CAPSTONE PROJECT 2

Planning Document

Project Title

Using machine learning to improve energy efficiency when loading web sites

by

Teo Wei Shuen

19103647

BSc(Hons) in Computer Science

Supervisor: Ms Charis Kwan Shwu Chen

Semester: September 2023

Date: 7 December, 2023

Department of Computing and Information Systems

School of Engineering and Technology

Sunway University

Abstract

This capstone project presents a comprehensive research into the possibility of using a machine learning model to reduce power consumption when attempting to load a tab with highly detailed images. Due to real world problems like digital inequality, there is a rising need for these approaches to exist and be implemented into the current web infrastructure. The study also sheds light on various reasons as to why images in traditional websites should not be as detailed as they are normally rendered. Extensive research was done on other approaches that could be used such as predictive prefetching and removing redundant code, but not all proposed methods were used, the main focus was still directed towards using a machine learning model to generate a compression scale to perform image compression. To further understand the concept of neural networks, multiple articles on the topic were referenced and included in the literature review. Each and every major activation function that could be used in conjunction with the hidden layers and the output layer were also described in detail. Node Package Manager was used for managing modules and libraries, and Webpack was chosen as the bundler and the runtime for this project. Several participants were called for data gathering purposes, and different training epochs were experimented on. The paper demonstrates the capabilities of 6 activation methods in a regression model, namely Rectified Linear Unit (ReLU), Exponential Linear Unit (Elu), Rectified Linear Unit 6 (ReLU6), Scaled Exponential Linear Unit (SeLU), Mish and Swish. It critically analyzes the difference between each of them in relation to power consumption and provides adequate explanation for the overall results. All of the neural network models and machine learning related functions will be done in TensorflowJS. By capitalizing on the abundant amount of features available to TensorflowJS, this paper hopes to optimize neural network training by experimenting with the various settings available. This research paper ultimately aims to explore the possibility and feasibility of using a machine learning model to reduce power consumption, and discover the extent of effectiveness that the proposed solution can provide. In the end, the results obtained from this research showed that the proposed solution works under specific conditions. The neural network model only yields positive results when compressing 50 images and above in a single website, anything less than that could potentially result in more power consumed. The best activation method among ReLU, Elu, ReLU6, SeLU, Mish and Swish is Mish. Mish beat out the 2nd best activation method, SeLU by 0.3% (4.1% vs 3.8%) in terms of power saved.

Table Of Contents

List of Figures

List of Tables

1.0 Introduction

As technology gradually evolves, the realm of possibility increases along with it. The websites we have now are a far cry from the websites made back in the 90s. Websites have gone from being a wall of text to having eye popping images along with unique animations. And while the overall feel of websites have drastically improved over the years thanks to all this, one cannot help but to wonder, where is the drawback in all this? Are all of these fancy illustrations and designs truly free from problems?

1.1 Problem Statement

Although putting in pictures will definitely look better compared to a wall of text, the tradeoff is that there will be more things to load, which means the time taken to load increases. Thanks to recent breakthroughs in 5G and improvements to the network architecture as a whole, websites that are resource demanding to load will still be able to load without much of a problem [1]. However, is it safe to assume that everyone has access to 5G, much less acceptable internet speed?

This brings us to digital inequality. Digital Inequality is described as the difference in loading times caused by reliance on limited bandwidths and low-end devices, as well as internet bandwidth costing more in different countries [2]. In the research done by Chaqfeha et al., they found out that the same 100 webpages take up to 4 times longer in certain countries. Furthermore, cost-per-Gigabyte in certain countries goes up to 43 US Dollars, while certain countries provide 1 Gigabyte for only 8 cents. One last factor to consider is the fact that not everybody has access to the latest, state-of-the-art devices. Having an older phone model with lower processing power and older processing architecture causes JavaScript loading times to take a big hit [2].

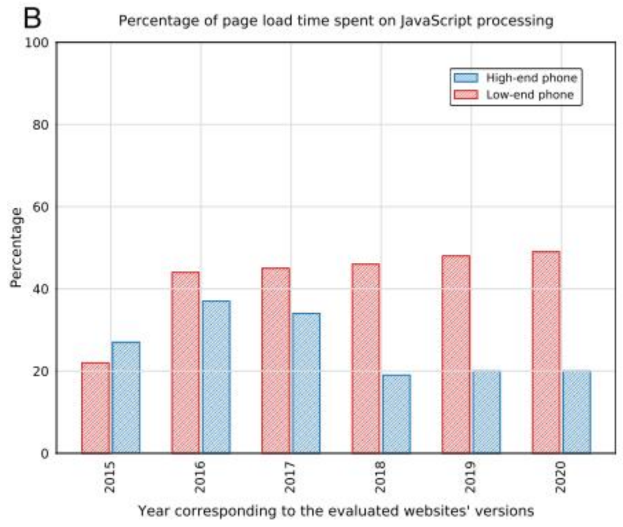


Figure 1 Amount of time spent on JavaScript processing [2]

According to the graph above, the difference just gets bigger and bigger as the years go by, which further increases the inequality. The results above suggest that Digital Inequality is indeed a very real and impactful phenomenon. With the advancement in coding languages and computers these days, it is indeed very tempting to add more and more features into websites. But while processing technology can indeed keep up, downloading the files and information itself is another issue. Digital Inequality has shown us that countries that are not as technologically advanced will take an unreasonable amount of time to load web pages. To make matters worse, cost-Per-Gigabyte in undeveloped countries also costs more than 10 times of the average amount [2]. With increasing details in a website comes increasing file size, which can lead to huge expenses on internet quota.

A case can be made with how much information can the eye actually perceive? When certain images are seen from afar, it can be perceived as crisp and clear with immaculate details. However, when seen up close, the illusion falls off and the eye discovers various flaws present within the image that was once thought to be perfect [3].

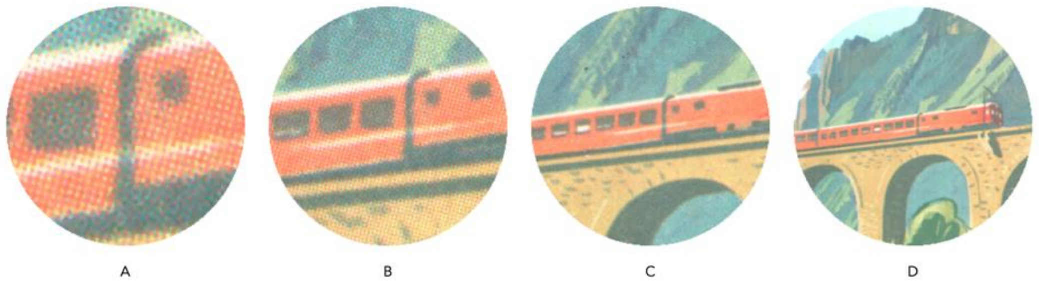


Figure 2: Artificially zoomed in images

Case in point, the 4 images above. Image A represents image D, but artificially zoomed in. Realistically, the human eye will never be able to discern the space between each pixel. This brings up a point. Humans do not perceive every detail of the image presented to them. This results in web pages over-rendering images in such detail to the point where it becomes redundant. To counteract this, the project will offer a neural network driven solution which serves to predict compression scales that suit each and every image size. Furthermore, the project will investigate the effects of different activation methods on the effectiveness of the neural network model, and how each activation method will impact the resulting power consumption.

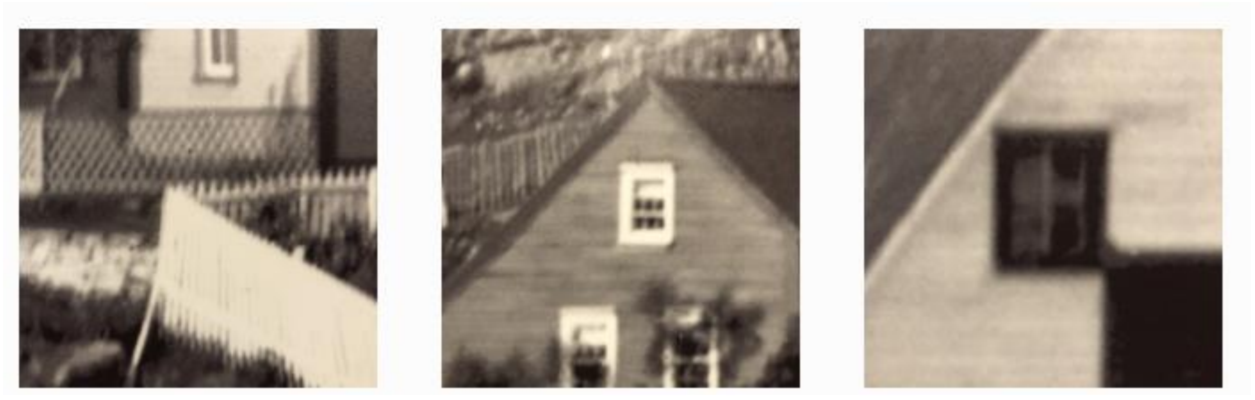


Figure 3: Extracts from a bigger image

The 3 images are extracted from a larger image. In the original image, the quality looks pristine. But when details are amplified and the 3 parts of the image are extracted, it can be clearly seen that certain parts are blurred.

1.2 Project Objectives

The problems regarding digital inequality and the problems stemming from it cannot be completely eradicated, but can be minimalized. A machine learning model will be trained to tackle that problem. The objective of the machine learning model is simple, it is required to:

- Reduce power consumption when loading pictures
- To find the best activation method for compression
- Lower power consumption without drastically impacting image quality

In order to achieve all the objectives, a machine learning neural network that can predict a scale for image compression was proposed as a possible solution. The idea is that the neural network will be trained with datasets regarding good compression scales for images based on their width and height. After being trained, the neural network will then scan the images in the website, compress it and then redisplay and redraw it on the website. To ensure that less power will be required to actually load the website, the model will intercept the images after the GET protocol. This way the original image will be loaded to allow for the model to make modifications, but it still will not drain power for the CPU to render it onto the webpage.

1.3 Project Scope

In this project, only 1 neural network type will be used and tested. This neural network type will be selected based on the suitability and which neural network type will produce the best result for the scale of compression. The code will only be configured for Windows 11, and the code will be fine tuned for Firefox due to Firefox having the required tools to perform power profiling. The project code is written in JS because of the nature of the work, JS is the most flexible language when it comes to interacting with a website and modifying pictures. This project also aims to find out how activation methods, different neuron values and different training iterations affect the output of the neural network, and which configuration will yield the most optimal result.

2.0 Literature Review

There are various techniques that we can employ to reduce the loading time of web pages. The goal is to utilize artificial intelligence

2.1 What is Artificial Intelligence (AI)?

AI is a machine given knowledge. However, that statement itself is a blur line. How do we make the distinction between intelligent and non-intelligent machines? How much intelligence is needed for us to properly say that the machine has artificial intelligence? In 1950, Turing devised a test called the Turing Test. This test suggested that under specific conditions, when a 3rd party is unable to discern between a machine and an actual human, then it is fair to conclude that both machine and human are equal [4].

The next question would be how does machine learning and artificial intelligence tie into each other? These two terms are always used interchangeably, and are commonly confused to have the same title and meaning. AI is said to be an intelligent agent [5].

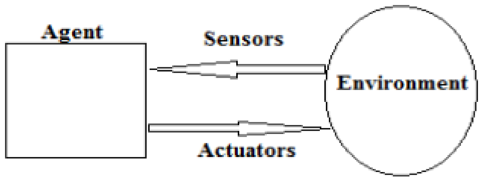


Figure 4: AI sensors

AI uses "sensors" to interact with the environment and react accordingly. The reaction is based on the actuators which is mapped to a function within an agent. The agent has a knowledge base and actions are determined from there. The knowledge base is updated when the agent interacts with the environment and this interaction continues back and forth [4] Machine learning on the other hand, is considered a field that grew out of AI. AI at its base is explicitly programmed to do something. Although AI still has some capacity to learn, it cannot do so without being explicitly programmed. Machine learning on the other hand can learn without being explicitly programmed. This means that machine learning does not require programmers to painstakingly code out every scenario, and does not require us to spoon feed everything to the machine [5]. A general learning algorithm is devised to learn, and the algorithm will analyze datasets to discern patterns and identify similarities that humans would otherwise be unable to do so.

The learning algorithm is the main part that makes machine learning work. Therefore it would make sense that there are variations on how the machine learns to ensure that the optimal goal can be reached. In this paper, we will only be looking at supervised and unsupervised learning as these are the 2 dominant learning techniques when dealing with web pages.

2.1.1 Supervised Learning

Supervised learning depends on comparing the output with an expected output. The learning factor here is when the algorithm adjusts parameters in order to achieve the expected output [4]. For example, we could use machine learning to enhance images, but it could instead blur it. Therefore, we must ensure the algorithm knows exactly what kind of image is expected so that it can find out the appropriate methods to achieve the goal.

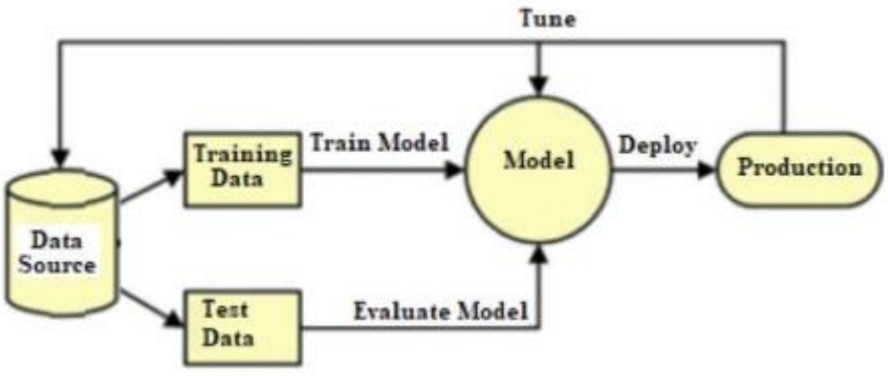


Figure 5: Supervised learning flow

Supervised learning works by splitting the data source into two types of data. One being the training data and the other being the test data. Using the training dataset, the model will then try to identify a pattern and infer a function. This function is then cross checked with the test data to evaluate the model [6]. The more times the model trains and iterates, the more accurate it will become. However, there is a certain breakpoint for training cycles, if the model undergoes too many iterations of training, the issue of overfitting might present itself.

2.1.2 Unsupervised Learning

Unsupervised on the other hand is purely letting the algorithm analyze data sets on its own. This learning method is commonly used to find patterns within large amounts of data sets. The data in this method is normally divided into clusters and is also known as a clustering algorithm [4]. Unsupervised learning is used to discover similar examples within the data, which is how it determines patterns and similarities. As such, in the process of discovering patterns and similarities, the data will inevitably be clustered together since only similar data will produce patterns.

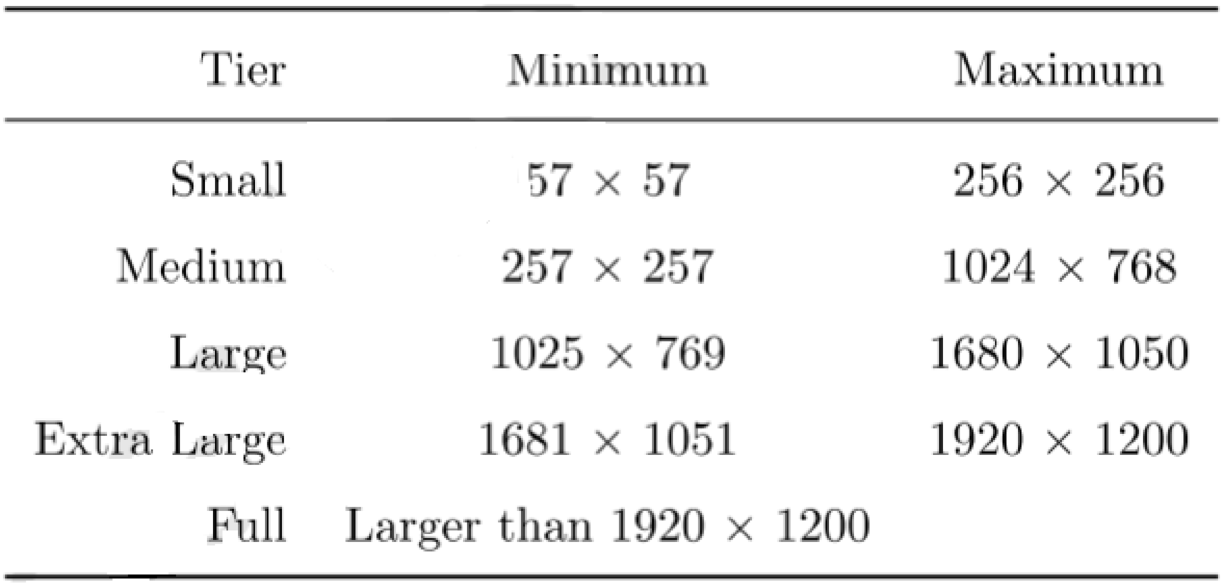
In unsupervised learning, there are no correct answers as well. The algorithm is left to discover and learn interesting patterns. Unsupervised learning is mainly used for clustering and feature reduction [7]. Notable types of unsupervised learning include K-means clustering and Principal Component Analysis

2.2 Web Energy Efficient Computing (WEEC)

The first main problem that WEEC tackles is to reduce the problem of images. To first determine which part of the website contains images, the AI model scans the DOM and locates HTML image tags. By using Intel CPU's Running Average Power Limit (RAPL) energy sensors, we can determine whether WEEC has an actual positive effect on the result [1].

Next, the format of the image is analyzed. Images are categorized as small tier, medium tier and full tiers. Small tiers remain unchanged. Medium to full tiers is where the focus is at. Researchers found out that JPG images consume significantly lower levels of energy compared to PNG, so image formats are changed.

The second step is to downsample images according to their tiers. Downsampling scale lies from 0-1. Downsampling uses a linear regression model.



Tier	Minimum	Maximum
Small	57 × 57	256 × 256
Medium	257 × 257	1024 × 768
Large	1025 × 769	1680 × 1050
Extra Large	1681 × 1051	1920 × 1200
Full	Larger than 1920 × 1200	

Figure 6: Image tier determination criteria

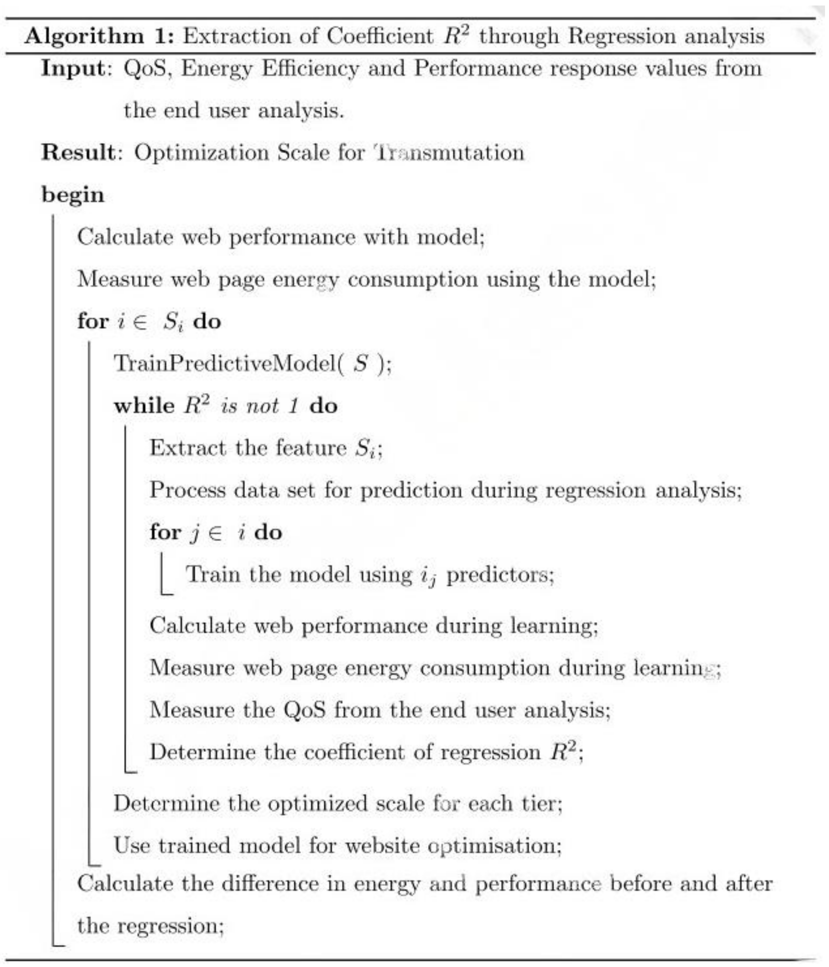
Small images have a downsample scale of 0.8. Medium images are downsized by up to 0.9. Large images are downsized by up to 0.6 and extra large images are downsized to the scale of up to 0.7. Full images are downsized by up to 0.6 [1].

The final step would be to take the downsized and downsampled images and put them back into the web page.

WEEC uses a predictive model to determine which downsampling model and which coefficient to use. The model uses regression analysis to process the data set. The model extracts the features in the web page such as images, and determines the regression coefficient. Regression analysis is performed by using mathematical analysis to figure out which variable matters. The variable at play here is the power consumed and the variable manipulated is the regression coefficient. The web performance is first calculated, then the energy consumption is measured. The end user's quality is also measured in accordance with the changes in the coefficient [1]. The model ensures that the best quality is served to the user while energy consumption is still kept to a minimum.

This process is repeated for each tier of images mentioned above (small, medium, large, extra large and full). The training data is achieved using a self developed extension in the chrome browser.

This extension is responsible for extracting the DOM tree and interpreting the HTML tags containing images. The extension then re-renders the web page by applying the downsampling techniques using regression coefficients determined from the model.



```
Algorithm 1: Extraction of Coefficient $R^2$ through Regression analysis
Input: QoS, Energy Efficiency and Performance response values from
       the end user analysis.
Result: Optimization Scale for Transmutation
begin
    Calculate web performance with model;
    Measure web page energy consumption using the model;
    for $i \in S_i$ do
        TrainPredictiveModel( $S$ );
        while $R^2$ is not 1 do
            Extract the feature $S_i$;
            Process data set for prediction during regression analysis;
            for $j \in i$ do
                Train the model using $i_j$ predictors;
            Calculate web performance during learning;
            Measure web page energy consumption during learning;
            Measure the QoS from the end user analysis;
            Determine the coefficient of regression $R^2$;
        Determine the optimized scale for each tier;
        Use trained model for website optimisation;
    Calculate the difference in energy and performance before and after
    the regression;
```

Figure 7: Algorithm Pseudocode [1]

The effectiveness was measured with respect to energy efficiency, power utilization and load time. Energy efficiency was calculated by comparing the energy consumed before and after WEEC was applied. The algorithm makes adjustments to the regression coefficient in accordance to the energy consumption as well as the quality of service and how users respond to it.

