computer program, performing arithmetic and logical operations, and controlling the input/output operations of the computer system. The CPU is typically composed of one or more processing cores that work together to execute instructions and process data.

RAM: Random Access Memory is a type of computer memory that stores data and instructions temporarily, allowing the CPU to access them quickly for processing. RAM is volatile, meaning that it requires power to keep the data stored in it. When the power is turned off, the data stored in RAM is lost.

Input Pipeline: An input pipeline is a method of feeding data into a machine learning model. It involves a series of steps to load, preprocess, and transform data into a format that can be consumed by the model.

Activation function: An activation function is a mathematical function that is applied to the output of a neural network node or neuron. Its purpose is to introduce non-linearity to the output, allowing the network to model complex relationships between inputs and outputs. Activation functions are typically used to transform the weighted sum of the inputs into a range that can be interpreted as an output or prediction. Common activation functions include sigmoid, ReLU, and tanh.

Graphical User Interface: Graphical User Interface (GUI) refers to the visual elements of a software application or operating system that allow users to interact with the system.

Model: A model is a mathematical representation of a system or a problem, which is designed to make predictions or decisions based on input data.

Function: A mathematical function is a rule that maps one set of values (the domain) to another set of values (the range).

Linear: Linear refers to a relationship or function where the output is directly proportional to the input, and can be represented as a straight line on a graph.

# References

CGP, 2017. *GCSE OCR Computer Science.* 1st ed. Newcastle Upon Tyne: CGP.

Chatzigeorgiou, A., 2003. Performance and power evaluation of Cþþ object-oriented. *Information and Software Technology,* 45(4), p. 195.

Codecademy, n.d. *Build a Machine Learning Pipeline.* [Online]   
Available at: https://www.codecademy.com/learn/build-ml-pipelines-course

Codecademy, n.d. *Learn Python 2.* [Online]   
Available at: https://www.codecademy.com/learn/learn-python

Dsouza, J., 2020. *What is a GPU and do you need one in Deep Learning?.* [Online]   
Available at: https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d  
[Accessed 19 September 2022].

freeCodeCamp, 2020. *TensorFlow 2.0 Complete Course - Python Neural Networks for Beginners Tutorial.* [Online]   
Available at: https://youtu.be/tPYj3fFJGjk

Github, 2021. *The 2021 State of the Octoverse.* [Online]   
Available at: https://octoverse.github.com/  
[Accessed 24 October 2022].

Google, 2022. *The Size and Quality of a Data Set.* [Online]   
Available at: https://developers.google.com/machine-learning/data-prep/construct/collect/data-size-quality  
[Accessed 20 October 2022].

Google, 2022. *Transfer learning and fine-tuning.* [Online]   
Available at: https://www.tensorflow.org/tutorials/images/transfer\_learning  
[Accessed 11 November 2022].

IBM, 2010. *Compiled versus interpreted languages.* [Online]   
Available at: https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-compiled-versus-interpreted-languages  
[Accessed 30 October 2022].

IBM, 2021. *Overfitting.* [Online]   
Available at: https://www.ibm.com/cloud/learn/overfitting  
[Accessed 24 October 2022].

Krizhevsky, A., Sutskever, I. & Hinton, G., 2012. ImageNet Classification with Deep Convolutional neural networks. pp. 1, 5-8.

Nitish Srivastava, G. H. A. K. I. S. R. S., 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research,* p. 2.

Radečić, D., 2020. *Google Colab: How does it compare to a GPU-enabled laptop?.* [Online]   
Available at: https://towardsdatascience.com/google-colab-how-does-it-compare-to-a-gpu-enabled-laptop-851c1e0a2ca9  
[Accessed 19 September 2022].

Sharma, C., 2022. *Comparison of CNN and Pre-trained models: A Study.* Waterloo: s.n.

Shaw, A., 2021. *Intro to GPUs.* [Online]   
Available at: https://course19.fast.ai/gpu\_tutorial.html  
[Accessed 19 September 2022].

Shukla, P., 2021. *Getting Started With Tensorflow Input Pipeline.* [Online]   
Available at: https://medium.com/analytics-vidhya/getting-started-with-tensorflow-input-pipeline-aa0f31eea74a#:~:text=The%20input%20pipeline%20is%20a,in%20few%20lines%20of%20code.  
[Accessed 24 October 2022].

Sibi, P., Jones, S. A. & Siddarth, P., 2013. Analysis of different activation functions using back propagation neural networks. *Journal of Theoretical and Applied Information Technology,* 47(3), pp. 1266-1267.

Srinivas, S. et al., 2016. A Taxonomy of Deep Convolutional Neural Nets for Computer Vision. *Frontiers in Robotics and AI,* 2(36), p. 1.

Voskoglou, C., 2017. *What is the best programming language for Machine Learning?.* [Online]   
Available at: https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7  
[Accessed 24 October 2022].