The model was trained using a data set generated from end user analysis. The dataset was gathered via manual methods such as by surveying various web users with varying skills and age. Questionnaires were used to aid in gathering data. The responses were then compiled to train the

Linear Regression Model [1]. The Linear Regression Model then predicted the downsampling scale for each image dimension.

$$Energy\ Efficiency\ (\eta) = \frac{O_{av}}{R_{av}} \times 100\%$$

Next, power utilization was calculated by finding out the energy used per unit time

Lastly, average load time is calculated using this equation

$$Average\ Load\ Time = \frac{\sum_{i=1}^{n} l_i}{n} \qquad Power = \frac{E_c}{t}$$

The results of how the algorithm affected web page loading and power consumption efficiency is as such:

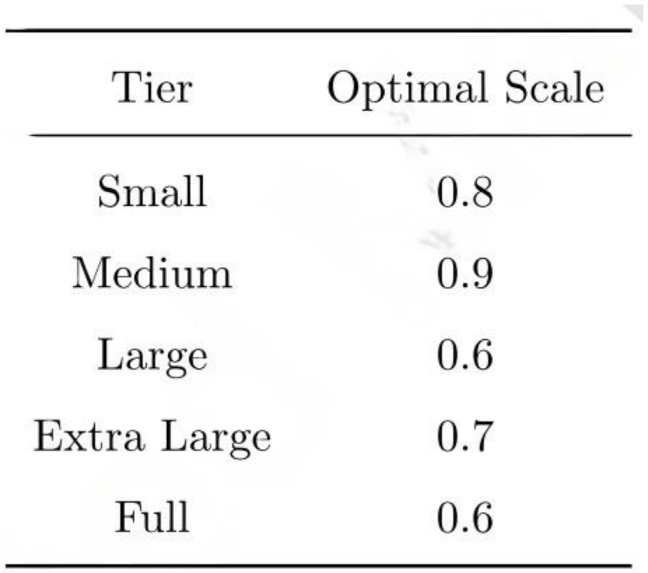
Image Tier	Scale	Energy Efficiency	Quality Of		Overall UX	Load Time	
			Image	Service		Image	Page
Small		92%	0.15	0.18	0.19	0.30	0.44
Medium		90%	0.19	0.20	0.22	0.33	0.48
Large		86%	0.30	0.31	0.28	0.50	0.55
Extra Large		84%	0.34	0.35	0.29	0.50	0.62
Full		85%	0.38	0.36	0.30	0.60	0.63
Small		91%	0.18	0.19	0.30	0.36	0.50
Medium		90%	0.22	0.24	0.31	0.38	0.52
Large	0.2	87%	0.32	0.33	0.32	0.55	0.60
Extra Large		82%	0.36	0.37	0.44	0.60	0.68
Full		84%	0.40	0.39	0.40	0.62	0.69
Small		91%	0.2	0.21	0.41	0.44	0.60
Medium		88%	0.25	0.26	0.44	0.49	0.60
Large	0.3	85%	0.38	0.40	0.41	0.60	0.67
Extra Large		83%	0.42	0.45	0.50	0.66	0.74
Full		82%	0.48	0.50	0.60	0.69	0.75
Small		90%	0.32	0.36	0.49	0.50	0.65

Medium	0.4	89%	0.33	0.35	0.50	0.52	0.63
Large		85%	0.50	0.61	0.76	0.68	0.72
Extra Large		84%	0.52	0.54	0.62	0.72	0.79
Full		82%	0.56	0.61	0.72	0.76	0.80
Small	0.5	88%	0.45	0.48	0.62	0.55	0.68
Medium		86%	0.39	0.40	0.61	0.58	0.65
Large		84%	0.68	0.82	0.85	0.72	0.78
Extra Large		82%	0.62	0.61	0.74	0.78	0.82
Full		80%	0.62	0.72	0.80	0.80	0.83
Small	0.6	80%	0.52	0.55	0.74	0.62	0.70
Medium		85%	0.46	0.48	0.72	0.65	0.72
Large		84%	0.78	0.92	0.96	0.90	0.84
Extra Large		83%	0.76	0.70	0.80	0.82	0.85
Full		79%	0.84	0.90	0.94	0.90	0.86
Small	0.7	83%	0.64	0.66	0.80	0.86	0.80
Medium		82%	0.59	0.62	0.84	0.78	0.76
Large		80%	0.80	0.94	0.86	1.00	0.90
Extra Large		79%	0.80	0.84	0.92	0.90	0.92
Full		76%	0.89	0.90	0.90	1.00	0.94

Small	0.8	80%	0.77	0.85	0.98	0.90	0.85
Medium		78%	0.62	0.76	0.90	0.82	0.79
Large		72%	0.86	0.95	0.80	1.00	0.92
Extra Large		65%	0.85	0.85	0.82	1.00	0.94
Full		62%	0.92	0.92	0.85	1.00	1.00
Small	0.9	76%	0.79	0.86	0.86	1.00	0.95
Medium		72%	0.74	0.86	0.96	0.90	0.82
Large		68%	0.88	0.95	0.77	1.00	0.95
Extra Large		62%	0.89	0.85	0.80	1.00	0.98
Full		59%	0.95	0.92	0.82	1.00	1.00

Table 1: Table of data based on regression coefficient

According to the table, we can see how each coefficient affects the power consumption and web page loading time. With this, we can easily extract the best scale for each image tier with respect to energy consumption, load time and quality.



Tier	Optimal Scale
Small	0.8
Medium	0.9
Large	0.6
Extra Large	0.7
Full	0.6

Figure 8: Optimal scale for each image tier [1]

However, it is important to mention that this is just the best fit line generated from regression analysis.

2.3 Linear Regression

Before talking about linear regression, we must first talk about regression itself. Regression is a technique used for forecasting and prediction. By using linear regression, the model can make predictions based on the input data. Secondly, regression allows the model to determine patterns from independent and dependent variables [7].

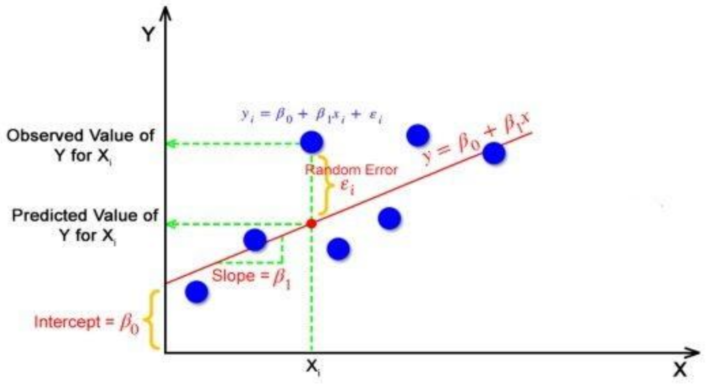

Y
$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$
Observed Value of Y for X_i
Random Error ε_i
$y = \beta_0 + \beta_1 x$
Predicted Value of Y for X_i
Slope = β_1
Intercept = β_0
X_i
X

Figure 9: Visualization of linear regression

Regression model tries to find a line of best fit according to input data. The value for Y is the dependent value while X is the independent value. The model tries to predict the value of Y for every X. Of course, the prediction will never be 100% accurate, hence we have the term “random error”. Random error depicts the inaccuracy between the predicted data and the actual data. Linear regression is a part of regression analysis. Regression analysis comprises of multiple types of regression, a few examples include Simple Linear Regression, Multivariate Linear Regression, Polynomial Regression.

2.3.1 Simple Linear Regression

Simple Linear Regression deals with a single independent variable. Simple Linear regression will then define the dependence of the variable. In short, Simple Linear regression distinguishes the influence of independent variables [7].

2.3.2 Multivariate Linear Regression (MLR)

MLR is another regression technique that is used to predict the result of an answer variable using other explanatory variables [7].

2.3.3 Polynomial Regression

Polynomial Regression is a type of regression analysis that is a special case of MLR. The Polynomial equation generated from the model will have a curvilinear shape between the dependent and independent [6].

2.4 Deep Learning

2.4.1 Convoluted Neural Network

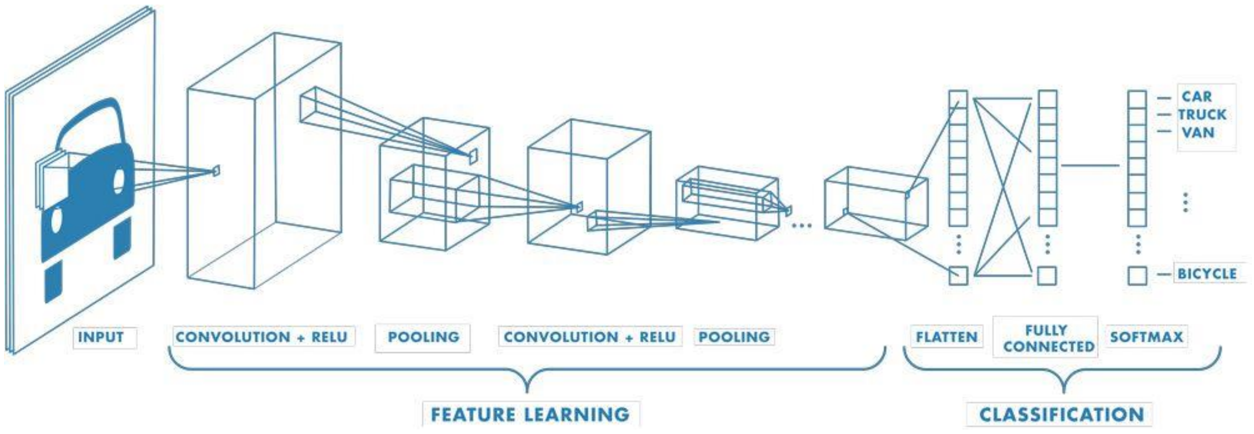


Figure 10: Convoluted Neural Network breakdown

Convoluted Neural Network comprises of convolutional layers. At each convolutional layer, the input data is convolved with a set of K kernels. The kernels are also known as receptive fields [10]. Unlike other neural networks, CNN has shared weights and bias values which are the same for all the neurons in a given layer. Convoluted Neural Network is generally used for detecting images. Finally, after learning the features throughout the many layers, CNN will shift to classification at the end. The classification layer will determine the final output [11].

2.4.2 Recurrent Neural Network

Recurrent Neural Network (RNN) involves an input sequence, recurrent units, hidden state and finally the output.

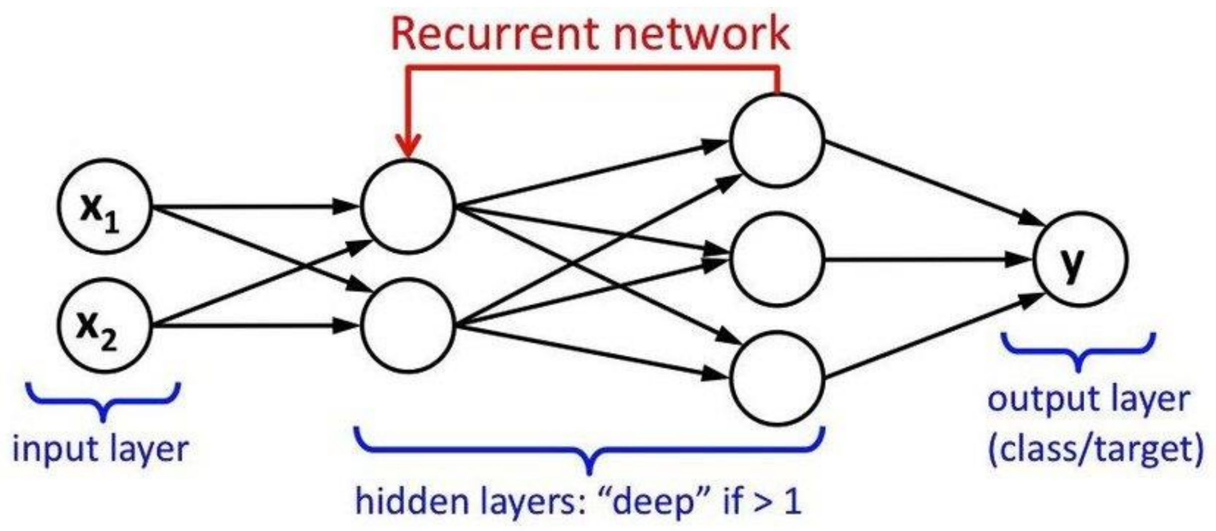


Figure 11: Recurrent Neural Network breakdown

The recurrent units process the input sequence while maintaining the hidden states. For each element in the input sequence, the recurrent unit takes both the current input and the previous hidden state as inputs and produces an output while updating the hidden states. The hidden state contains information about the previous elements in the input sequence, by referencing the hidden state, it allows the network to capture patterns and dependencies across time [12]. After going through the whole recurrent neural network and the hidden layers, the output is produced.

In theory, different types of deep learning can be used for the Keras model, and the different results yielded from the deep learning types can then be compared with each other to find out which deep learning technique yields the best result.

2.5 Sequential Model

Sequential model in Tensorflow is mainly used for inputting one data and outputting one data. In this case, a sequential model is not recommended to be used due to the fact that the use case for the training model requires the input of the dimensions of the image, namely width and height. This results in having 2 input data to output a predicted "scale" of compression. Although not impossible to do so, the sequential model handles this poorly.

The sequential model consists of layers that are stacked linearly. The reason it is called sequential is because the layers are built in a step by step manner.

When forcing the dense layers to use an input shape of 2 to allow for 2 input data, an occurrence called overfitting happens. Overfitting is defined as the model adhering closely to the input dataset, where data that are extremely close to the dataset will be predicted with 100% accuracy, but anything else that does not conform to the given dataset will fail horribly when trying to generate a prediction. To summarize, overfitting is defined as an analysis that corresponds too closely with the given dataset, and will fail to predict anything outside of the known verse [15].

2.6 Keras (Functional) Model

Keras model is also referred to as a functional model. It is a model available in Tensorflow, but originally derived from Keras. This model is majorly different from a sequential model. The first main difference is that the layers in a functional model are not declared sequentially. Functional model provides more features, more customization and is also connected differently. It allows for more than one input. The report will explain both sequential and functional models in detail in the methodology section below.

2.7 Neural Network types

Sequential and Functional models are just ways that the models can be built. In other words, the main difference between the two is just how the layers are connected. We can take this a step further in deciding the type of layers to use and what we decide to do with it. In other words, the structure of the neural network can be modified to suit multiple purposes. The main 9 types of neural networks structures are: Perceptron, Feed Forward Neural Network, Multilayer Perceptron, Convolutional Neural Network, Radial Basis Functional Neural Network, Recurrent Neural Network, LSTM – Long Short-Term Memory, Sequence to Sequence Models and Modular Neural Network.

2.7.1 Perceptron

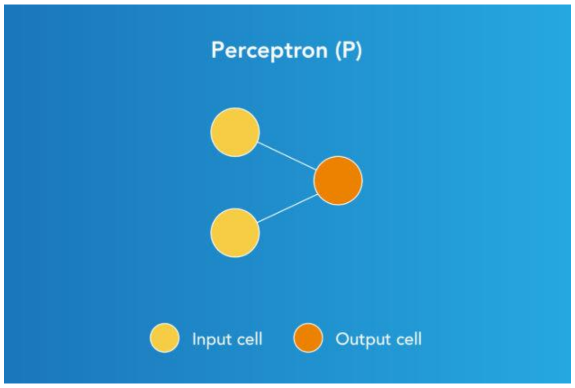


Figure 12: Perceptron visualization

The Perceptron model is one of the oldest and simplest Neurons and was proposed by Minsky and Papert. The Perceptron model is the smallest neural network and is only useful in solving specific linear problems. This is due to the fact that there are no hidden layers. Without the presence of hidden layers, the neural network will not be able to learn patterns and will inevitably fall into linearity [16]. Realistically, a perceptron model is not even considered a neural network due to it missing hidden layers and only being able to learn linear patterns. A perceptron is only used in logic gates like AND, OR or NAND gates.

2.7.2 Feed Forward Neural Networks

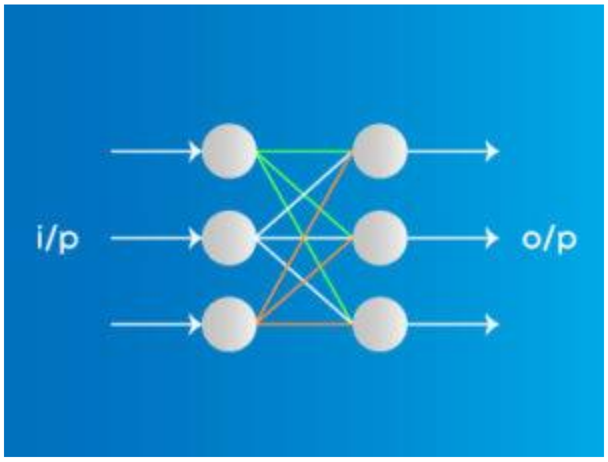


Figure 13: Feed Forward visualization

Feed forward neural network is an advancement of the perceptron model. In the feed forward neural network, hidden layers may or may not be present. It is in feed forward neural networks that we start to see some resemblance to the modern day neural networks like CNN or RNN. However, unlike modern day neural networks, feed forward neural networks can only pass information forward. No form of back propagation is present in feed forward [17]. Feed forward neural networks are normally used for simple classification

2.7.3 Multilayer Perceptron

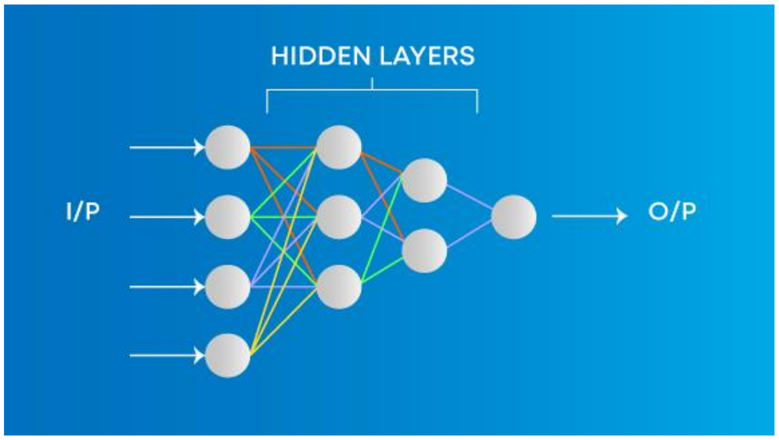


Figure 14: Multilayer Perceptron visualization

Multilayer perceptrons are the start of complex neural networks. In this neural network, there are various layers of neurons as well as the guaranteed presence of hidden layers. Inside a multilayer perceptron, weights are also factored in and backpropagation, where information flows in multiple directions, is also considered during the calculation of the output. All the nodes are also connected to the next layer which classifies multilayer perceptrons as a fully connected layer as well.

2.7.4 Convolutional Neural Network

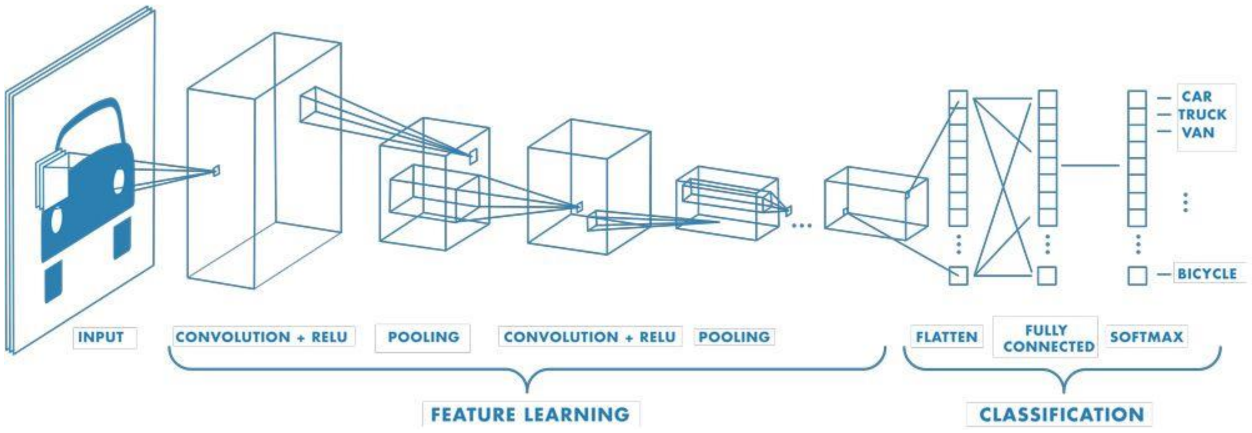


Figure 15: Convoluted Neural Network visualization

Convoluted Neural Network comprises of convolutional layers. At each convolutional layer, the input data is convolved with a set of K kernels. The kernels are also known as receptive fields [10]. Unlike other neural networks, CNN has shared weights and bias values which are the same for all the neurons in a given layer. Convoluted Neural Network is generally used for detecting images. Finally, after learning the features throughout the many layers, CNN will shift to classification at the end. The classification layer will determine the final output [11].

2.7.5 Recurrent Neural Network

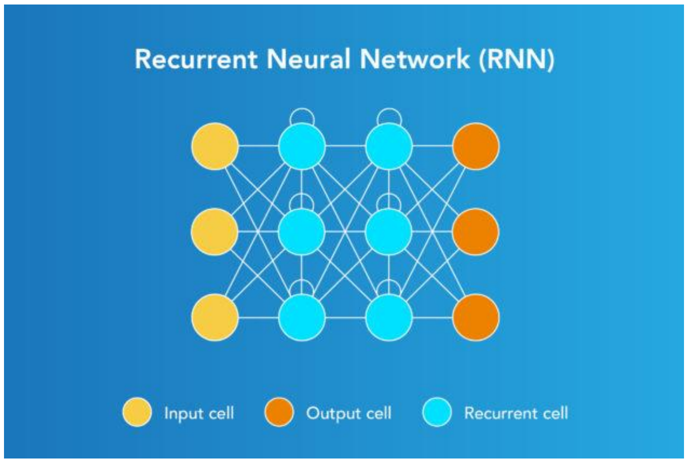


Figure 16: Recurrent Neural Network breakdown

The recurrent units process the input sequence while maintaining the hidden states. For each element in the input sequence, the recurrent unit takes both the current input and the previous hidden state as inputs and produces an output while updating the hidden states. The recurrent neural network will retain information from previous iterations, and forward propagation is utilized when the prediction is wrong. Recurrent Neural Network also utilizes backward propagation in conjunction with forward propagation to correct errors [18].

2.7.6 Long-Short-Term-Memory (LSTM)

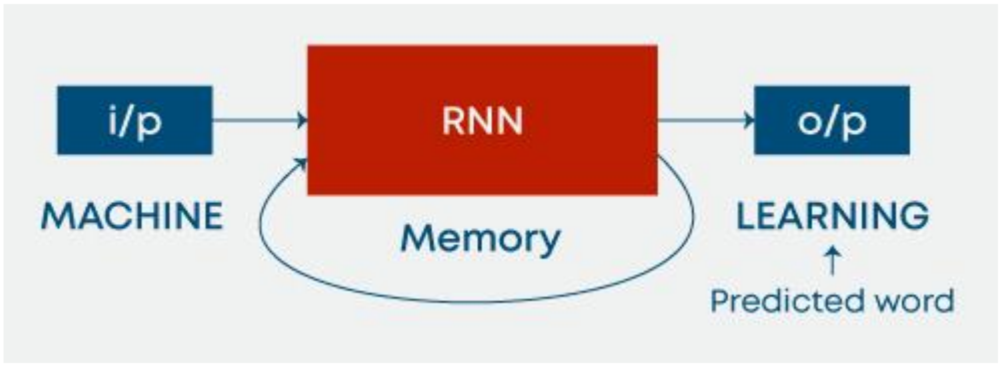


Figure 17: LSTM visualization

LSTM is a type of RNN that includes a memory cell to maintain information for long periods of time, hence long short term memory. Additionally, a separate set of systems are in place to dictate when information should be retained and forgotten.

2.7.7 Sequence to sequence models

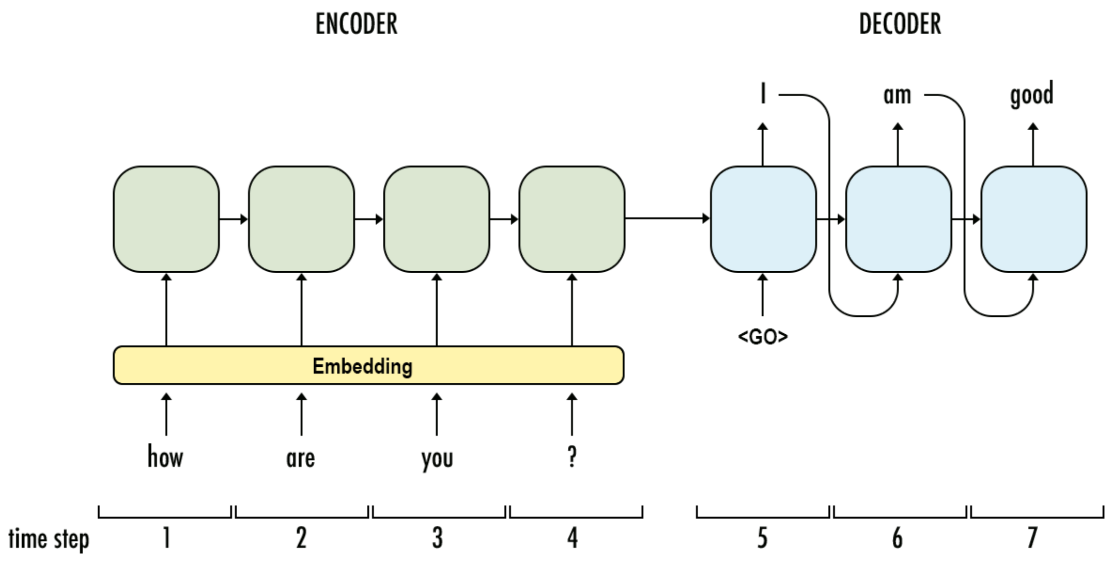


Figure 18: Sequence to sequence model visualization

Sequence to sequence model is a combination of two Recurrent Neural Networks, in this model, an encoder and decoder is present. The encoder is used as the input section and the decoder is used for the output section. The encoders and decoders can have the same or different parameters. Sequence to sequence models are normally used in chat bots.

2.7.8 Modular Neural Network

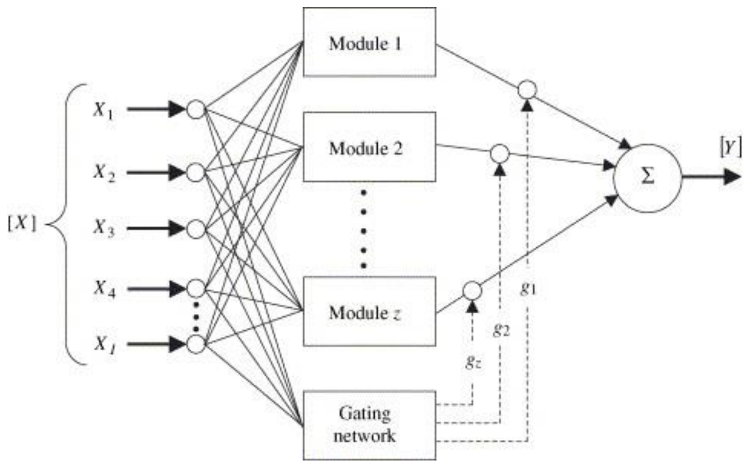


Figure 19: Modular Neural Network visualization

A modular neural network is a network that uses multiple neural networks to achieve the goal. These neural networks work independently and do not communicate or share information with each other. This allows for large complicated tasks to be split up and delegated to different networks, allowing for faster speed [19].

2.8 Structure of Choice

After listing down all the major neural networks, it was time to make a decision on which network will be chosen. By process of elimination, Perceptron is too simple and will not be able to yield any meaningful results. Convolutional Neural Network on the other hand is generally used for classification. The project requires regression since we are dealing with numbers and scales. Recurrent Neural Network could be used and is theoretically suitable for the nature of this experiment, however due to its complicated steps and procedures, it was ultimately decided against using it. This also means that all RNN related models were not selected to be used.. Therefore we reach the final conclusion that the Multilayer Perceptron Neural Network and Feed Forward Neural Network will be both chosen. Multilayer Perceptron is simple enough, yet offers features like hidden layers and backward propagation, which prevents linearity in the model. Feed Forward will be used in conjunction with Multilayer Perceptron to ensure proper flow. Settings like neuron amount, activation methods and hidden layers can also be manually tweaked to experiment for optimality.

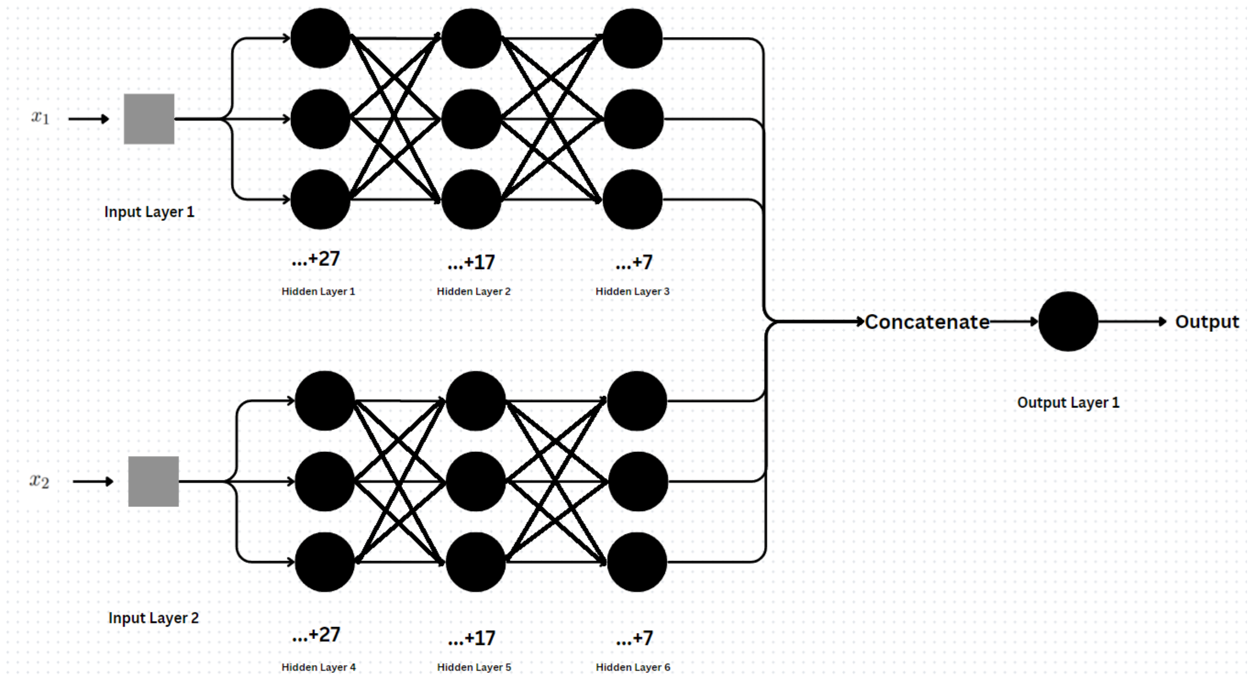
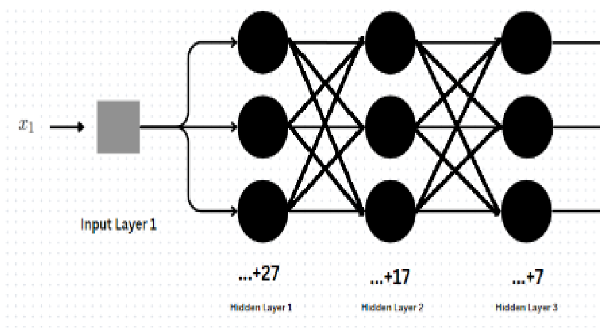
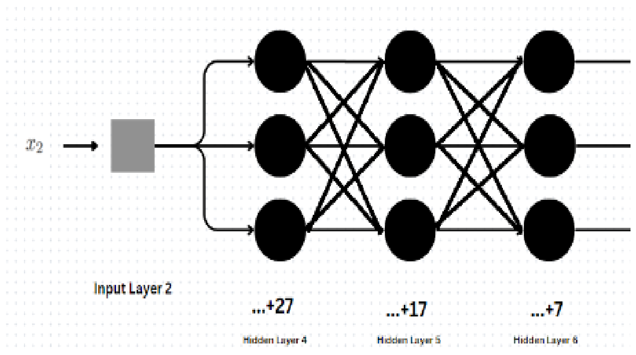


Figure 20: Overall visualization of the neural network

The overall structures of the layers can be represented by the image above. The functional model is derived from the Keras API. It is designed to handle multiple inputs by using non-linear topology as well as sharing layers [20]. Within the neural networking layers, multiple activation functions can be used.

The whole model itself is a Feed Forward Neural Network, but within the model, parts of it are Multilayer Perceptrons.





The hidden layers that respectively belong to each of the layers before concatenation are fully connected, and also utilize backpropagation to gain knowledge from the dataset. This defining feature is what separates these two specific parts from the other sections of the neural network. The information from these 2 layers are processed and then passed down for concatenation before being entered into the output layer. This part of the process is considered feed forward since there is no involvement of backpropagation in this step [21].

2.9 Activation Functions

There are 13 main activation functions available for the Keras API right now: elu, hardSigmoid, linear, mish, relu, relu6, selu, sigmoid, softmax, softplus, softsign, swish and tanh. Each of these activation functions will affect the layers in one way or the other.

2.9.1 Backpropagation

Before going into activation functions, backpropagation needs to be mentioned first. Backpropagation is a technique that is commonly used in neural networks and affects how neural networks learn. Backpropagation is a technique used for supervised learning neural network models. It is broken down into 5 steps: Forward pass, Compute loss, Backward pass, Updating weights and lastly Iterating the process [21].

Forward pass involves feeding the input through the neural network layer by layer, and each neuron in the network will perform a weighted sum of its input, the activation function of the neuron in the layer will also be applied when weighting the input. The input is then passed along to the next layer.

Computing losses is when the model measures the accuracy of the output against the input. The error between the two values is the loss value that is calculated. Inside this section, there are normally two different types of loss functions that the neural network will use, the first one is mean-squared error while the second is cross-entropy. If the problem is regression in nature, then mean-squared error is used, and if it is a classification type problem, cross-entropy is used.

Backward pass is the step where the model backtracks to adjust the weights of the neurons in the network. The model calculates the gradient of the data losses with respect to the weights. It then adjusts the weights of the network overall according to the calculation done.
When adjusting the weights, the model does it in the opposite direction of the layers to reduce losses. Weight adjustment is done according to learning rate. Learning rate dictates how fast the weights are adjusted by determining how big of a step the model will take when learning. The gradient previously calculated is scaled according to this learning rate before the update.

Lastly, the model will iterate through steps 1 to 4 for multiple epochs until the network starts converging.

2.9.2 Relu

ReLU activation is currently one of the most famous activation functions right now. ReLU works as an identity for positive arguments and zero otherwise [22]. The main benefit of ReLU is that it diminishes the effect of the famous vanishing gradient problem as well as correcting for bias shifts.

$$f(x) = x^{+} = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \quad f'(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x < 0. \end{cases}$$

Figure 21: Activation function of ReLU

Above is the formula for ReLU. The function will calculate the input and classify the output according to the criterias set [23]. ReLU is unbounded in the positive domain, but bounded to 0 when it attempts to dive into negative domains. ReLU can still be used as an activation function for regression problems, but only in the hidden layers.

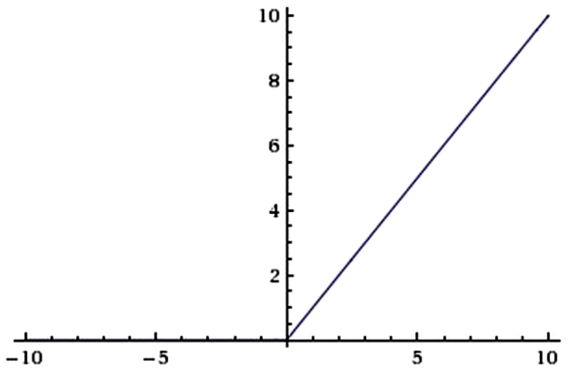


Figure 22: Graph of ReLU activation function

If ReLU is used in the output layer, all negative outputs will end up turning into zero because of the nature of ReLU. This issue is called the dying ReLU problem and is a form of the vanishing gradient issue [24].

2.9.2.1 Vanishing Gradient problem

The Vanishing Gradient Problem exists because of backpropagation and other gradient based learning methods employed by neural networks. When traversing between layers and learning, neural networks will update weights that are proportional to the partial derivative of the loss function and multiply it with the current weight [25]. The problem comes in when both the loss function value and the current weight have low values. When a small value is multiplied with another small value, it will ultimately result in a smaller value. This smaller value will then affect future weight values as the model continues learning, resulting in a domino effect that leads to the gradient eventually vanishing into the sea of decimal points. Hence, the model will eventually cease to be able to properly learn due to not having a gradient, which results in the famous vanishing gradient problem [25].

2.9.2.2 Bias Shifts

Bias shifts on the other hand occur due to the non-zero activation of a method. Since ReLU is non-zero and identifies positive arguments, it results in a mean activation of more than 0. This phenomenon will cause a sort of bias when traversing to the next layer, and when more units are correlated, bias shift will occur. When bias shift occurs, the overall gradient of the training function will be impacted and learning rate will decrease [22]. To account for this, the weight updates in

ReLU must be adjusted when traversing through the layers to ensure that the gradient can be adjusted to match a more natural gradient [22].

2.9.3 Elu

Elu activation function, also known as Exponential Linear Units, is said to be a better form of ReLU. ReLU is known for causing bias shifts as training goes on due to it having a mean activation of above zero [22]. Elu on the other hand diminishes this problem by introducing negative values. By introducing negative values, it will allow for a mean value closer to zero, which lessens the effect of bias shift and ultimately results in a more natural gradient which is beneficial for a good learning rate of the model. Elu has a smooth profile, bounded ot -1 in the negative domain, and unbounded in the positive domain.

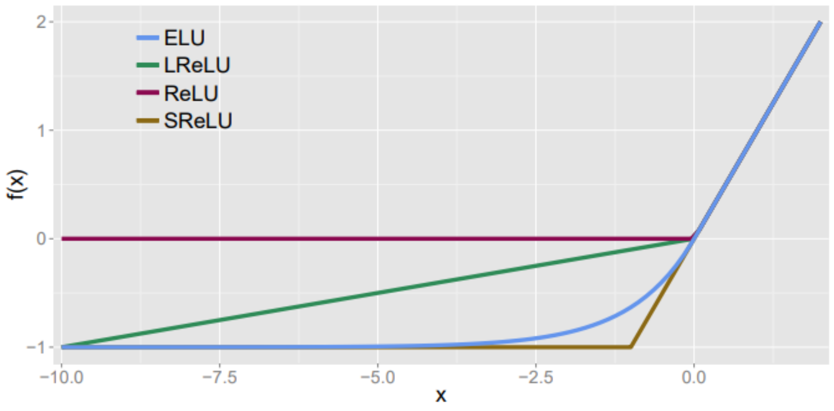


Figure 23: 4 types of activation methods at training rate = 0.1

The image above depicts 4 activation methods: Exponential Linear Units (ELU), Leaky Rectified Linear Unit (ReLU), Rectified Linear Unit (ReLU) and shifted Rectified Linear Unit (SReLU). As we can see, only Elu and SReLU have values at the negatives, which conforms to their ideology of trying to avoid bias shiftings.

$$\mathrm{ELU}(x) = \max(0, x) + \min(0, \alpha(e^x - 1))$$

Figure 24: Formula of Elu activation function

Elu can also be used for regression.

2.9.4 Selu

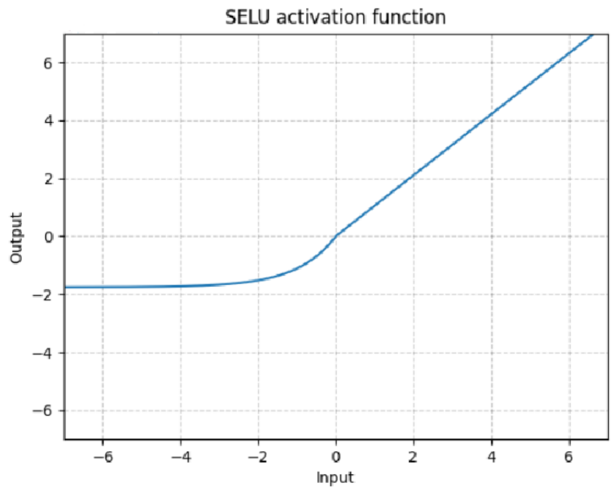


Figure 25: Selu activation graph

SeLU stands for scaled exponential linear unit. This activation function is unique in the sense that it injects self normalization into the model itself [24]. Since SeLU accepts negative values, it will have a mean that is closer to zero. This allows for SeLU to converge faster even if there is noise present in the dataset. SeLU works particularly well in feed forward type neural network models [24].

$$\gamma(\max(0, x) + \min(0, \alpha(e^x - 1)))$$

Figure 26: Formula of SeLU

SeLU is also unbounded in the positive domain, and also accepts negative values to a certain extent. SeLU can also work for regression

2.9.5 Relu6

ReLU6 is similar to ReLU in many ways, the only defining difference lies in the number 6.

$$f(x) = x^+ = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \quad f'(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x < 0. \end{cases}$$

Figure 27: Activation function of ReLU

In ReLU6, max(0,x) is changed to max(0,6). This means that ReLU6 is a modified version of ReLU where the activation function itself is hard limited to 6.

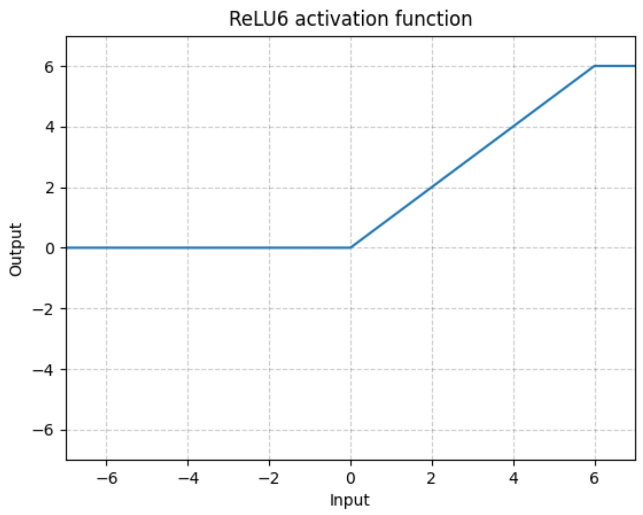


Figure 28: Graph of ReLU6

The benefits of ReLU6 is that it is a faster activation method to execute. It is all thanks to the fact that the function is limited to 6. Although it does impact accuracy and training losses, it will be faster. In a research done by Nworu et al., they attempted to find out how different versions of ReLU would impact image classification. The result was that ReLU had a better training accuracy 0.9747 vs 0.9705 and test loss at 0.5647 vs 0.9527, but ReLU6 had a better test accuracy 0.781 vs 0.7772 and training loss at 0.0844 vs 0.1036. The big difference comes down to execution time, ReLU has an execution time of 3184.74 seconds whereas ReLU6 has an execution time of 2828.26 seconds. This solidifies the idea that ReLU6 is a more lightweight activation method due to its faster execution time, but the accuracy suffers immensely [26].

2.9.6 Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{1+e^{x}} = 1 - \sigma(-x).$$

Figure 29: Sigmoid activation formula

The sigmoid activation function is a mathematical function that is characterized by having a curve in its shape. Sigmoid activation functions have a domain of all real numbers and will return a value between 0 to 1 or 1 to -1 [27].

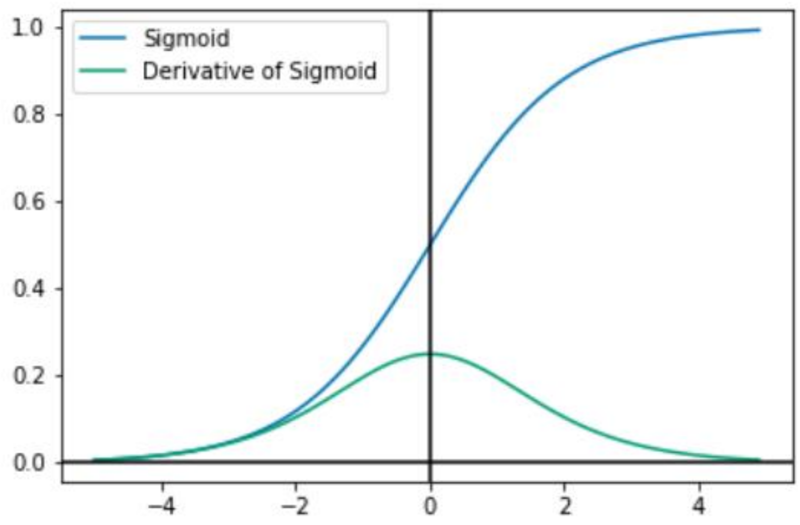


Figure 30: Sigmoid activation curve and its derivative

Sigmoid functions are commonly used in classification and logistic regression problems [27]. This means that sigmoid is not usable for regression problems where a continuous range of data is required. Logistic regression problems are problems that only output two values like true/false or yes/no. It also means that sigmoid activation functions are only capable of outputting two numbers: 1 or 0, or in some cases, -1 [28] . Sigmoid also suffers from vanishing gradient problems when being used in hidden layers with backpropagation [29]. The reason why Sigmoid is normally used for logistic regression and classification is due to the fact that Sigmoid normally exists between 0 to 1. This characteristic is unsuitable for regression of continuous values.

2.9.7 hardSigmoid

$$f(x) = \max\left(0, \min\left(1, \frac{(x+1)}{2}\right)\right)$$

Figure 31: hardSigmoid activation function

The hardSigmoid activation function falls under the same region as ReLU6, where it is a lightweight version of the original. Instead of using e^x, hardSigmoid uses only x values, which lowers computational resources and makes computational time faster [30]. Similarly to sigmoid, hardSigmoid is also only used in classification and logistic regression problems.

2.9.8 Linear

A linear activation function prevents neural networks from being flexible.

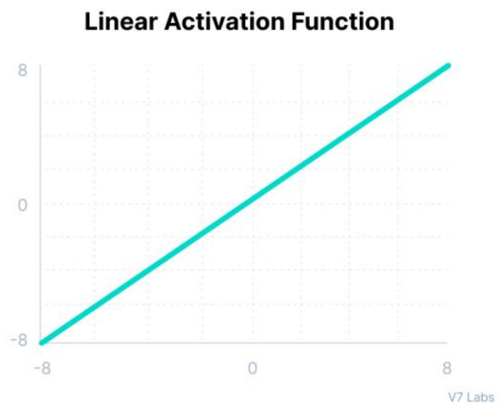


Figure 32: Linear activation graph

A hidden layer uses activation functions to learn and allow it to predict data. All of these data cannot be easily seen or understood by humans since it is encoded in the form of weights [31]. The activation functions are one of the factors affecting the weights. When a linear activation function is used, it prevents the hidden layers from being able to make any adjustments. The model is forced to conform to a straight line and make prediction values along those lines. This can be fine when the problem is simple. In short, if a linear activation function is used, the entire neural network model will just end up being a linear regression model. The formula for a linear activation function is f(x) = x. Even if linear activation is used, it is only used in the output layer.

2.9.9 Mish and Swish

The Mish and Swish activation function is an activation function that is relatively new. Mish and Swish were proposed to be an upgrade above ReLU activation methods. It was proposed to solve issues caused when negative weights are present. Mish and Swish was also proposed to solve the vanishing gradient problem caused by ReLU and other activation functions.

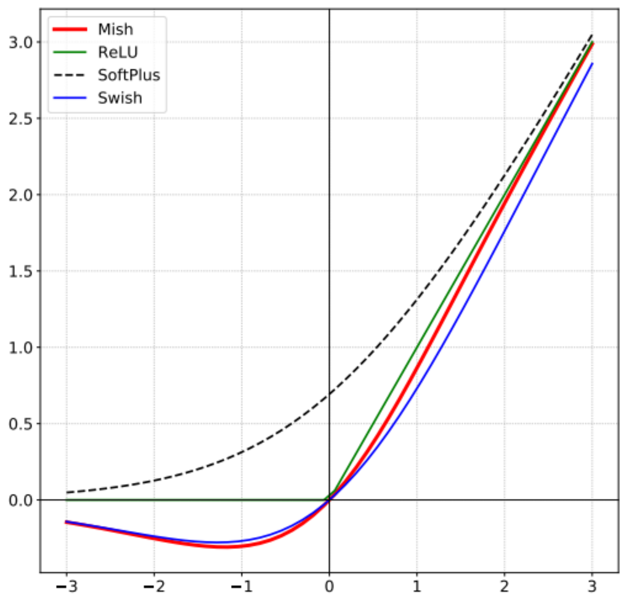


Figure 33: Activation function of Mish vs ReLU vs SoftPlus vs Swish

In the picture above, we can see that ReLU and SoftPlus do not consider values below 0 at all. Whereas Swish and Mish considers these values.

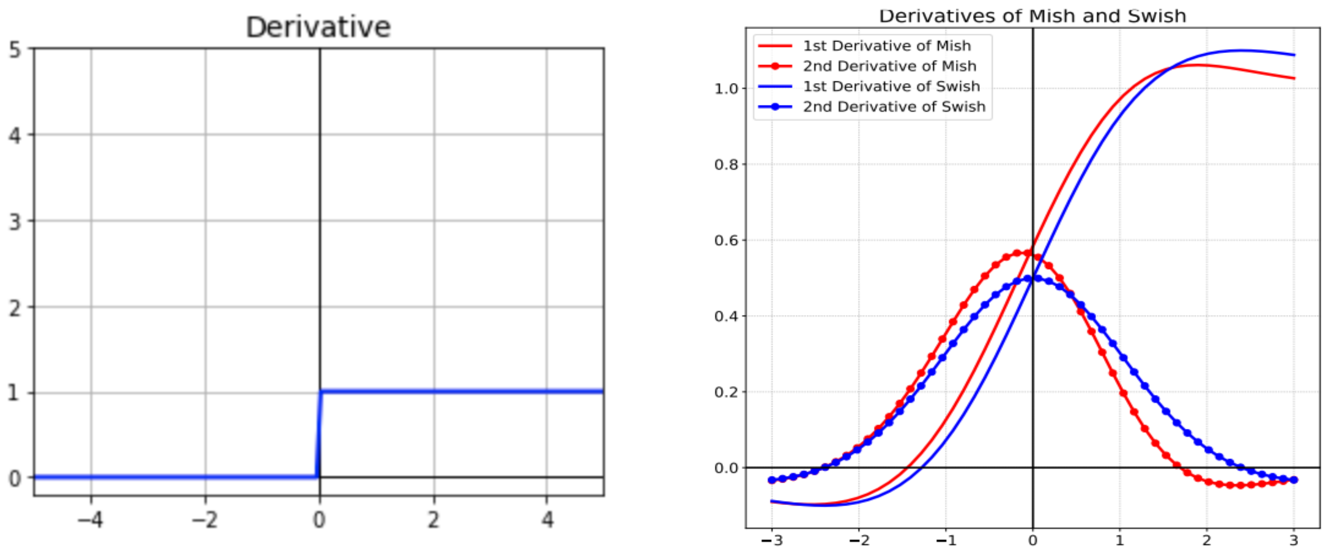


Figure 34: Derivative of ReLU vs Mish and Swish

By having a different derivative function compared to ReLU, and as well as being able to preserve negative weights through the negative concavity of the activation function itself, Mish and Swish is able to achieve non-monotonicity, preservation of small negative weights and an overall smoother profile [32]. ReLU has a gradient continuity of zero, meaning ReLU is not suited to be continuously differentiated, which causes gradient issues. However Mish and Swish do not have this problem due to the nature of their differentiation function.

$$x \tanh(\log(1 + e^{x})) \qquad \frac{x}{1+e^{-x}}$$

Figure 35: Formula for Mish (left) and Swish (right)

The main difference between Mish and Swish is that Mish is overall better and provides more consistent results over Swish, but at the cost of computational resources. Mish and Swish, like ReLU, can be used in the hidden layers for regression and classification problems.

2.9.10 Tanh

Tanh is also known as the hyperbolic tangent activation function.

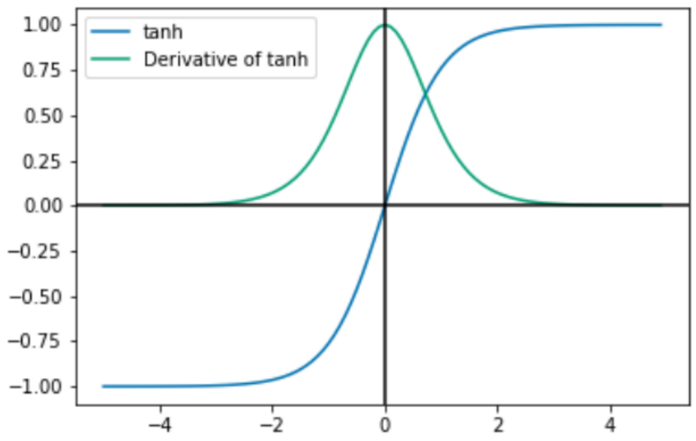


Figure 36: Activation and derivative of tanh

Tanh has a similar structure when compared to a sigmoid activation function. The main difference is that tanh squashes the entire input into a range of -1 to 1 [24]. The main advantage of this is that the derivative function of tanh is steeper overall, which allows tanh to get more values compared to sigmoid. However, because of its bounded input value, the vanishing gradient problem will occur in tanh [24].

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Figure 37: Formula of tanh

Tanh is an activation formula that is generally used in the output layer of the neural network. Tanh is also generally used in classification and logistic regression problems.

2.9.11 Softmax

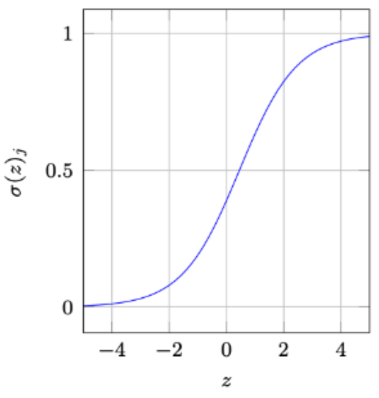


Figure 38: Softmax activation graph

The softmax activation function is a function that converts a vector of real numbers and assigns them into a probability distribution of K possible outcomes [33]. Softmax is normally used in multinomial logistic regression and is normally used as the very last activation function in the output layers. It is done so that the softmax function is able to normalize the overall output of the neural network

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ	= softmax
$\vec{z}$	= input vector
e^{z_i}	= standard exponential function for input vector
K	= number of classes in the multi-class classifier
e^{z_j}	= standard exponential function for output vector
e^{z_j}	= standard exponential function for output vector

Figure 39: Softmax activation formula

Softmax can be considered to be a variant of the sigmoid activation formula. Softmax is normally used in the output layer. Since Softmax is a variant of the Sigmoid activation function, it generally means that Softmax is only used in classification and logistic regression problems.

2.9.12 Softplus

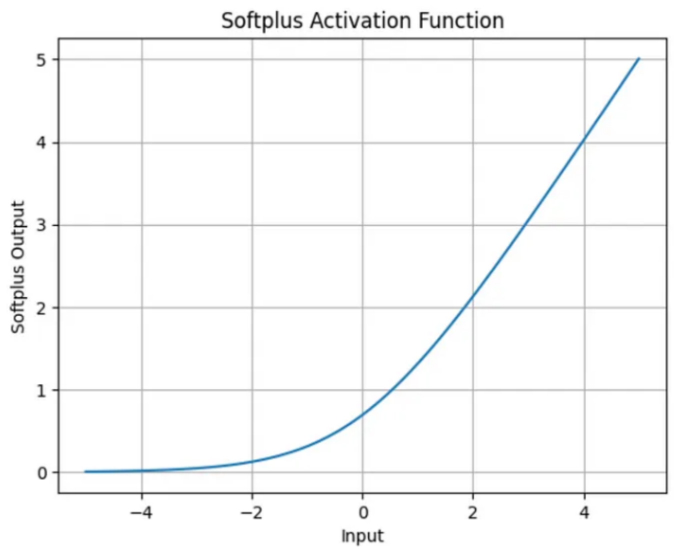


Figure 40: Softplus activation graph

Softplus aims to solve the vanishing gradient problem present in sigmoid by making a few adjustments. The fact that sigmoid is unbounded vertically is one of the causes of it encountering the vanishing gradient problem when going deeper into the hidden layers. To counteract this, softplus preserves certain parts of the gradients, while still remaining unbounded vertically [34]

$$f(x) = \log(1 + \exp(x))$$

Figure 41: Softplus activation formula

Softplus also has advantages over ReLU. ReLU does not have a smooth derivative, which results in an unsmooth profile. Softplus however is smooth on the definition domain itself [34]. It can be argued that this unique advantage stems from the formula of softplus itself, which uses logarithms and exponentials. Softplus is mainly used in the output layer as well. ReLU replaces Softplus when it comes to implementations in the hidden layer. Since Softplus is like ReLU, it can be used for regression

2.9.13 Softsign

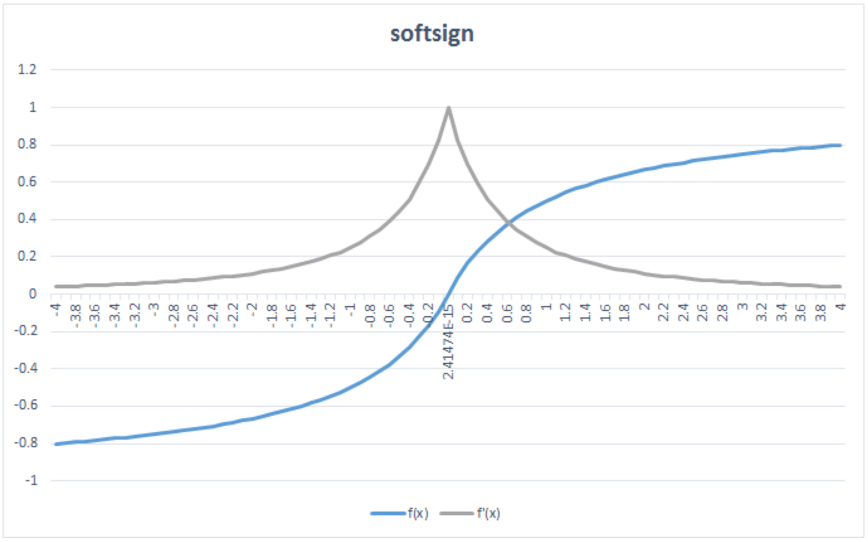


Figure 42: Softsign activation graph+derivative

The softsign activation function is similar in nature to the hyperbolic tangent function tanh. Because softsign's derivative function is sharper and the fact that the area close to x=0 is less, which results in the degree of non-linearization being high [35].

$$y = softsign(x) = f(x) = \frac{x}{|x| + 1}$$

Figure 43: Softsign activation formula

Softsign is considered as a polynomial function due to its nonlinear nature. Softsign is simple in computational calculation, has better fault tolerance and allows for easy convergence [35]. This is because Tanh converges exponentially whereas Softsign converges polynomially.

2.9.14 Activation function analysis

In short, ReLU6 can be seen as the less resource intensive activation method. Elu attempts to solve certain problems within ReLU by introducing exponents and negative weights to reduce bias shifting. SeLU enforces normalization into the neural network model. Mish and Swish aim to solve most problems present in ReLU by having both negative weights and a different differentiation method. All variants of ReLU, as well as Mish and Swish will be used for testing. Sigmoid will not be used. This is because of the characteristic and nature of Sigmoid that makes it unsuitable for regression. Tanh, Softmax and Softsign will not be used as output functions due to them being based off of Sigmoid. Linear and Softplus can be used since Linear is unbounded in both axes and Softplus is similar to ReLU. However, due to time constraints, only Linear will be used in this research.

3.0 Methodology

After careful consideration and time delegation, the project was decided to only implement image compression. The first reason is that the Web Energy Efficient Computing (WEEC) proposed by Uzair et al in 2019 has the most convincing results along with a clear guide on the theory behind it. WEEC uses linear regression to predict the compression scale, and finds out which setting is best. This project however aims to do things in a different way. This project aims to use a neural network to find out the prediction scale instead. Different activation methods will be used in conjunction with the neural networks and the changes in results will be studied. The quality of the image will not be the main focus of this study, but the standard of quality will still be preserved throughout the project. In order to determine a scale for measuring the success of the result, the power consumption will be measured using a built-in power profiler within Firefox. 3.1 Library Choices

3.1 Libraries and Runtime

3.1.1 Node Package Manager

Node Package Manager (NPM) was used because of flexibility with installing libraries. Traditionally, JavaScript does not support require() functions, which is needed for importing libraries. Additionally, a key library named TensorFlowJS is required and NPM is required to perform a proper installation. NPM is the main management system for Node.js, but it does not force users to develop the whole project in Node.js, nor does it enforce a Node.js runtime.

The benefits of NPM are that NPM is a reliable service that is commonly used by developers to build projects. NPM is widely used within the community, which means that the support for NPM is good, and most functioning aspects of NPM will be up to date. If errors ever occur, vast amounts of documentation and community support exist to help developers overcome the obstacles that they are facing. Lastly, NPM compiles all dependencies into a singular package.json file, which can be especially useful for multiple developers working on the same project, or working on multiple projects that require the same dependency. When compiling the application, NPM will be able to automatically detect which node modules have been installed, and which are not just by referencing the package.json file. NPM will then download the modules accordingly.

3.1.2 Webpack

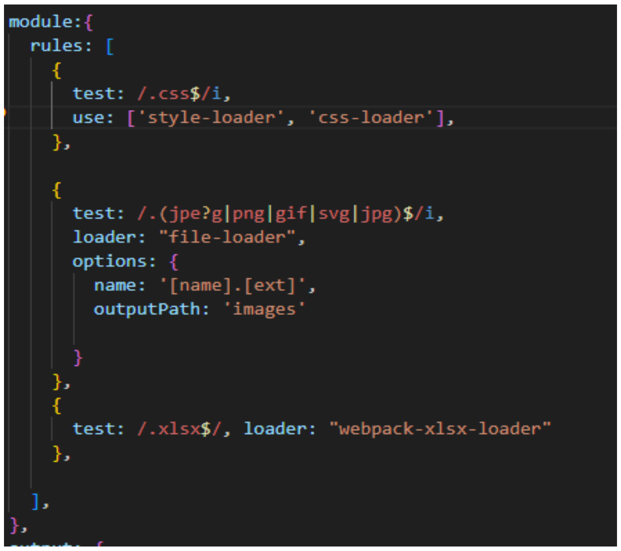
Webpack is a module bundler for JavaScript projects. When bundling, Webpack will build a dependency graph from entry points within the code, it then combines the modules within the project into bundles which act as static assets to serve content [36]

There are several aspects of webpack that require further explanation: Entry, Output, Loaders, Plugins and Mode.

1. Entry: Entry point dictates where webpack should start building its dependency graph. Along the way, webpack will continue figuring out modules and libraries that the entry point depends on. The default entry point for most webpack projects point to './src/index.js'. This project will utilize the default entry point.

2. Output: The output property dictates the location as to where webpack should place the bundles into. By default, the bundles are placed into './dist'.

3. Loaders: By default, webpack only understands JavaScript and JSON files. This means that webpack is unable to process any other files like xlsx, css and all image files.

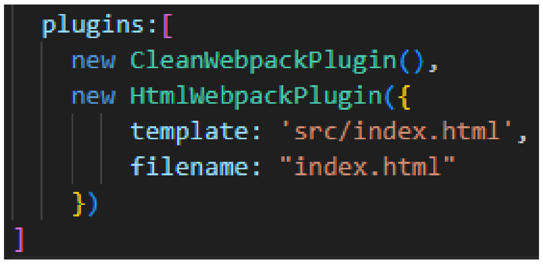


```
module:{
  rules: [
    {
      test: /.css$/i,
      use: ['style-loader', 'css-loader'],
    },

    {
      test: /.(jpe?g|png|gif|svg|jpg)$/i,
      loader: "file-loader",
      options: {
        name: '[name].[ext]',
        outputPath: 'images'

      }
    },
    {
      test: /.xlsx$/, loader: "webpack-xlsx-loader"
    },

  ],
},
```

Figure 44: Image of loaders in webpack.config.js

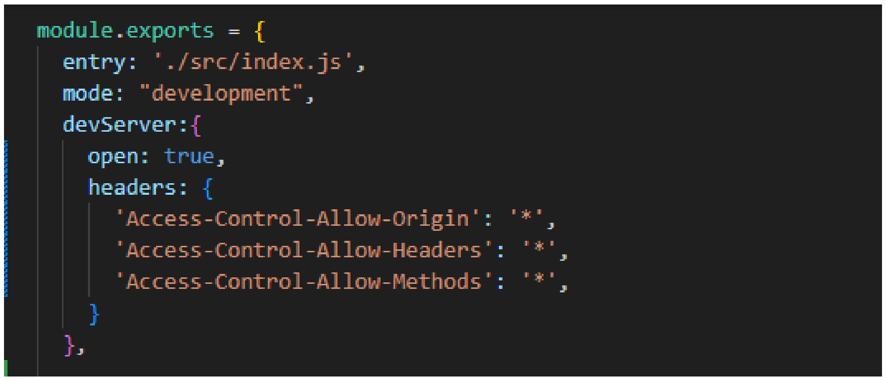
Within the loaders, there are two main properties: test and use. The test property will check for the file's file type. Using regular expressions (regex), the config can specify which file type will utilize which loader. In the first example, files with the css extension type will use the style-loader and css-loader modules, both of which were installed using Node Package Manager. In the second example, the name and outputPath is specified. This part will be relevant when the bundle is generated into the specified folder

4. Plugins: Plugins in webpack are used to transform certain modules to perform tasks like bundle optimization, asset management and injection of environment variables. The main plugin used in webpack is the HtmlWebpackPlugin. HtmlWebpackPlugin simplifies the creation of HTML files. This HTML file will include all the bundled modules.



```
 plugins:[
   new CleanWebpackPlugin(),
   new HtmlWebpackPlugin({
       template: 'src/index.html',
       filename: "index.html"
   })
]
```

Figure 45: Image of plugins



```
module.exports = {
  entry: './src/index.js',
  mode: "development",
  devServer:{
    open: true,
    headers: {
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Headers': '*',
      'Access-Control-Allow-Methods': '*',
    }
  },
```

Figure 46: Headers option

Certain settings are crucial within the webpack module. The first important part is the headers. Access-Control-Allow-Origin, Access-Control-Allow-Headers and Access-Control-Allow-Methods are 3 headers that are important in bypassing CORS issues for browsers. The CORS error is an error that happens when a webpage makes a request to a different domain that did not serve the server due to the origin of the header in the request being blocked by the server [37]. To bypass this, the heading of the request must be set to asterisks. This allows the webpage being used to be able to bypass CORS limitations on servers. The modules are in place to load file types like css, images and xlsx files. JavaScript at its base is not able to do these functions and require some form of library or node module to do so.

There are multiple benefits of using webpack in the language bundling.

1. Speeding up development: Webpack’s development server offers features like not requiring a full reboot of the web server to reflect every change in the code. Limitations to this feature do exist, for example when updating configuration files relating to Webpack itself. However, the benefits outweigh the costs. Webpack will automatically detect when the code has been changed, and upon change automatically refresh the page running on the development server to reflect the changes.
2. Removing the problem of overwriting global variables: Webpack, as a module bundler for JavaScript applications, facilitates the use of modules to structure code. In the context of JavaScript modules (specifically ES6 modules), each module has its own scope. This means that variables declared within a module are local to that module, which reduces the risk of conflicts happening between global variables.
3. Provides splitting of code: Because it utilizes a module system, Webpack treats files as modules. Treating files as modules enables the utilization of one file's functionalities in another. This allows us to access the same advantages across different files, facilitating the division of our code into distinct modules.
4. Provides Minification: Minification involves reducing the code size without altering its functionality. This process eliminates whitespace, line breaks, and unnecessary code, as well as shortens long variable names. By doing so, file size is reduced, leading to a more concise and streamlined code.
5. Supports feature flagging: Feature flagging is a software development technique that involves directing code to various environments during feature testing. Webpack serves a dual purpose by aiding in both development and testing processes.

With these in mind, webpack is chosen for bundling as well as being the runtime for the execution of the code for this project.

3.1.3 TensorflowJS (TFJS)

TensorflowJS, which will be abbreviated as TFJS from here on out, is the neural network library of choice for this project. There exists a myriad of other neural network libraries out there, like Ml5.js, Brain.js and Keras.js I. However, TFJS is widely regarded as the bedrock of the neural network community. Ml5.js works in conjunction with TFJS, it acts like a high level interface that sits on top of TFJS, and provides additional features while still using TFJS as a baseline [38]. The same can be said about Keras.js. Much like Ml5.js, Keras.js is also a high-level API running on top of TFJS. The only remaining competitor to TFJS is Brain.js. The following table will list down detailed comparisons between TFJS and Brain.js, and justify the use of TFJS.

Aspect	TFJS	Brain.js
Ecosystem & Community	Has a large community alongside with the original Tensorflow in Python	Smaller community, lesser support
Flexibility & Versatility	Comprehensive, supports various machine learning models	Lightweight, focused on neural networks, lacking in certain features
Ease of Use	Extensive documentation, may have a steeper learning curve	Simplicity in usage
Model Types	Supports a wide range of machine learning models beyond neural networks	Supports various neural network types
Browser Integration	Designed for running in browsers and on Node.js	Well-suited for simple browser applications
Community Support	Extensive community support, popular with the ML community	Active, but not as extensive as TensorFlow

Table 2: TFJS vs Brain.js

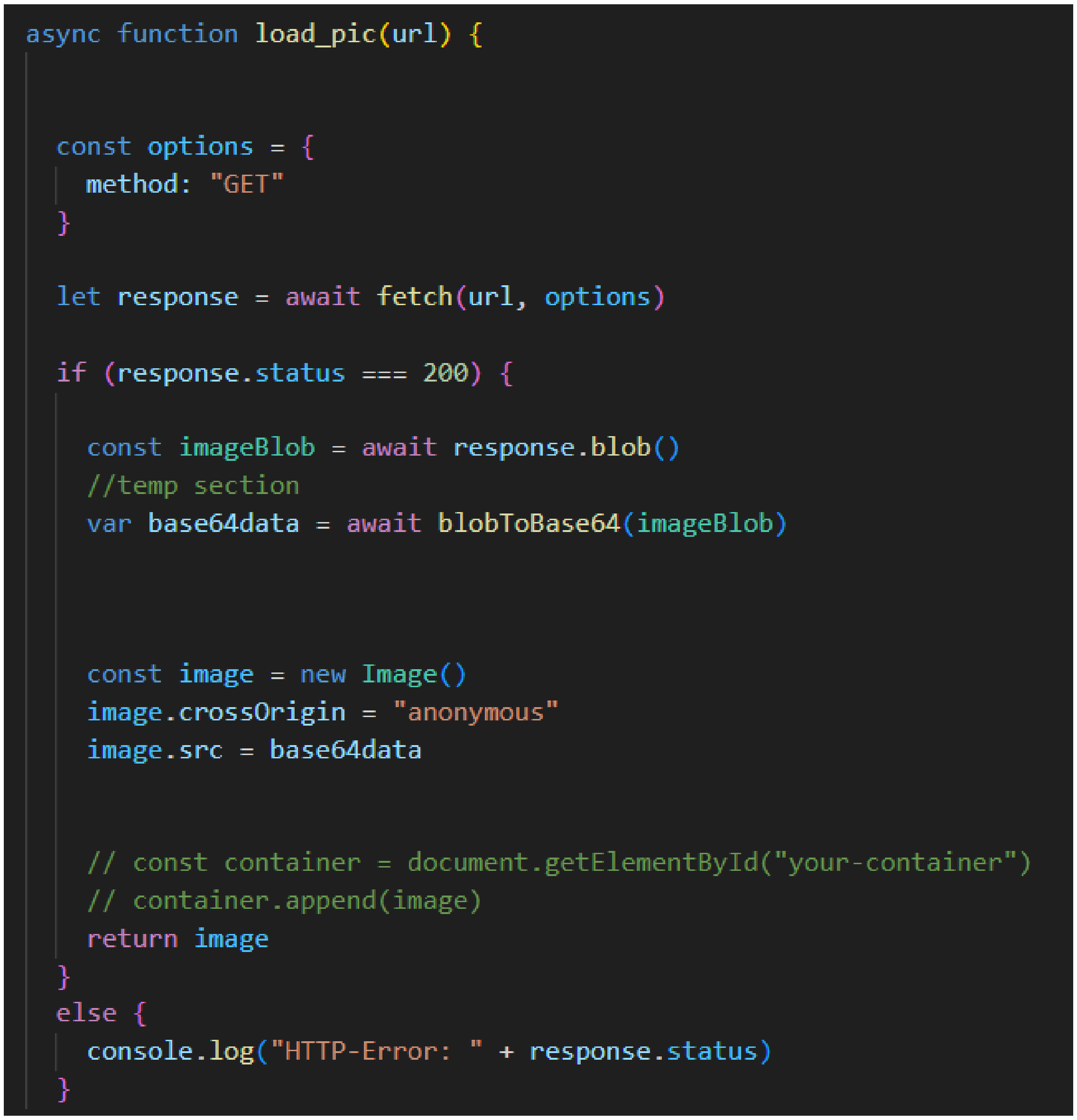
From the information shown, it can be seen at a glance that TFJS is the superior version. Albeit being harder to use, TFJS offers more features, and is widely supported and recognized by the community. This fact will be crucial during development since more support and recognition generally leads to more explored solutions as well as an abundance of comprehensive guides that can be referred to when problems arise.

3.2 Approach

The nature of this project is one with the goal of answering a question. In this case, two main questions are asked. The first question aims to find out the validity of this approach. Can the proposed method effectively reduce power consumption? And the second questions the effect of different activation methods on the results. Therefore, since the project is that of a research nature, the website will not be designed with the intent of being a fully functional application to be used by users. The code and the project overall will fully focus on solving the question at hand using whatever means possible. Supervised learning will be used because there is both training and testing data. Two model types will be used: Sequential and Functional model. These models will use Multilayer Perceptron for the input section containing hidden layers, but the overall model's structure will be a Feed Forward Neural Network.

3.3 Code

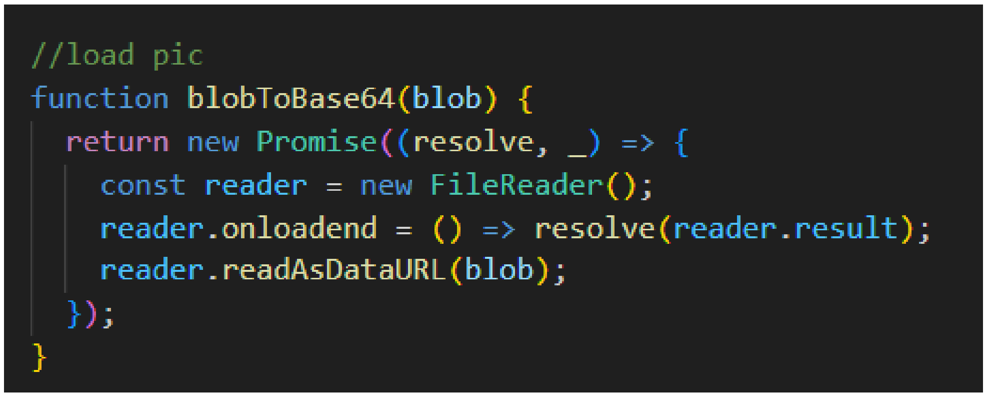
The test simulation was coded in such a way that it would best resemble a real life scenario. In a real life scenario, the image is first fetched from a server, and then rendered onto the webpage.



```
async function load_pic(url) {


  const options = {
    method: "GET"
  }

  let response = await fetch(url, options)

  if (response.status === 200) {

    const imageBlob = await response.blob()
    //temp section
    var base64data = await blobToBase64(imageBlob)



    const image = new Image()
    image.crossOrigin = "anonymous"
    image.src = base64data


    // const container = document.getElementById("your-container")
    // container.append(image)
    return image
  }
  else {
    console.log("HTTP-Error: " + response.status)
  }
```

Figure 47: Code for fetching images from server

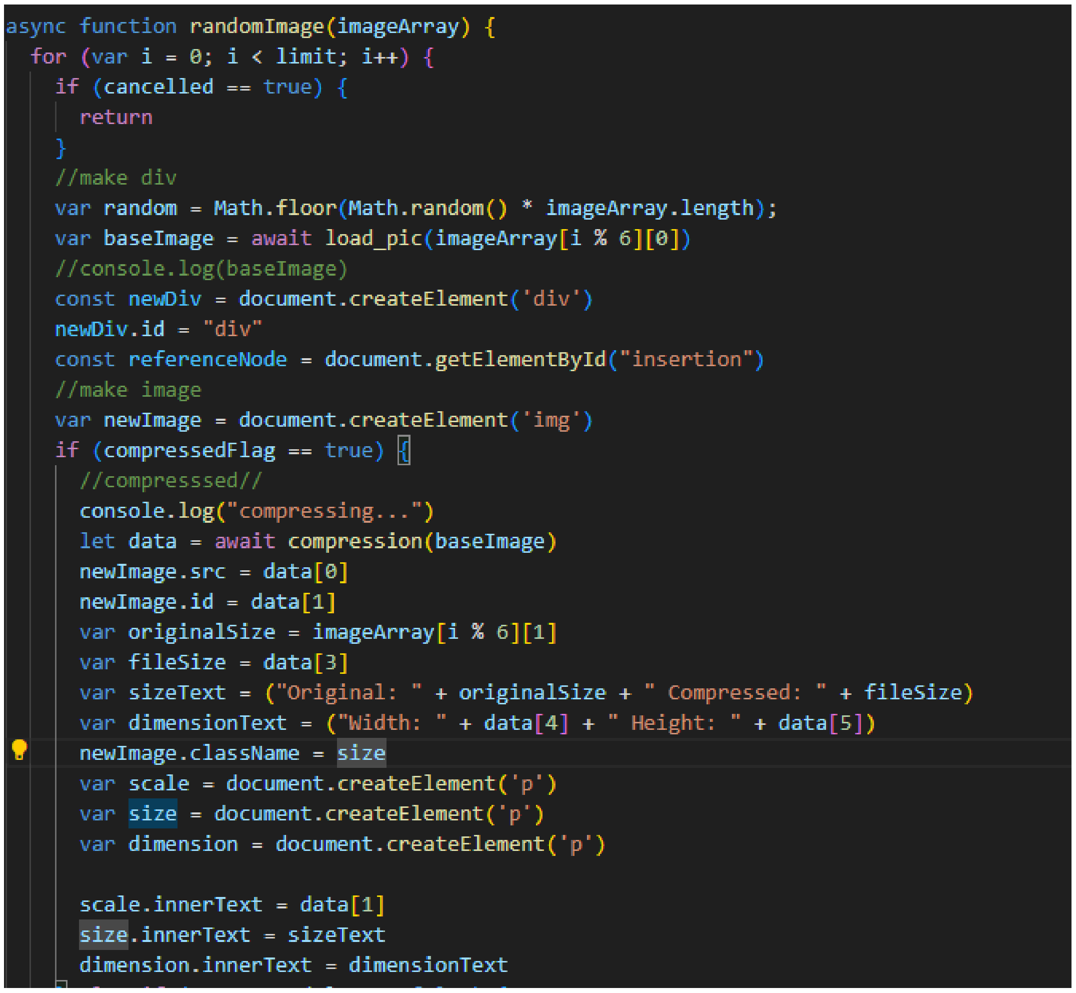
This function will take in a url parameter, and then load the image according to that url. The image loaded is then converted into base64 for easy manipulation.



```
//load pic
function blobToBase64(blob) {
  return new Promise((resolve, _) => {
    const reader = new FileReader();
    reader.onloadend = () => resolve(reader.result);
    reader.readAsDataURL(blob);
  });
}
```

Figure 48: Code for converting image from blob to base64

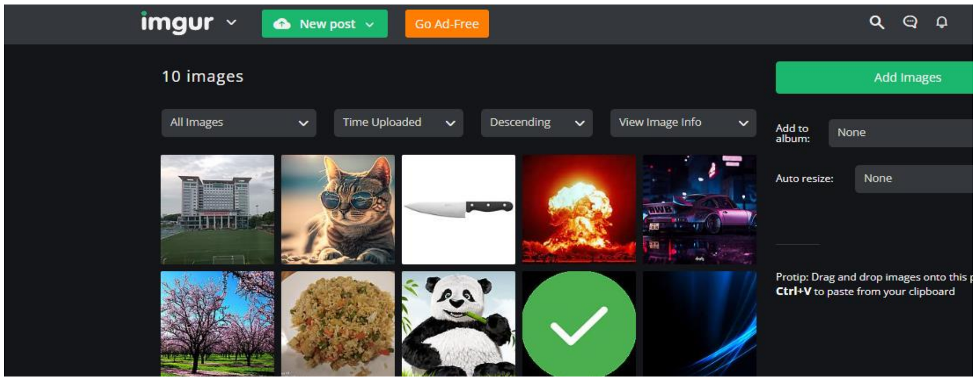
The image needs to be converted into base64. The reason for this is because a blob object is hard to make modifications to. Compression functions handle base64 formatting way better than a blob object.



```
async function randomImage(imageArray) {
  for (var i = 0; i < limit; i++) {
    if (cancelled == true) {
      return
    }
    //make div
    var random = Math.floor(Math.random() * imageArray.length);
    var baseImage = await load_pic(imageArray[i % 6][0])
    //console.log(baseImage)
    const newDiv = document.createElement('div')
    newDiv.id = "div"
    const referenceNode = document.getElementById("insertion")
    //make image
    var newImage = document.createElement('img')
    if (compressedFlag == true) {
      //compresssed//
      console.log("compressing...")
      let data = await compression(baseImage)
      newImage.src = data[0]
      newImage.id = data[1]
      var originalSize = imageArray[i % 6][1]
      var fileSize = data[3]
      var sizeText = ("Original: " + originalSize + " Compressed: " + fileSize)
      var dimensionText = ("Width: " + data[4] + " Height: " + data[5])
      newImage.className = size
      var scale = document.createElement('p')
      var size = document.createElement('p')
      var dimension = document.createElement('p')

      scale.innerText = data[1]
      size.innerText = sizeText
      dimension.innerText = dimensionText
```

Figure 49: Algorithm that initializes compression

After that, the randomImage function is called. This function will be the function that draws x amount of images onto the website. If compressedFlag is true, it means that we will be using the compressed version of the image to test for the effectiveness of the model. The randomImage function will first create a img DOM element, and then compress the image parsed using the compression function. After getting the compressed image, the randomImage function will then set the previously created img DOM element's source as the compressed image .The randomImage function will then create DOM elements that contain information about the compressed image like the scale used as well as the size of the image and the dimension of the image rendered.



Add to album: None

Auto resize: None

Protip: Drag and drop images onto this p
Ctrl+V to paste from your clipboard

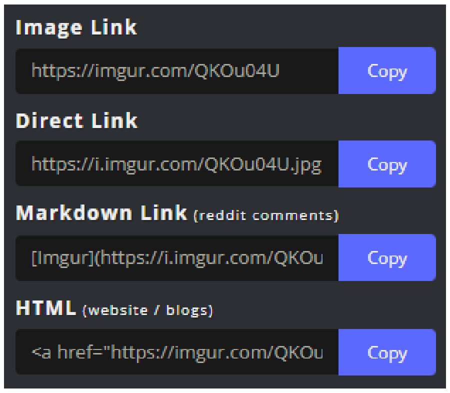


Image Link
https://imgur.com/QKOu04U Copy

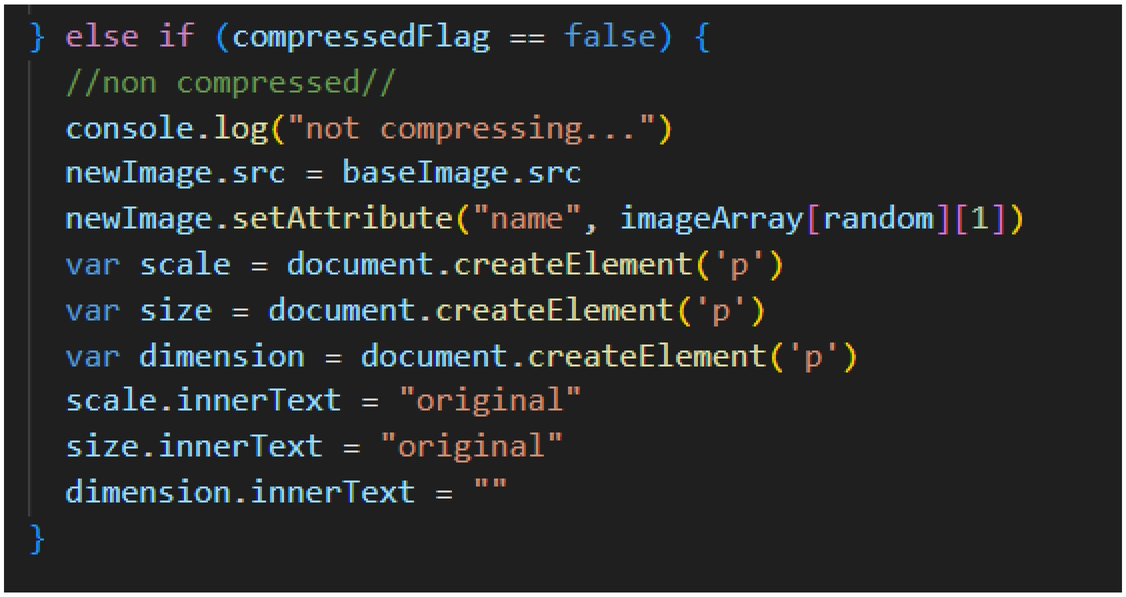
Direct Link
https://i.imgur.com/QKOu04U.jpg Copy

Markdown Link (reddit comments)
[Imgur](https://i.imgur.com/QKOu Copy

HTML (website / blogs)
<a href="https://imgur.com/QKOu Copy

Figure 50: Images of Imgur

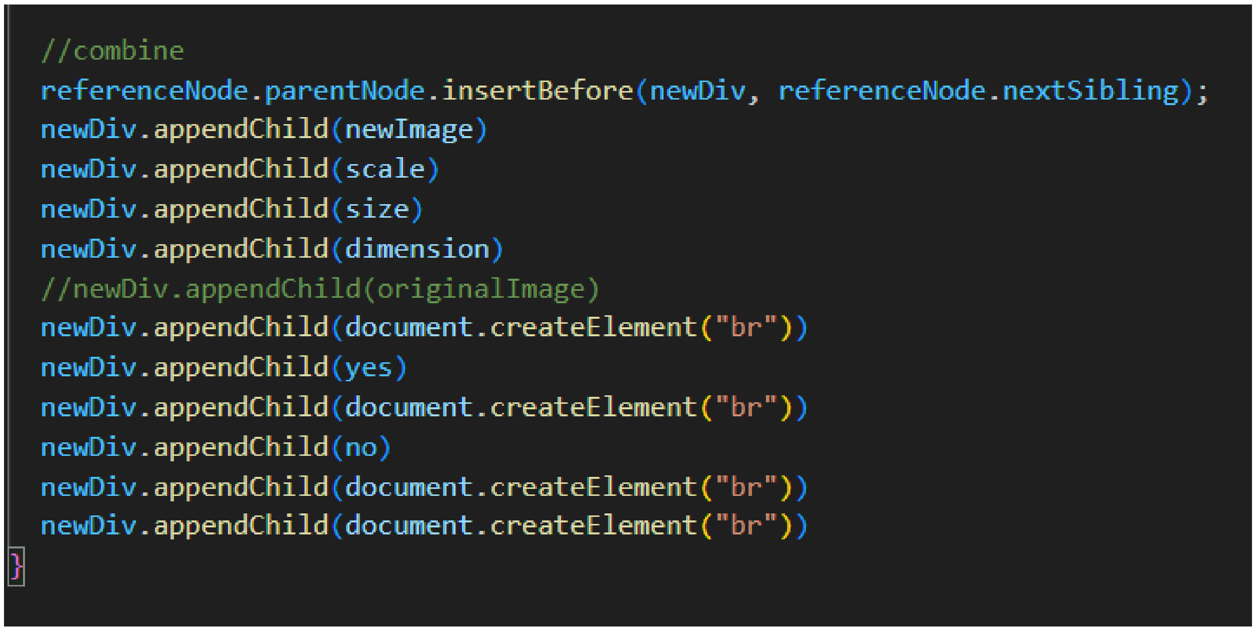
All of the images are uploaded into an image hosting website called imgur. Imgur was selected as the image website because of its simplicity. Imgur is also ideal for this scenario because it has built in support for JavaScript to directly access links. This reduces problems like compatibility issues or security issues to some extent.



```
} else if (compressedFlag == false) {
  //non compressed//
  console.log("not compressing...")
  newImage.src = baseImage.src
  newImage.setAttribute("name", imageArray[random][1])
  var scale = document.createElement('p')
  var size = document.createElement('p')
  var dimension = document.createElement('p')
  scale.innerText = "original"
  size.innerText = "original"
  dimension.innerText = ""
}
```

Figure 51: If statements for compression check

If the non compressed version is the one set for the current runtime, then the randomImage function will instead execute this code block. This code block basically assigns the parsed image from the imageArray and draws it onto the previously created img DOM.

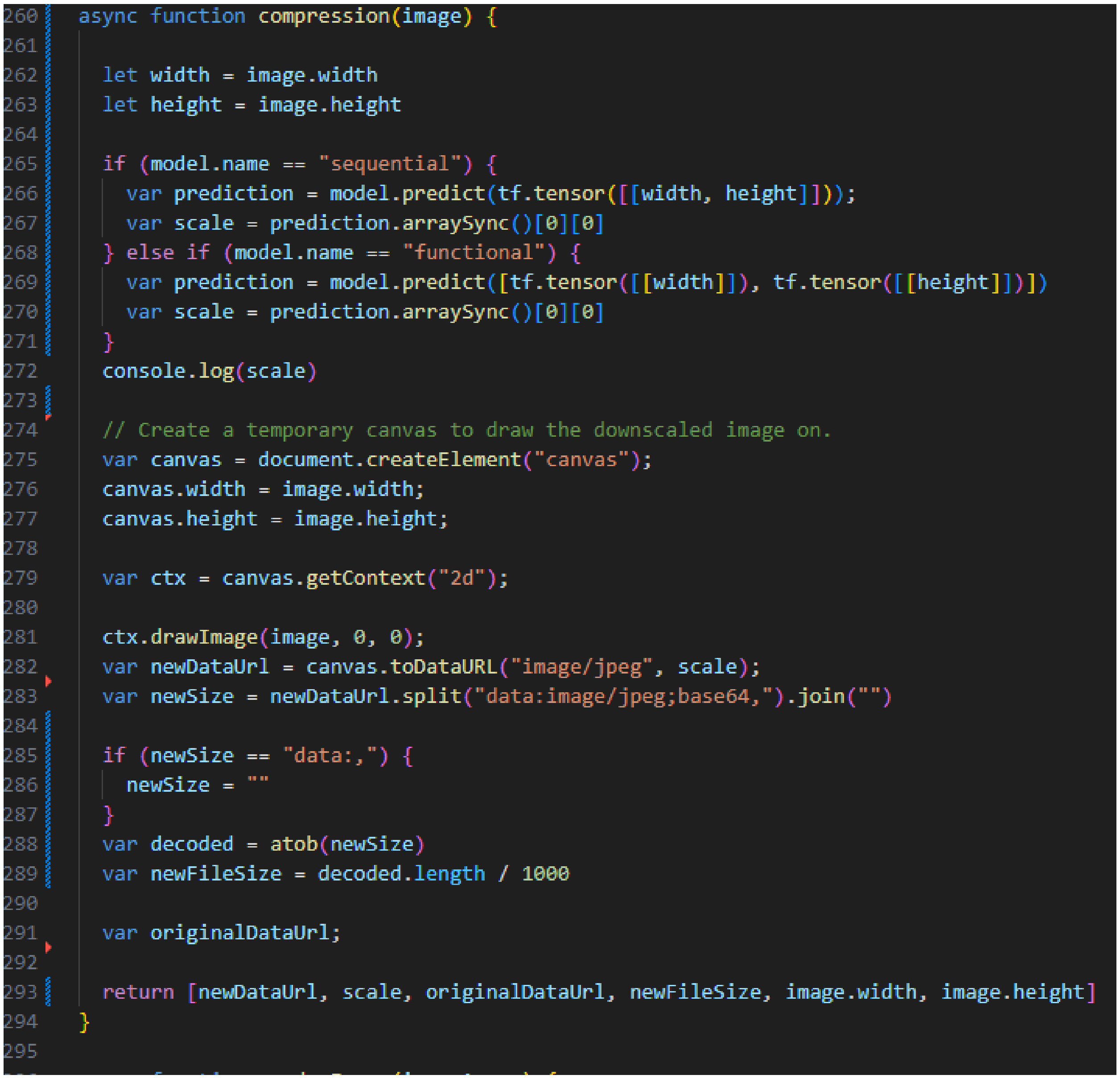


```
  //combine
  referenceNode.parentNode.insertBefore(newDiv, referenceNode.nextSibling);
  newDiv.appendChild(newImage)
  newDiv.appendChild(scale)
  newDiv.appendChild(size)
  newDiv.appendChild(dimension)
  //newDiv.appendChild(originalImage)
  newDiv.appendChild(document.createElement("br"))
  newDiv.appendChild(yes)
  newDiv.appendChild(document.createElement("br"))
  newDiv.appendChild(no)
  newDiv.appendChild(document.createElement("br"))
  newDiv.appendChild(document.createElement("br"))
}
```

Figure 52: Code for adding necessary DOM elements

This section of the code is the section that is responsible for appending to the DOM. A for loop runs this section according to the amount of images, then the DOM nodes are appended accordingly.

This code section is the part that appends all the DOM elements created from above. Inside the base HTML file, there exists a div with the id set as newDiv. This div serves as a reference point for us to insert all the created DOM elements into the website.

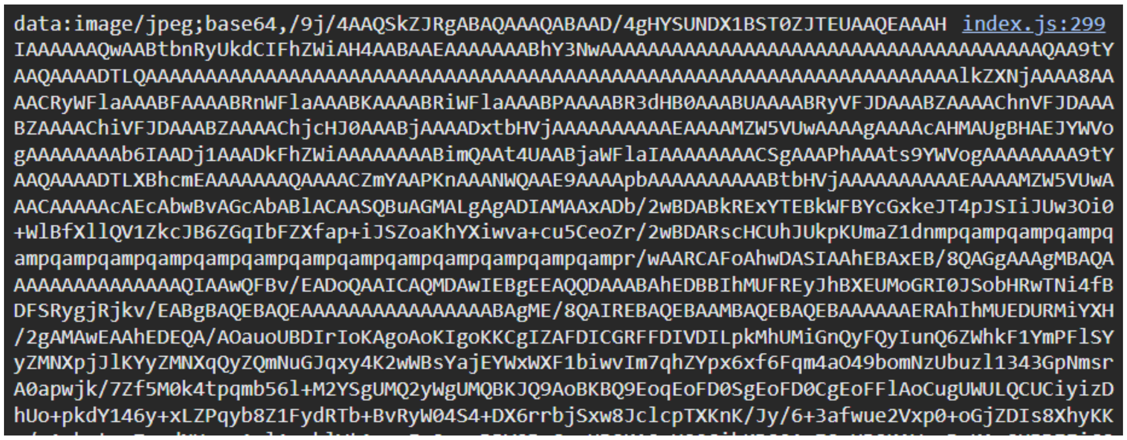


```
async function compression(image) {

  let width = image.width
  let height = image.height

  if (model.name == "sequential") {
    var prediction = model.predict(tf.tensor([[width, height]]));
    var scale = prediction.arraySync()[0][0]
  } else if (model.name == "functional") {
    var prediction = model.predict([tf.tensor([[width]]), tf.tensor([[height]])])
    var scale = prediction.arraySync()[0][0]
  }
  console.log(scale)

  // Create a temporary canvas to draw the downscaled image on.
  var canvas = document.createElement("canvas");
  canvas.width = image.width;
  canvas.height = image.height;

  var ctx = canvas.getContext("2d");

  ctx.drawImage(image, 0, 0);
  var newDataUrl = canvas.toDataURL("image/jpeg", scale);
  var newSize = newDataUrl.split("data:image/jpeg;base64,").join("")

  if (newSize == "data:,") {
    newSize = ""
  }
  var decoded = atob(newSize)
  var newFileSize = decoded.length / 1000

  var originalDataUrl;

  return [newDataUrl, scale, originalDataUrl, newFileSize, image.width, image.height]
}

```

Figure 53: Code where image compression happens

This function is where the compression of the image happens. The function will first check what the model type is, and then make a prediction accordingly. The prediction will be the compression scale used. Before compression happens, a canvas object that corresponds to the size of the image. The image will then be copied onto the created canvas image, while specifying the compression scale. In this case, the lower the scale, the higher the compression.

The term compression can be defined as so: a process that makes image files smaller. The canvas.toDataURL function is made to save images by sacrificing detail. The detail sacrificed is specified by the scale parsed into the function. However, this can be seen as a deviation from the standard compression understanding. Lastly, all images will be converted into JPEG format. In a research done by Uzair et al. in 2019, JPG images were the standard for compression when trying to reduce power consumption caused by loading images. In this case, JPG is not available to the library, but JPEG exists, which is literally the same file type, but renamed due to the updates that the Windows operating system received [39]. From line 283 onwards, the code is used to calculate the file size of the image for comparison purposes.



```
data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/4gHYSUNDX1BST0ZJTEUAAQEAAAH  index.js:299
IAAAAAAQwAABtbnRyUkdCIFhZWiAH4AABAAEAAAAAAABhY3NwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAA9tY
AAQAAAADTLQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAlkZXNjAAAA8AA
AACRyWFlaAAABFAAAABRnWFlaAAABKAAAABRiWFlaAAABPAAAABR3dHB0AAABUAAAABRyVFJDAAABZAAAAChnVFJDAAA
BZAAAAChiVFJDAAABZAAAAChjcHJ0AAABjAAAADxtbHVjAAAAAAAAAAEAAAAMZW5VUwAAAAgAAAAcAHMAUgBHAEJYWVo
gAAAAAAAAb6IAADj1AAADkFhZWiAAAAAAAABimQAAt4UAABjaWFlaIAAAAAAAACSgAAAPhAAAts9YWVogAAAAAAAA9tY
AAQAAAADTLXBhcmEAAAAAAAQAAAACZmYAAPKnAAANWQAAE9AAAApbAAAAAAAAAABtbHVjAAAAAAAAAAEAAAAMZW5VUwA
AACAAAAAcAEcAbwBvAGcAbABlACAASQBuAGMALgAgADIAMAAxADb/2wBDABkRExYTEBkWFBYcGxkeJT4pJSIiJUw3Oi0
+WlBfXllQV1ZkcJB6ZGqIbFZXfap+iJSZoaKhYXiwva+cu5CeoZr/2wBDARscHCUhJUkpKUmaZ1dnmpqampqampqampq
ampqampqampqampqampqampqampqampqampqampqampqampqampr/wAARCAFoAhwDASIAAhEBAxEB/8QAGgAAAgMBAQA
AAAAAAAAAAAAAAQIAAwQFBv/EADoQAAICAQMDAwIEBgEEAQQDAAABAhEDBBIhMUFREyJhBXEUMoGRI0JSobHRwTNi4fB
DFSRygjRjkv/EABgBAQEBAQEAAAAAAAAAAAAAAAABAgME/8QAIREBAQEBAAMBAQEBAQEBAAAAAAERAhIhMUEDURMiYXH
/2gAMAwEAAhEDEQA/AOauoUBDIrIoKAgoAoKIgoKKCgIZAFDICGRFFDIVDILpkMhUMiGnQyFQyIunQ6ZWhkF1YmPFlSY
yZMNXpjJlKYyZMNXqQyZQmNuGJqxy4K2wWBsYajEYWxWXF1biwvIm7qhZYPx6xf6Fqm4aO49bomNzUbuzl1343GpNmsr
A0apwjk/7Zf5M0k4tpqmb561YKUM2YSgUMQ2yWgUMQBKJQ9Eo0l4egUPRKILRaGoNFFVBoaiUQVUShqJRRVRKHqwxhKK
```

Figure 54: base64 information of an image

The images are saved in base64 code format. In order to calculate it, we must first split the file type section of the code away from the base64 code, which is what line 283 does. The if statement in line 285 is added in the event that an image fails to load. When an image fails to load, it will be saved as data:, by default. After splitting off the data type from the base64 code, the built in atob function in js is called to calculate the base64 image file size. Lastly, the output is divided by 1000 to convert it into kilobytes.

3.4 Gathering datasets

Before training the model, data had to first be collected and a dataset had to be established. To do this, a code was written that generated 102 compressed images according to a random scale. A human then manually judged and decided which images looked acceptable, and which images got over-compressed to the point where there was severe quality loss. These images are then rejected and will not be added into the dataset. After all 102 images have gone through the acceptance test, the code will automatically generate a dataset containing the image's width, height, their original file size, their compressed file size and the scale of compression. Data that had a bigger file size after compression are removed. Although it is stated that 102 compressed images will be generated, this does not mean 102 unique images will be used. 5 images will be used, and the code will generate 17 sets of 6 images, all with different compression scales

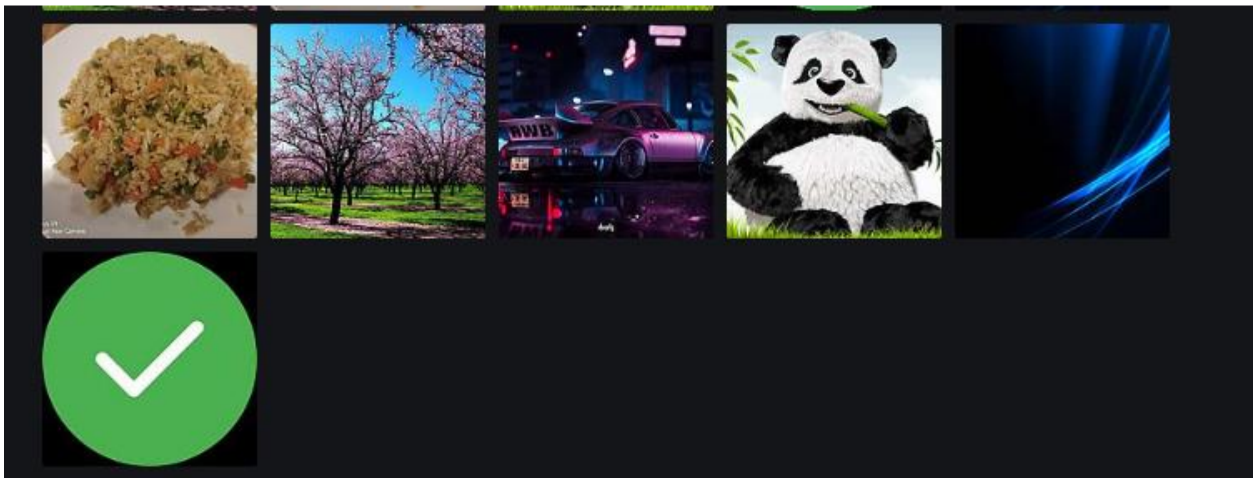


Figure 55: Training images

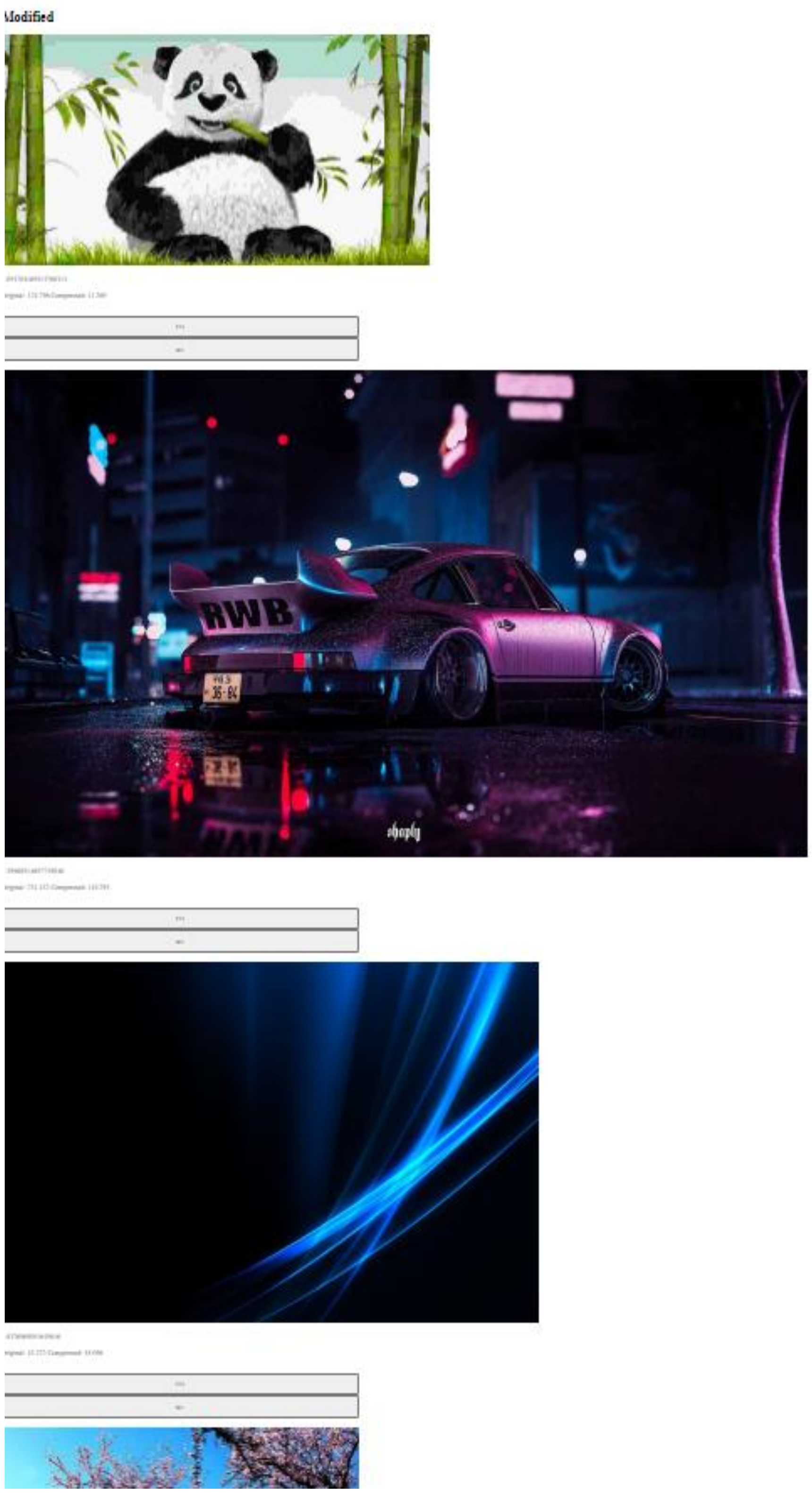
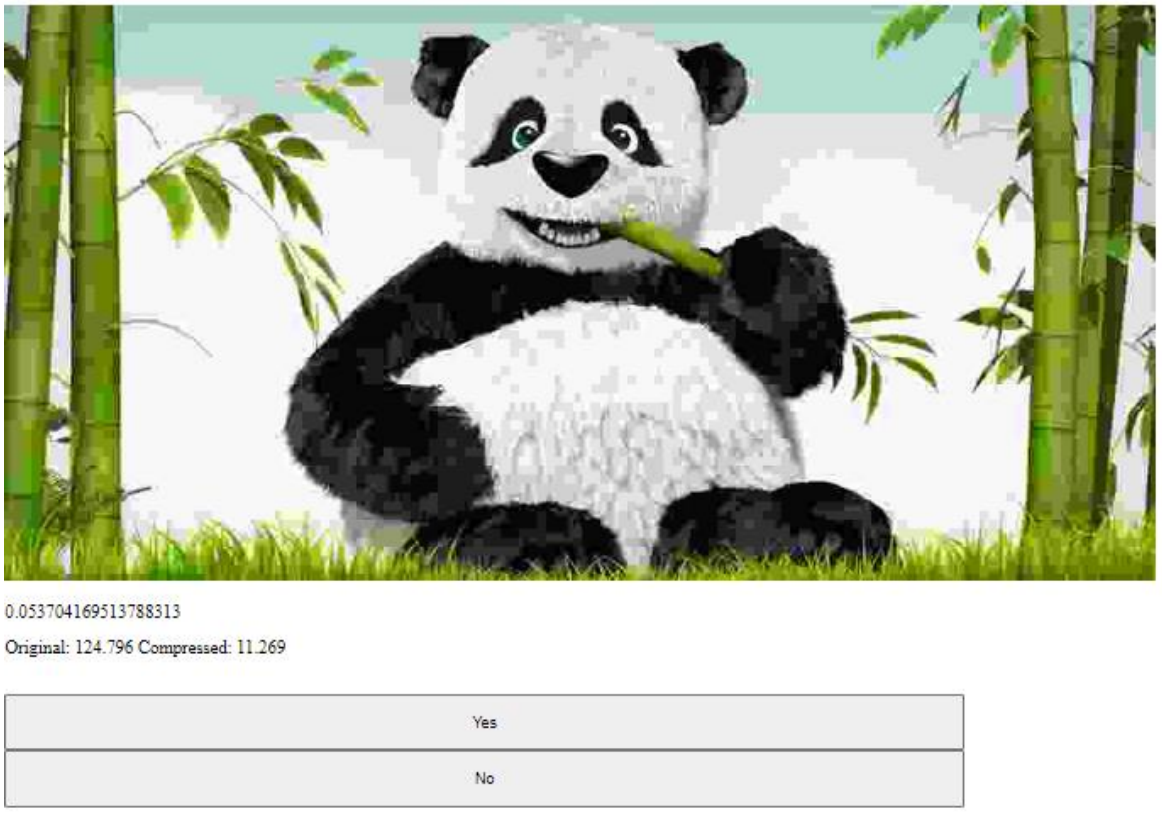
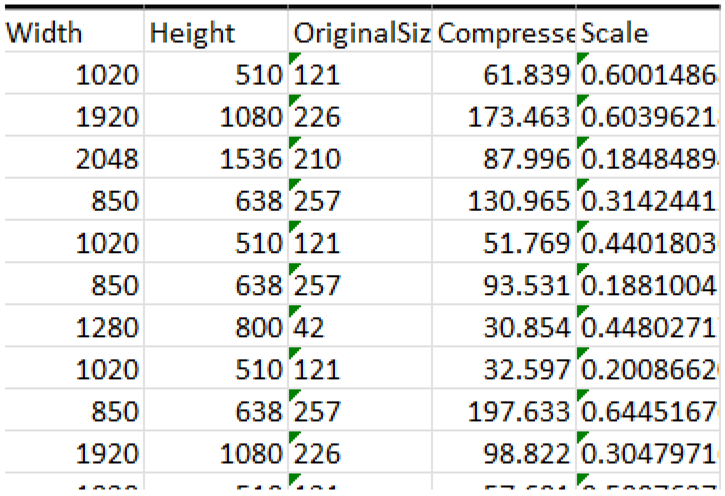


Figure 56: General Overview of the training page


0.053704169513788313

Original: 124.796 Compressed: 11.269

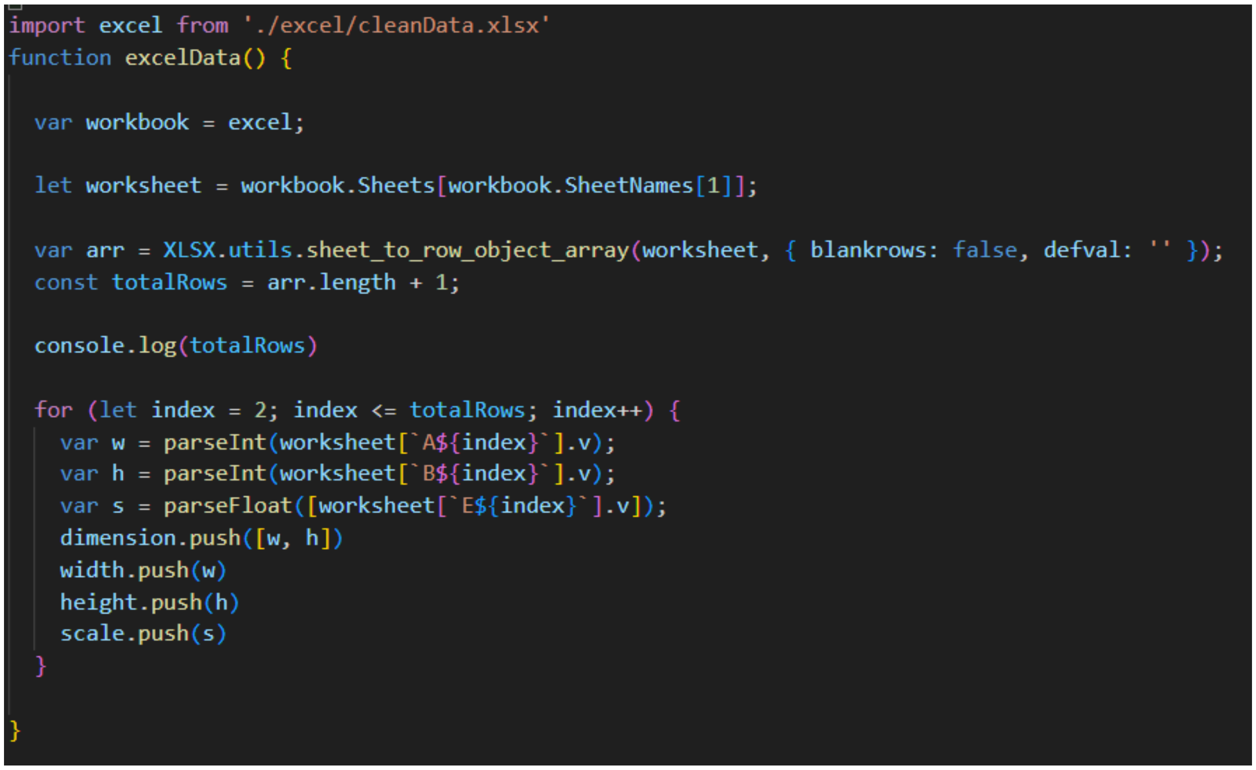
Figure 57: Closeup of one of the training images



Width	Height	OriginalSiz	Compresse	Scale
1020	510	121	61.839	0.6001486
1920	1080	226	173.463	0.6039621
2048	1536	210	87.996	0.1848489
850	638	257	130.965	0.3142441
1020	510	121	51.769	0.4401803
850	638	257	93.531	0.1881004
1280	800	42	30.854	0.4480271
1020	510	121	32.597	0.2008662
850	638	257	197.633	0.6445167
1920	1080	226	98.822	0.3047971

Figure 58: General overview of the dataset

Multiple people were asked to evaluate the images. For each person, 100 images with random compression scales were given to them. The participants were then asked to press either “Yes” if the image looked passable, or press “No” if the image had severe quality loss. Over the span of 7 people, 417 data that passed the criteria was collected. This meant that 417/700 of the images randomly compressed passed the test under the evaluation of a normal person's naked eye.

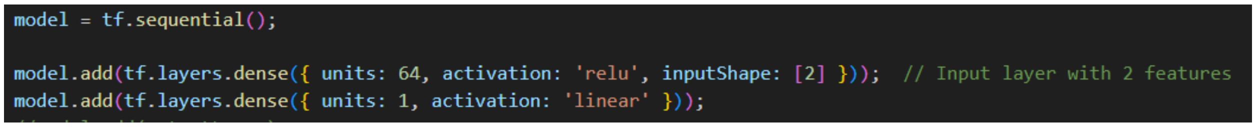


```
import excel from './excel/cleanData.xlsx'
function excelData() {

  var workbook = excel;

  let worksheet = workbook.Sheets[workbook.SheetNames[1]];

  var arr = XLSX.utils.sheet_to_row_object_array(worksheet, { blankrows: false, defval: '' });
  const totalRows = arr.length + 1;

  console.log(totalRows)

  for (let index = 2; index <= totalRows; index++) {
    var w = parseInt(worksheet[`A${index}`].v);
    var h = parseInt(worksheet[`B${index}`].v);
    var s = parseFloat([worksheet[`E${index}`].v]);
    dimension.push([w, h])
    width.push(w)
    height.push(h)
    scale.push(s)
  }

}
```

Figure 59: Retrieving data from excel

The dataset gathered is then saved under a file called cleanData.xlsx, which is imported into the javascript file. The excelData function then reads through the entire worksheet containing the dataset gathered, and saves it into 3 arrays: width, height and scale. At the same time, a fourth array called dimension is also created for testing purposes, but will not be used in the testing data.

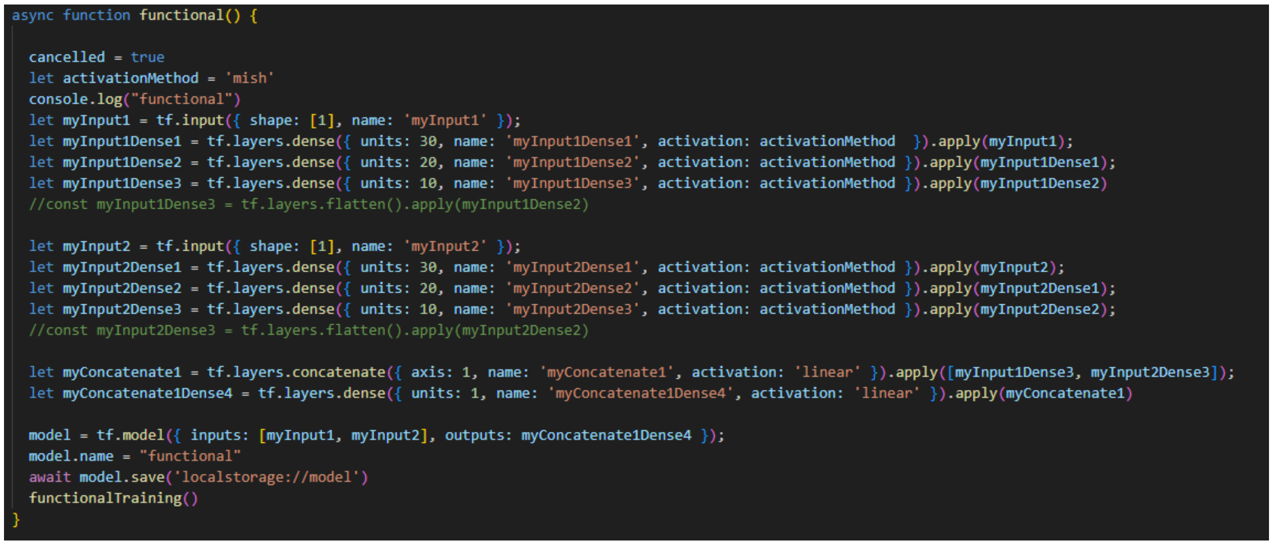
3.5 Neural Network Model



```
model = tf.sequential();

model.add(tf.layers.dense({ units: 64, activation: 'relu', inputShape: [2] }));  // Input layer with 2 features
model.add(tf.layers.dense({ units: 1, activation: 'linear' }));
```

Figure 60: Sequential model

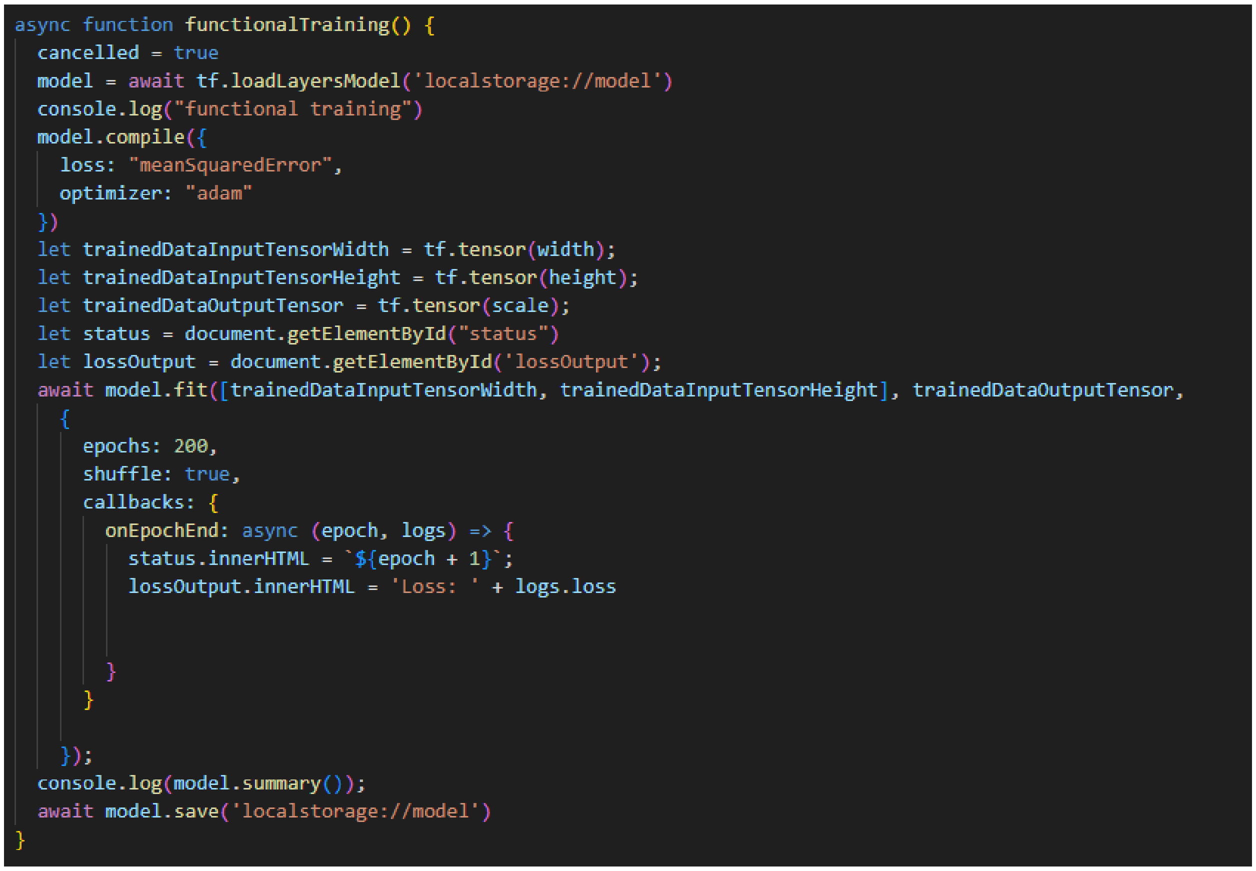
The first neural network model type is the sequential model. As previously mentioned, sequential models are neural network models that connect layers linearly. However, it was later discovered that the sequential model is not suitable for the nature of this project. In section 3.6, hard evidence will be provided to dispute the usage of sequential models.



```
async function functional() {

  cancelled = true
  let activationMethod = 'mish'
  console.log("functional")
  let myInput1 = tf.input({ shape: [1], name: 'myInput1' });
  let myInput1Dense1 = tf.layers.dense({ units: 30, name: 'myInput1Dense1', activation: activationMethod  }).apply(myInput1);
  let myInput1Dense2 = tf.layers.dense({ units: 20, name: 'myInput1Dense2', activation: activationMethod }).apply(myInput1Dense1);
  let myInput1Dense3 = tf.layers.dense({ units: 10, name: 'myInput1Dense3', activation: activationMethod }).apply(myInput1Dense2)
  //const myInput1Dense3 = tf.layers.flatten().apply(myInput1Dense2)

  let myInput2 = tf.input({ shape: [1], name: 'myInput2' });
  let myInput2Dense1 = tf.layers.dense({ units: 30, name: 'myInput2Dense1', activation: activationMethod }).apply(myInput2);
  let myInput2Dense2 = tf.layers.dense({ units: 20, name: 'myInput2Dense2', activation: activationMethod }).apply(myInput2Dense1);
  let myInput2Dense3 = tf.layers.dense({ units: 10, name: 'myInput2Dense3', activation: activationMethod }).apply(myInput2Dense2);
  //const myInput2Dense3 = tf.layers.flatten().apply(myInput2Dense2)

  let myConcatenate1 = tf.layers.concatenate({ axis: 1, name: 'myConcatenate1', activation: 'linear' }).apply([myInput1Dense3, myInput2Dense3]);
  let myConcatenate1Dense4 = tf.layers.dense({ units: 1, name: 'myConcatenate1Dense4', activation: 'linear' }).apply(myConcatenate1)

  model = tf.model({ inputs: [myInput1, myInput2], outputs: myConcatenate1Dense4 });
  model.name = "functional"
  await model.save('localstorage://model')
  functionalTraining()
}
```

Figure 61: Functional model

The second neural network model is the functional model. Inside the functional model, the amount of neurons per layer and the activation method per layer is set. Activation methods have already been explained in detail. So all that remains is for this section to elaborate on the choice of neurons for each layer. The main trend for deciding neuron layers is that each subsequent layer within the current section must be half of the previous layer. Furthermore, the output layer in a regression model should only contain 1 neuron.

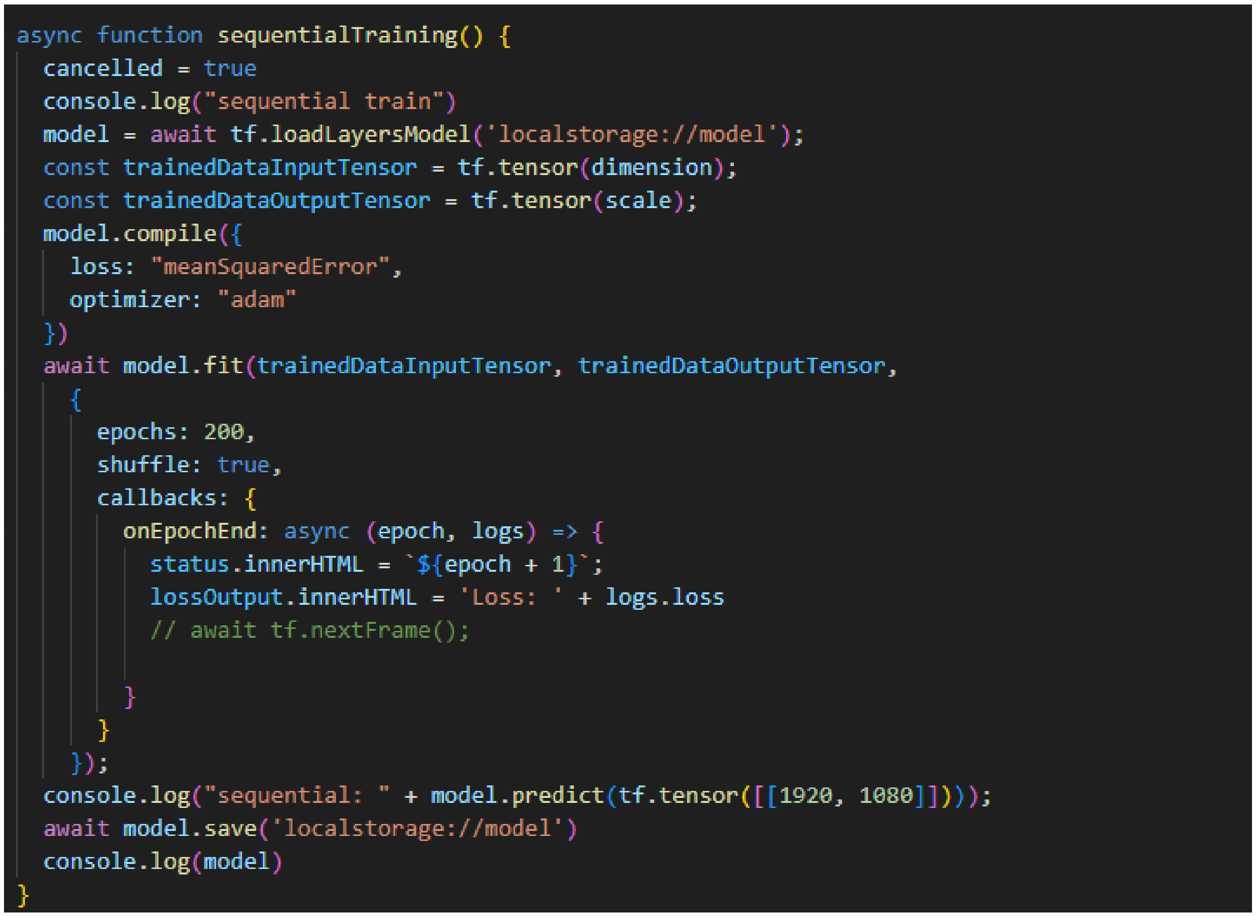


```
async function functionalTraining() {
  cancelled = true
  model = await tf.loadLayersModel('localstorage://model')
  console.log("functional training")
  model.compile({
    loss: "meanSquaredError",
    optimizer: "adam"
  })
  let trainedDataInputTensorWidth = tf.tensor(width);
  let trainedDataInputTensorHeight = tf.tensor(height);
  let trainedDataOutputTensor = tf.tensor(scale);
  let status = document.getElementById("status")
  let lossOutput = document.getElementById('lossOutput');
  await model.fit([trainedDataInputTensorWidth, trainedDataInputTensorHeight], trainedDataOutputTensor,
    {
      epochs: 200,
      shuffle: true,
      callbacks: {
        onEpochEnd: async (epoch, logs) => {
          status.innerHTML = `${epoch + 1}`;
          lossOutput.innerHTML = 'Loss: ' + logs.loss



        }
      }

    });
  console.log(model.summary());
  await model.save('localstorage://model')
}
```

Figure 62: Functional Training

Within the training function, 3 tensors are created. The first 2 tensors: trainedDataInputTensorWidth and trainedDataInputTensorHeight are the input tensors that accept the width and height of the image. The last tensor: trainedDataOutputTensor is the tensor that will output the predicted compression scale. Other variables like status and lossOutput are just there to give the user some form of feedback when the model is still training.

Model.fit is the main function that starts training the model. In this training function, the model goes through 200 epochs worth of training. 1 Epoch is defined as the training model running through and training with all of the available data in the dataset one time [40]. The data will also be shuffled prior to the training. The callbacks in model.fit are for feedback purposes. This section ties into the previously declared variables status and lossOutput. The function will dynamically update status and lossOutput as training goes on. Finally, when the model is done training, it is saved in the local storage.



```
async function sequentialTraining() {
  cancelled = true
  console.log("sequential train")
  model = await tf.loadLayersModel('localstorage://model');
  const trainedDataInputTensor = tf.tensor(dimension);
  const trainedDataOutputTensor = tf.tensor(scale);
  model.compile({
    loss: "meanSquaredError",
    optimizer: "adam"
  })
  await model.fit(trainedDataInputTensor, trainedDataOutputTensor,
    {
      epochs: 200,
      shuffle: true,
      callbacks: {
        onEpochEnd: async (epoch, logs) => {
          status.innerHTML = `${epoch + 1}`;
          lossOutput.innerHTML = 'Loss: ' + logs.loss
          // await tf.nextFrame();

        }
      }
    });
  console.log("sequential: " + model.predict(tf.tensor([[1920, 1080]])));
  await model.save('localstorage://model')
  console.log(model)
}
```

Figure 63: Sequential training

A similar pattern is used for the training of the sequential model as well. The main difference here is that instead of having two input tensors, only one is used due to the nature of the sequential model.

Mean Squared Error is used as the loss function of choice in this project. Mean Squared Error is a function that measures the average squared difference between the predicted value from the neural network and the actual value in the dataset. In other words, Mean Squared Error is used to estimate the difference between the estimated value against the actual value.[41]

There are two variants of the Mean Squared Error. The first variant is called the predictor. The predictor is responsible for determining the quality of a prediction where a sample of values is inserted into the machine learning model, and a new value is predicted. The second variant is called the estimator. The estimator variant of Mean Squared Error is only used when an entire sample of data is used in order to estimate a specific value of a population-oriented data. [41]

The formula for Predictor Mean Squared Error is given as such

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y_i}\right)^2.$$

Figure 64: Predictor Mean Squared Error formula

The formula for Estimator Mean Squared Error is given as such

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{E}_{\theta}\left[(\hat{\theta} - \theta)^2\right].$$

Figure 65: Estimator Mean Squared Error formula

In this case, Predictor Mean Squared Error is the variant that will be automatically selected by TensorFlowJS since the nature of this project falls into the predictor requirements of Mean Squared Error.

Another popular variant of loss functions that are available to TensorFlowJS are Cross Entropy loss functions. However, Cross Entropy loss functions are only used in classification problems, which is not applicable in this scenario. Hence Cross Entropy loss function will not be considered at all.

The difference between loss and accuracy of neural networks must also be mentioned. The accuracy of a neural network can be understood as it is. It depicts how accurate the neural network is in terms of %. The higher the amount of correct predictions, the higher the accuracy level. However, accuracy can only be clearly depicted when there is a universal fact at play. For example, a neural network model predicts that an image is a cat, when it is in fact a dog. In this case, it can be said that the neural network is factually incorrect with no room for debate. But the conclusion is different when a continuous range of numbers are at play [42]

This is where loss functions come in. Loss functions account for probabilities and uncertainties when comparing the prediction value from the true value. Loss functions are a summation of errors made from each sample and prediction. This summation is calculated using the formulas mentioned in the paragraph above. [42]. Hence we arrive at the conclusion that accuracy can never be used in regression problems, which is the reason why this paper omitted accuracy results during testing and coding.

The second aspect that needs to be discussed are optimizers. Optimizers are functions that adjust weights in a neural network as training is being performed. By doing this, optimizers help reduce loss in the neural network. Different optimizer functions can also impact the convergence speed, stability, and the overall performance of a neural network [43]. Multiple optimizer functions exist, to name a few: Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent, Adaptive Moment Estimation (Adam) and Root Mean Square Propagation (RMSprop). For this project, the optimizer function Adam was selected. There are multiple reasons for this. The first being that Adam is an extension of the already popular optimizer SGD. Adam dynamically adjusts learning rates based on each individual weight inside the neural network. Additionally, Adam combines features found in RMSprop to further provide effective updates to the neural network during training [42]. In essence, the Adam optimizer function utilizes the features of its predecessors and improves upon it. Adam optimizer is widely regarded as the most popular optimizer as of the current. It boasts a fast running time and also requires a miniscule amount of memory to execute

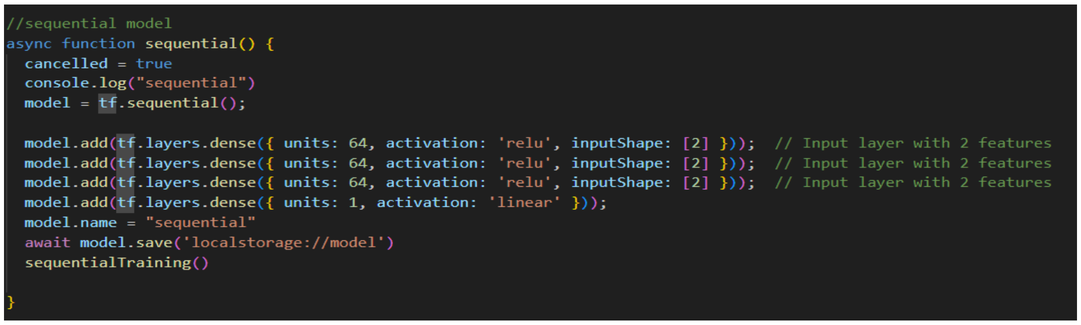
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta w_t} \right] v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\delta L}{\delta w_t} \right]^2$$

Figure 66: Formula for Adam optimizer

The formula for Adam optimizer is as such. B1 and B2 state the decay rate of the gradients during training.

3.6 Compression

After obtaining the dataset and training the model with the data, the prerequisite conditions have been met. Predictions can now be made with the neural network



```
//sequential model
async function sequential() {
  cancelled = true
  console.log("sequential")
  model = tf.sequential();

  model.add(tf.layers.dense({ units: 64, activation: 'relu', inputShape: [2] }));  // Input layer with 2 features
  model.add(tf.layers.dense({ units: 64, activation: 'relu', inputShape: [2] }));  // Input layer with 2 features
  model.add(tf.layers.dense({ units: 64, activation: 'relu', inputShape: [2] }));  // Input layer with 2 features
  model.add(tf.layers.dense({ units: 1, activation: 'linear' }));
  model.name = "sequential"
  await model.save('localstorage://model')
  sequentialTraining()

}
```

Figure 67: Coding for sequential model

Two neural network model types were used in this test. The first neural network type is the sequential model. The sequential model is a type of model that generally has a plain stack of layers and these layers have exactly one input and one output [44]. From this statement alone, it should be obvious that sequential models are not suitable for this test since this test requires the input of both the width and height of an image, which means two inputs. However, an attempt was still made to try and figure out what would happen if the layers were forced to accept 2 inputs by setting inputShape to 2.

Image size	Scale predicted
2048 x 1536	0.5389493107795715
850 x 638	0.24782826006412506
1280 x 800	0.2857484817504883
1920 x 1080	0.5127890706062317
1020 x 510	0.3437691926956177

Table 3 Results of prediction using Sequential Model

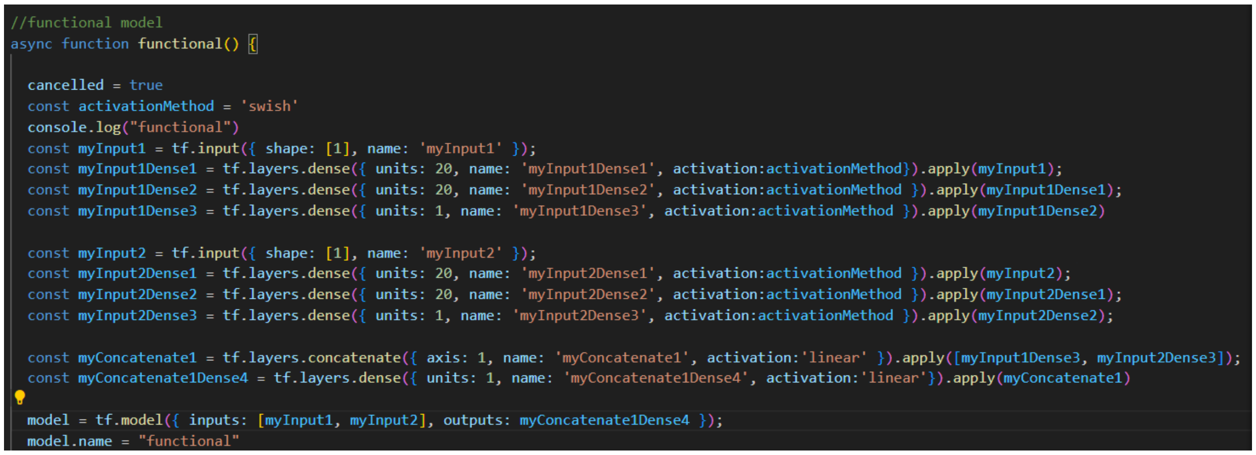
Overall, the results look fine at first glance. However, the 5 images used as input are images that are already part of the training dataset. If a whole new image were to be added to the input, something like this would happen:

```
2.8954875469207764

Original: 15.5 Compressed: 17.823

Width: 512 Height: 512
```

Figure 68: Result of testing data

A scale of 2.89 was predicted, resulting in the compressed image’s file size being larger than the original file size. Therefore, it was concluded that sequential models were not feasible for this test.

Next is the functional model. As previously mentioned, the functional model is derived from the Keras API. It is designed to handle multiple inputs by using non-linear topology as well as sharing layers [20]. Functional model fixes the issue of multiple inputs mentioned previously in the sequential model.



```
//functional model
async function functional() {

  cancelled = true
  const activationMethod = 'swish'
  console.log("functional")
  const myInput1 = tf.input({ shape: [1], name: 'myInput1' });
  const myInput1Dense1 = tf.layers.dense({ units: 20, name: 'myInput1Dense1', activation:activationMethod}).apply(myInput1);
  const myInput1Dense2 = tf.layers.dense({ units: 20, name: 'myInput1Dense2', activation:activationMethod }).apply(myInput1Dense1);
  const myInput1Dense3 = tf.layers.dense({ units: 1, name: 'myInput1Dense3', activation:activationMethod }).apply(myInput1Dense2)

  const myInput2 = tf.input({ shape: [1], name: 'myInput2' });
  const myInput2Dense1 = tf.layers.dense({ units: 20, name: 'myInput2Dense1', activation:activationMethod }).apply(myInput2);
  const myInput2Dense2 = tf.layers.dense({ units: 20, name: 'myInput2Dense2', activation:activationMethod }).apply(myInput2Dense1);
  const myInput2Dense3 = tf.layers.dense({ units: 1, name: 'myInput2Dense3', activation:activationMethod }).apply(myInput2Dense2);

  const myConcatenate1 = tf.layers.concatenate({ axis: 1, name: 'myConcatenate1', activation:'linear' }).apply([myInput1Dense3, myInput2Dense3]);
  const myConcatenate1Dense4 = tf.layers.dense({ units: 1, name: 'myConcatenate1Dense4', activation:'linear'}).apply(myConcatenate1)

  model = tf.model({ inputs: [myInput1, myInput2], outputs: myConcatenate1Dense4 });
  model.name = "functional"
```

Figure 69: Functional model code

Since there are two inputs, two branches of layers will need to be created. Within these 2 branches, 3 layers will be given to each input branch, and these branches will ultimately concatenate at the end, resulting in one dense output layer.

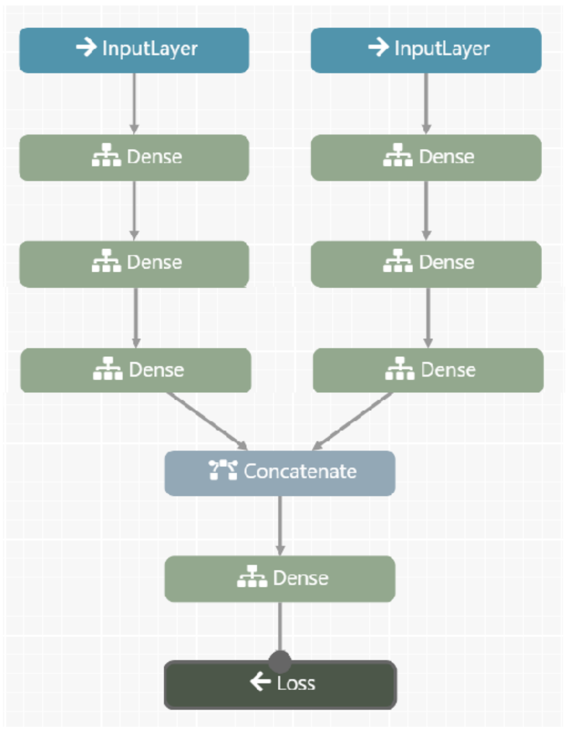


Figure 70: Simple diagram of neural network model

Above is the picture depicting the model. Inside each layer, there will be a specified activation function which will be tweaked to find out the optimal setting. Furthermore, the units of neurons will also be tweaked to find out the optimal amount.

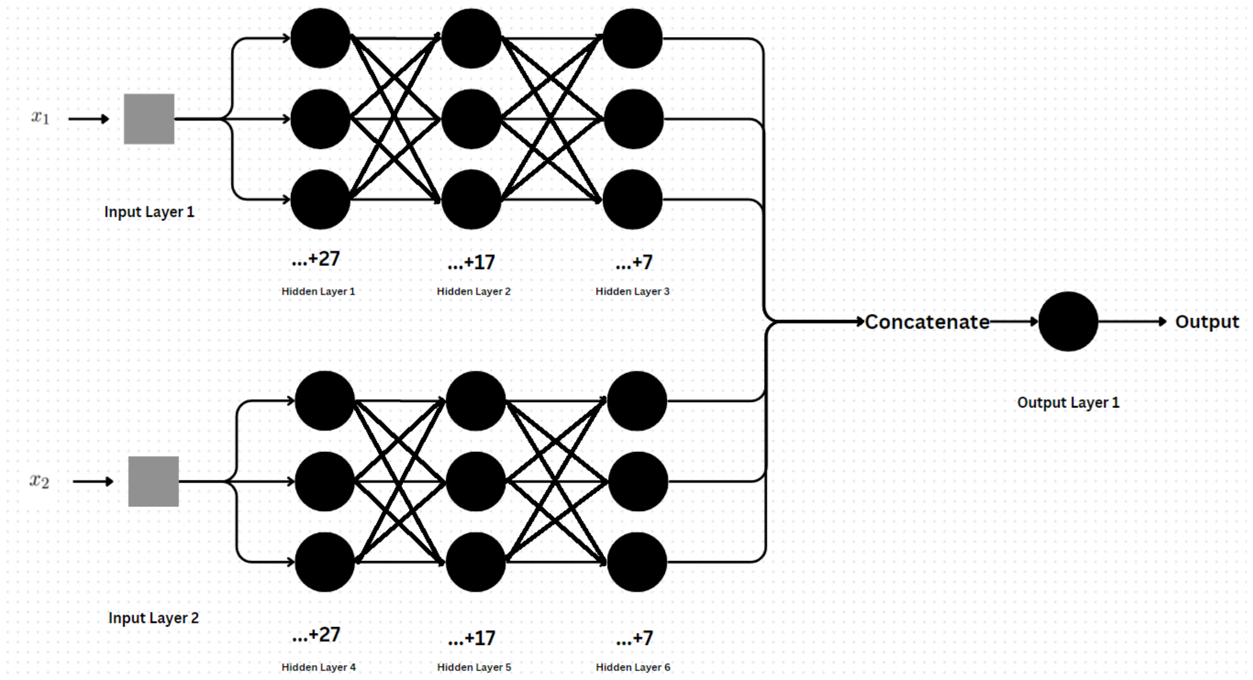


Figure 71: Structure of the neural network model

3.7 Testing Method

In terms of the activation methods used, all variants of ReLU, Mish and Swish will be used for testing in the hidden layers. As for output layers, Linear will be used. Each activation method will be tested for 100 images that are compressed and it will be compared against a version that is not compressed. This means that for every time an activation method is tested, a "control run" must also be performed where the power reading of the non-compressed version must also be tested. The power difference will be recorded. Each activation method will be tested for 5 times. The training model will be trained for 200 epochs every time the activation method changes, and the acceptance criteria for the model is to ensure that the model has a loss of 0.05 and below. If the model has a loss value of more than 0.05, then the model will be redeclared and trained again from scratch. The model's loss is a way to measure the summation of errors in the model, the lower the loss function, the better the model [45].

100 Images will be the amount selected in order to achieve the most consistent result. This section will be explored more in depth later on in the results section. 200 epochs was selected for training because this amount was found to be the best amount of training iterations for a dataset of 400. Anything more than 200 may or may not result in overfitting for certain activation methods.

Another interesting phenomenon that may happen due to excessive amounts of training epochs is that all of the images end up having the same compression scale, which works fine for some sizes, but produces terrible results for others.

The 100 images used are not 100 unique images, but instead are 10 unique images generated for 10 iterations.

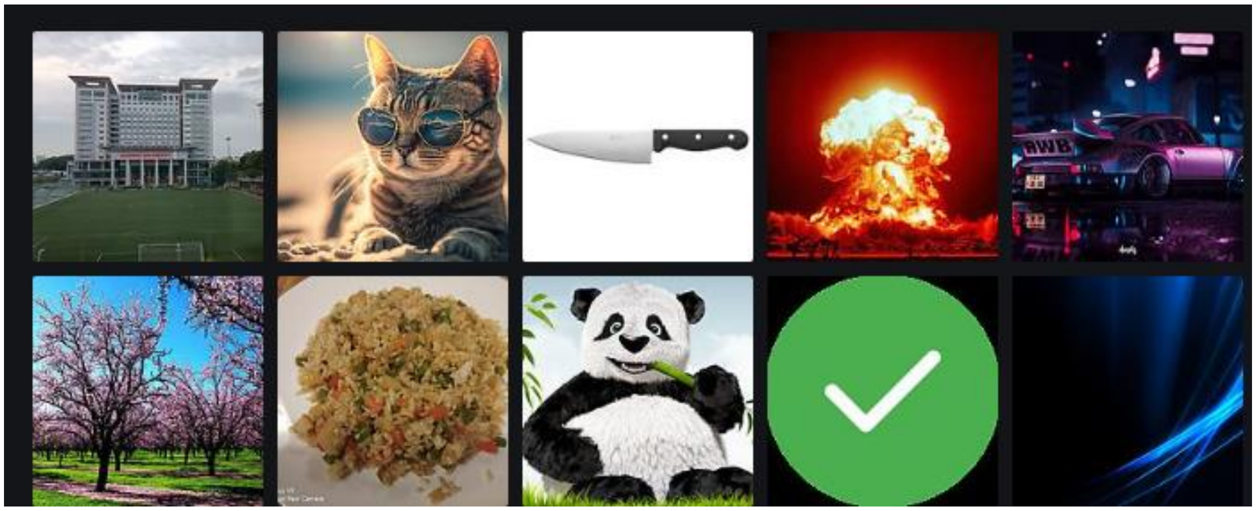


Figure 72: Testing data

The previous 6 images from the training dataset will be used, along with 4 new images to test if the neural network can properly predict images with dimensions that are not within its training dataset.

3.8 Power Profiling

Only 3 applications were opened during the test, these applications being Firefox, Visual Studio Code and the Command Prompt console that is used to host the servers. Within Firefox, only 3 tabs are opened: the 2 development servers, and Google Docs, which is used to record the results. The test is performed like so. First, there are the 2 development servers that were previously mentioned, one contains the compression while the other does not. From here onwards, they will be referred to as “Compressed” and “Not Compressed” tabs. The compressed tab is then reloaded and 100 images are processed. Simultaneously, the power consumption for this tab is measured. After the compressed tab has finished processing and fully loaded the images, the same action is done to the not compressed tab. Firefox measures the power consumption for the code to process

100 images, but without any compression algorithm. After that, the profiler tab containing all the results are opened

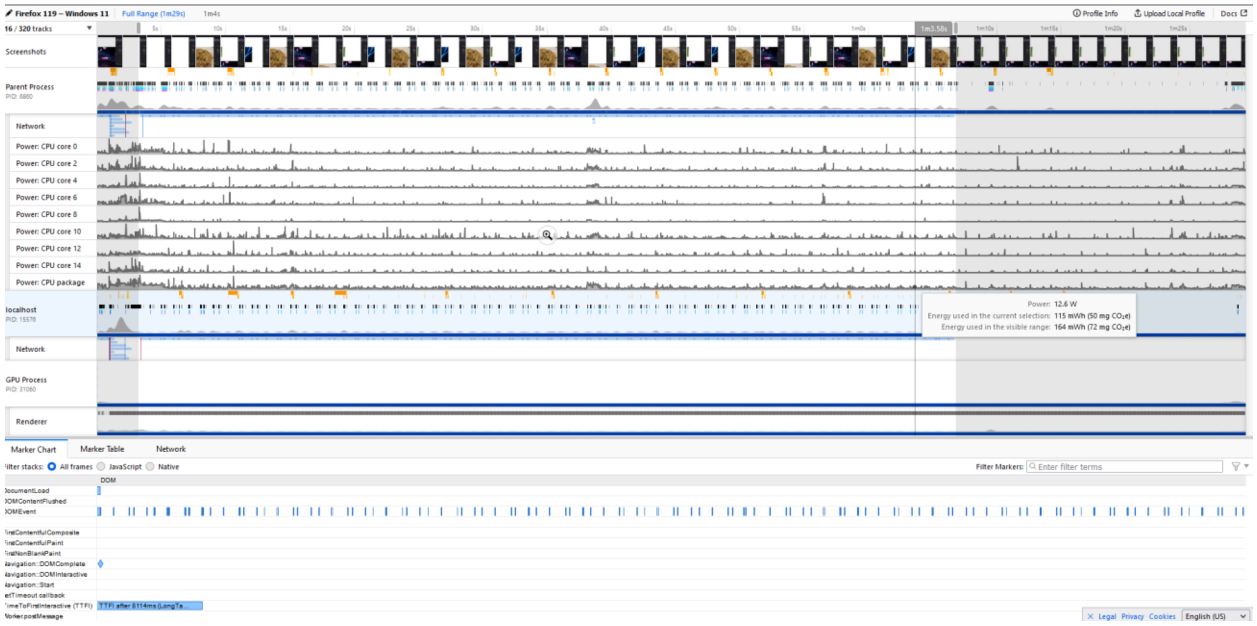
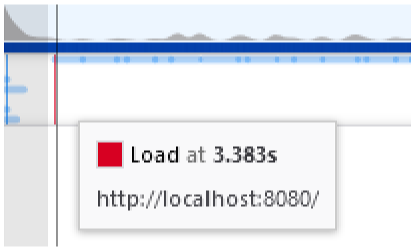


Figure 73: General overview of the power statistic

The power statistics can be divided into 4 parts. The first part is the entire left side of the graph. The left side contains the labels of the power statistics. This part includes the labels of the screenshots, information about each CPU core, GPU processes, and network traffic. The information about the statistics are shown in the middle part. This part contains the graph and the power consumption throughout the tab's lifecycle pertaining to each CPU's core, with the last one being the power consumption for the entire CPU's package. The part underneath that contains the information about the network protocols that were called throughout the tab's lifecycle. This part is important because it provides information on the specific timeframe at which the tab has finished loading, and also tells researchers the exact timeframe at which the website stops requesting for images. The marker chart section at the bottom states DOM events and specific DOM loading content, however this part will not be relevant since the information here is out of scope.

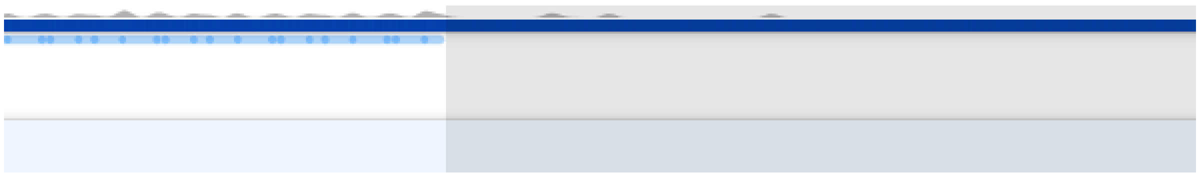
Inside the profiler tab, we can see the power consumption throughout the tab's lifecycle. Only the parts that are not grayed out are taken into consideration. This is because anything outside of the area that is grayed out is not part of the timeframe desired, which means it is just idle time. To determine which part of the graph is actually relevant, we have to look at the network section.



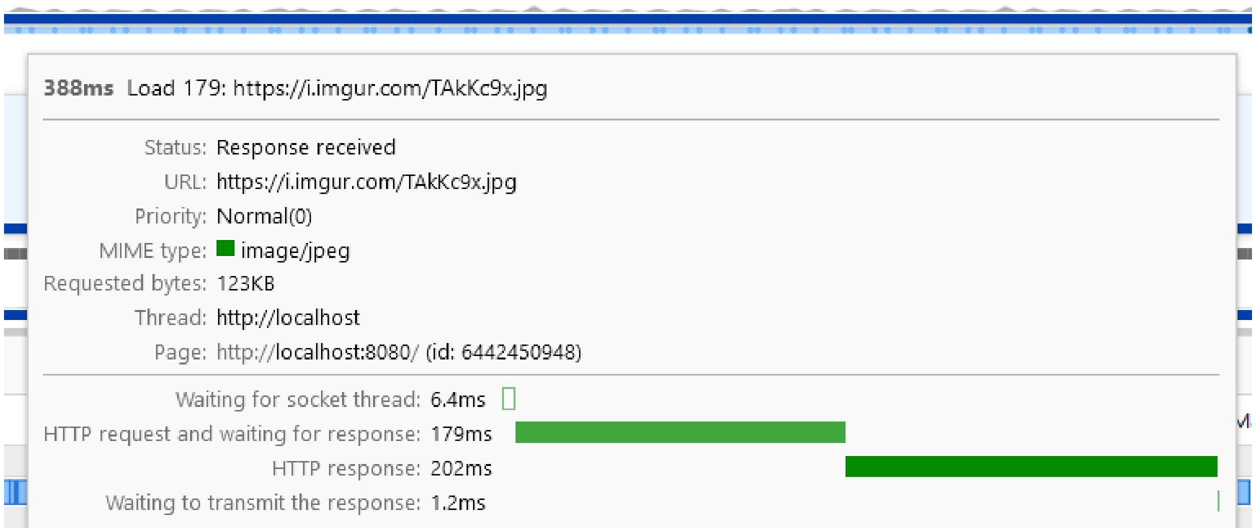
🟥 Load at 3.383s

http://localhost:8080/

The part with the red line is the part where we want to start measuring. This is the part where the tab starts to get loaded in.



To determine the point where we stop recording the results, we have to look at the blue bars of the network.

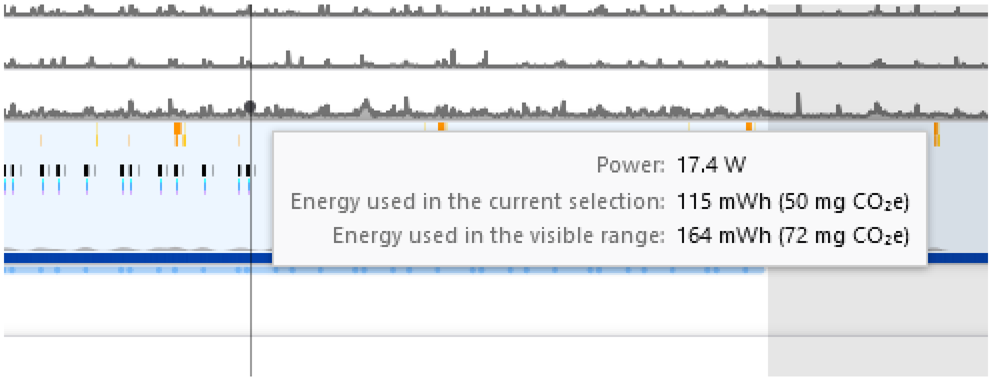


388ms Load 179: https://i.imgur.com/TAkKc9x.jpg

Status: Response received
URL: https://i.imgur.com/TAkKc9x.jpg
Priority: Normal(0)
MIME type: 🟩 image/jpeg
Requested bytes: 123KB
Thread: http://localhost
Page: http://localhost:8080/ (id: 6442450948)

Waiting for socket thread: 6.4ms
HTTP request and waiting for response: 179ms
HTTP response: 202ms
Waiting to transmit the response: 1.2ms

The blue bars represent the load protocol of the tab. After the blue bar ends, it means that there are no more images to be loaded in, which signifies that the tab has successfully completed the operation of loading and compressing 100 images.


Power: 17.4 W
Energy used in the current selection: 115 mWh (50 mg CO_2e)
Energy used in the visible range: 164 mWh (72 mg CO_2e)

What we are ultimately interested in is the section shown in the picture above. This section contains the energy used in the current selection. The graph shows power consumed for all the available cores of the CPU, as well as one additional section called the CPU package. The energy used for the CPU package will be the reading that is observed in this experiment since it represents the whole package of the CPU.

4.0 Results

Relu (0.04 loss):

Compressed/100: 100, 102, 97, 108, 116 (mWh): 104.6 (Average)

NonCompressed/100: 105, 98, 101, 93, 110 (mWh): 101.4 (Average)

Width	Height	Original (KB)	Compressed (KB)	Compression Scale
244	207	7.02	4.5	0.2687111496925354
225	225	1.98	1.749	0.2678743600845337
540	360	50.6	15.162	0.319317489862442
3211	2408	1630	616.106	0.833739697933197
512	512	15.5	7.027	0.3289552628993988
1020	510	125	46.355	0.39287200570106506
1920	1080	231	133.292	0.555811882019043
1280	800	43.2	24.022	0.4508041441440582
850	638	263	142.291	0.3825157582759857
2048	1536	208	173.154	0.6114535331726074
Total File Size (x10 iterations):		25753	11632.59	

Table 4: ReLU Results

Elu (0.038 loss)

Compressed/100: 101, 108, 98, 117, 97 (mWh): 104.2 (Average)

NonCompressed/100: 105, 110, 91, 113, 106 (mWh):105.2 (Average)

Width	Height	Original (KB)	Compressed (KB)	Compression Scale
244	207	7.02	5.61	0.4375995695590973
225	225	1.98	2.279	0.4483882486820221
540	360	50.6	20.237	0.4861578047275543
3211	2408	1630	329.514	0.4333648383617401
512	512	15.5	10.404	0.5410129427909851
1020	510	125	57.529	0.48765841126441956
1920	1080	231	127.088	0.510718047618866
1280	800	43.2	32.727	0.5321548581123352
850	638	263	174.22	0.5407741665840149
2048	1536	208	172.671	0.5385963916778564
Total File Size (x10 iterations):		25753	9322.79	

Table 5: Elu Results

Relu6 (0.043 loss)

Compressed/100:   96, 103, 105, 102, 103 (mWh): 98 (Average)

NonCompressed/100:103, 103, 100, 99, 101 (mWh): 101.2 (Average)

Width	Height	Original (KB)	Compressed (KB)	Compression Scale
244	207	7.02	18.126	-0.035092826932668686
225	225	1.98	1.755	0.07879802584648132
540	360	50.6	16.202	0.3563464879989624
3211	2408	1630	366.588	0.532832145690918
512	512	15.5	10.377	0.532832145690918
1020	510	125	59.207	0.532832145690918
1920	1080	231	131.322	0.532832145690918
1280	800	43.2	32.722	0.532832145690918
850	638	263	172.655	0.532832145690918
2048	1536	208	172.212	0.532832145690918
Total File Size (x10 iterations):		25753	9811.66	

Table 6: ReLU6 Results

Selu (0.037 loss)

Compressed/100:   110, 99, 99, 96, 98 (mWh): 100.4 (Average)

NonCompressed/100: 113,107,102, 99,101 (mWh): 104.4 (Average)

Width	Height	Original (KB)	Compressed (KB)	Compression Scale
244	207	7.02	6.374	0.5716161727905273
225	225	1.98	2.414	0.5919761657714844
540	360	50.6	20.293	0.5082711577415466
3211	2408	1630	393.896	0.6043425798416138
512	512	15.5	10.337	0.5274511575698853
1020	510	125	57.601	0.49861815571784973
1920	1080	231	129.1	0.5190174579620361
1280	800	43.2	32.031	0.5182079672813416
850	638	263	170.24	0.5206626653671265
2048	1536	208	177.116	0.5629980564117432
Total File Size (x10 iterations):		25753	9994.02	

Table 7: SeLU Results

Mish (0.044)

Compressed/100: 98, 93, 99, 97, 100 (mWh): 97.4 (Average)

NonCompressed/100: 100, 98,103, 99, 108 (mWh): 101.6 (Average)

Width	Height	Original (KB)	Compressed (KB)	Compression Scale
244	207	7.02	4.596	0.283345103263855
225	225	1.98	2.119	0.310484379529953
540	360	50.6	18.144	0.412820428609848
3211	2408	1630	329.514	0.4337911307811737
512	512	15.5	9.962	0.47986772656440735
1020	510	125	48.883	0.4246777892112732
1920	1080	231	115.714	0.34785160422325134
1280	800	43.2	30.478	0.4173198342323303
850	638	263	158.449	0.46043217182159424
2048	1536	208	116.846	0.36646318435668945
Total File Size (x10 iterations):		25753	8347.05	

Table 8: Mish Results

Swish (0.048)

Compressed/100:   103, 98, 108, 100 , 102 (mWh): 102.2 (Average)

NonCompressed/100: 103, 110, 100, 105, 98 (mWh): 103.2 (Average)

Width	Height	Original (KB)	Compressed (KB)	Compression Scale
244	207	7.02	5.804	0.46826082468032837
225	225	1.98	2.295	0.46780088543891907
540	360	50.6	20.168	0.515153706073761
3211	2408	1630	433.598	0.6732164025306702
512	512	15.5	10.462	0.5452682375907898
1020	510	125	58.246	0.5204076170921326
1920	1080	231	133.518	0.5458753705024719
1280	800	43.2	33.075	0.5480127334594727
850	638	263	176.397	0.5507338643074036
2048	1536	208	183.792	0.6013191342353821
Total File Size (x10 iterations):		25753	10573.55	

Table 9: Swish Results

Energy Consumed

Activation/Non Compressed	Power Consumption (mWh)					Average	%difference (1 d.p)	%size reduction	Loading Time
ReLU (0.04 loss)	100	102	97	108	116	104.6	-3.0%	54.8%	2m 20 sec
Non Compressed:	105	98	101	93	101	101.4			2m 15 sec
Elu (0.038 loss)	101	108	98	117	97	104.2	+0.9%	63.7%	2m 20 sec
Non Compressed:	105	110	91	113	106	105.2			2m 10 sec
Relu6 (0.043 loss)	96	103	105	102	103	101.8	-0.5%	61.9%	2m 30 sec
Non Compressed:	103	103	100	99	101	101.2			2m 20 sec
Selu (0.037 loss)	110	99	99	96	98	100.4	+3.8%	61.1%	2m 20 sec
Non Compressed:	113	107	102	99	101	104.4			2m 25 sec
Mish (0.044 loss)	98	93	99	97	100	97.4	+4.1%	67.5%	2m 30 sec
Non Compressed:	100	98	103	99	108	101.6			2m 20 sec
Swish (0.048 loss)	103	98	108	100	102	102.2	+0.9%	58.9%	2m 25 sec
Non Compressed:	103	110	100	105	98	103.2			2m 30 sec

Table 10: Overall energy consumption and other data

All power consumptions are shown in Megawatt Hour (mWh). Megawatt Hour is a unit that shows how many kilowatts of energy is being consumed per hour. The average of the results were calculated, and then the percentage differences between the activation and non-compressed variants were stated. The file size reduction difference was also calculated in percentage.

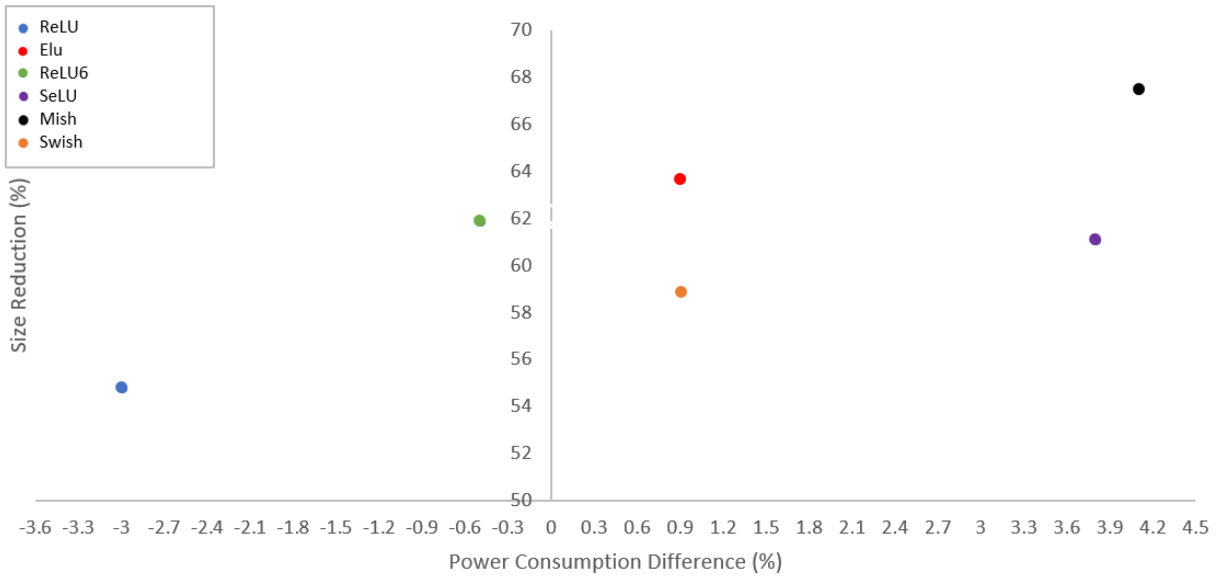


Figure 74: Chart of Size Reduction against Power Consumption Difference

This chart shows the difference of activation methods with file size reduction and power consumption differences in mind. In terms of standing, activation methods that lean towards the upper right quadrant will be superior, while activation methods that lean towards the lower left quadrant are worse in quality.

5.0 Discussion

5.1 Result analysis

Overall, Elu, SeLU, Mish and Swish had a positive impact on the power consumption. ReLU and ReLU6 both caused a higher increase in power consumption. This could be due to the fact that ReLU and ReLU6, which is a simpler and lower performance of ReLU had problems like vanishing gradient problems. Elu and SeLU are both improved versions of ReLU, so these problems were either mitigated or completely fixed. Elu fixes it by introducing negative values during training [22]. While SeLU injects self normalization into the model and also accepts negative values [24]. Whereas Mish and Swish are introduced as direct upgrades over ReLU [32].

One anomaly in the data can be seen during the ReLU6 run. During this run, the neural network model ended up predicting a scale of -0.03 for an image with the size of 244x207, which is the smallest image in the test run. ReLU6 does not consider negative values at all, so theoretically this would not be possible. The only explanation here would be due to the interference of the output layers that utilize linear activation. Linear activation is unbounded in both positive and negative domains. During training, although all the values learnt by the weights throughout the hidden layers will be positive, a few of the weights passed through the linear activation method resulted in a negative value instead.

The reason why multiple iterations of non-compressed must be executed for every activation method testing is due to how Firefox measures power. Firefox does not purely measure the power consumed for Firefox only. Rather it measures the power consumed for the entire time frame. This means that if there are multiple applications running in the background when the test is performed, Firefox will pick up on those power readings and factor them into the final result. Hence in order to have accurate data testing, multiple tests must be performed and the average is calculated to ensure that the variables affecting the test run are kept at a minimum.

To better calculate and see the true impact of the compression, the percentage difference between the non-compressed version and the compressed version for that cycle was calculated. Overall, Mish gives the biggest boost in performance with a difference of +4.1% in power consumption, while ReLU did the worst with having -3.0% difference in power consumption. In layman terms, it simply means that Mish reduced power consumption by +4.1% while ReLU did the opposite and increased power consumption by 3.0%. ReLU reduced file size from 25753 to 11632.59, Elu reduced it to 9322.79, ReLU6 reduced it to 9811.66, Selu reduced it 9994.02, Mish reduced it to 8347.05 and Swish reduced it to 10573.55. The file size for all of these have been multiplied by 10 due to the code running 10 iterations of 10 images.

This begs the question, even though all of the file sizes were ultimately decreased, why is it that some activation methods still cause power to increase instead? The reason for that would lie in the very act of predicting compression scales. By predicting the scale, power is also used, which will be picked up by the power profiler in Firefox. Running additional code to compress and reduce file size image is not free, it will cost the processor to process the information required to predict the scale and then compressing the actual image itself. This leads to another important point that needs to be discussed. There is a reason why 100 images were used.

A side test was done with Mish activation for 3, 4 and 5 iterations respectively. The same testing methods were used, 5 test cycles for compressed and 5 more for non-compressed. This step was repeated for each iteration number.

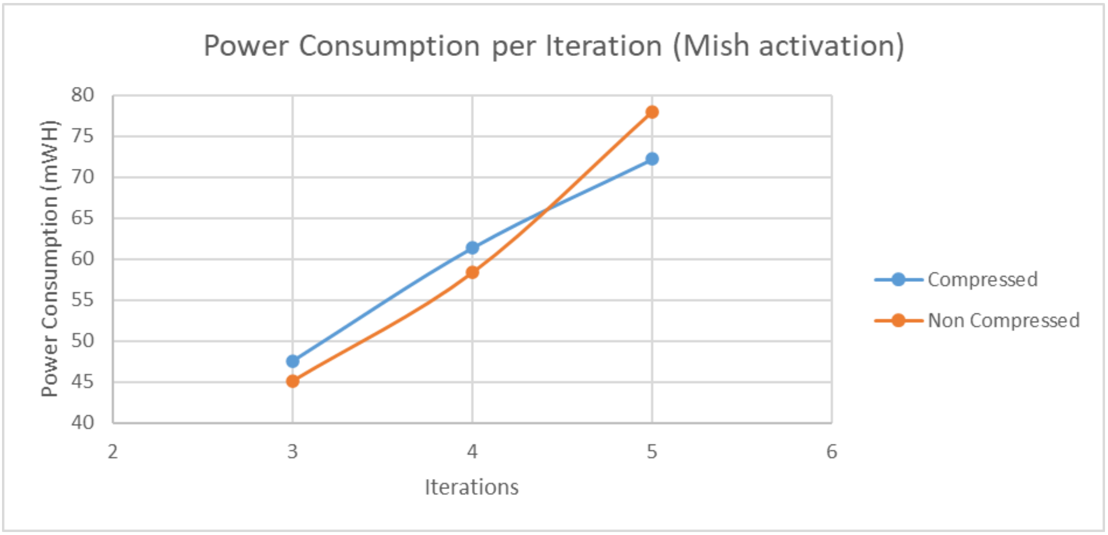


Figure 75: Power Consumption per Iteration with Mish activation

From the graph above, we can see that at an image count of 30 to 40, the power consumption by the compressed version consistently ranks higher than the one that is not compressed. Only at 50 images can we see improvements brought on by the compression. This result shows that there is a point where the power consumption from predicting "breaks even" due to compressed images using lesser power. This point can be found at 5 iterations and above. To remove any hidden factors that might be present, 10 iterations were used in the main testing method.

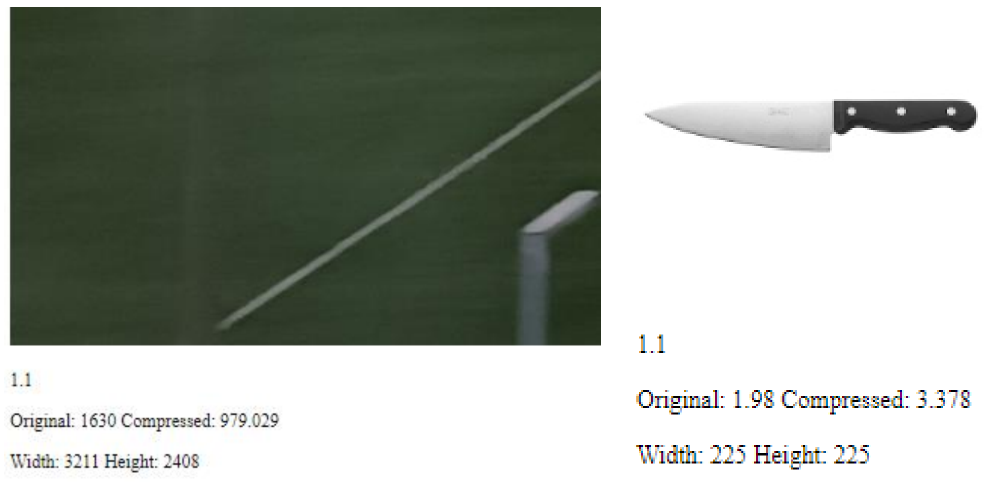
This leads to a problem, would this proposed solution to reducing power consumption still be applicable in real world situations? Would webpages realistically have more than 50 images and above? Thankfully, the proposed solution can still see use in very specific website genres like e-commerce. E-commerce websites like Shopee or Lazada load in huge amounts of images when consumers are scrolling the website. The proposed model can aid in reducing power consumed when rendering these images onto the website. Other things like banner art and logos within the website can also be compressed if done properly, which also leads to reduced power consumption without affecting user experience.

One prevailing question would be why not just use the lowest possible compression scale for all images and remove the need to predict in general? The reason for that is because one of the main objectives of this paper is to ensure a reduction in power consumption without affecting the image quality by a drastic amount. Hence why a proper dataset had to be gathered. When data is being inserted into the dataset, the participants were asked to judge the image with quality in mind. This eliminates compression scales that would provide exceptional improvement to the file size, but sacrifices image quality in the process. The goal of the neural network is to achieve a balance between the two.

The original consensus was that a lower resulting file size would ultimately mean a higher power consumption reduction. However, the results have proven this hypothesis wrong. ReLU6 reduced the overall size by 61.9% but caused an increase in power consumption by 0.5%. Swish on the other hand reduced the file size by only 58.9%, but still managed to result in a positive reduction in power consumption by 0.9%. This finding leads to an interesting theory. Different activation methods will not only lead to different power consumption when training, but could also affect neural networks when making a prediction. Activation functions are used to determine which neuron will be activated when making a prediction, as well as modifying the weights between layers (). If activation functions determine which neuron is activated during prediction, it could mean that different activation functions are more efficient compared to others when trying to make a prediction. Mish and Swish are advanced activation functions that are said to be a direct improvement over ReLU [46] . This could explain why Swish still gave a positive result when predicting even though the percentage of file size reduction was lower than ReLU. Mish and Swish manage to make predictions by using lesser neurons in the neural network model, which in turn resulted in lower power consumption overall despite a bigger image file size being rendered onto the webpage.

A comparison can be made towards brains. Neural networks are designed around the concept of how the human brain works. The human brain is said to have a signal for energy request that uptakes glucose to provide the energy for the brain, and this signal occurs when the brain is trying to focus on processing important information [47]. This suggests that the brain's neuron does indeed fluctuate in power when trying to process difficult information. Similarly, when the brain has fully processed something, neural pathways will be established and the same information will require less effort to recall, thus requiring lesser power. The same logic can be applied to neural network models. Different activation functions could wire the neural network in different ways. It can be theorized that the Mish activation function teaches the neural network to process compression scales in an efficient manner. This means that the neural network will end up having a better understanding of the problem overall and will use up lesser power when trying to predict a scale overall.

The canvas.toDataURL compression function generally accepts a scale from 0 to 1, however, there are specific cases where compression scales surpassing the range of 0 to 1 will still yield a positive result on trimming down file size


1.1

Original: 1630 Compressed: 979.029

Width: 3211 Height: 2408

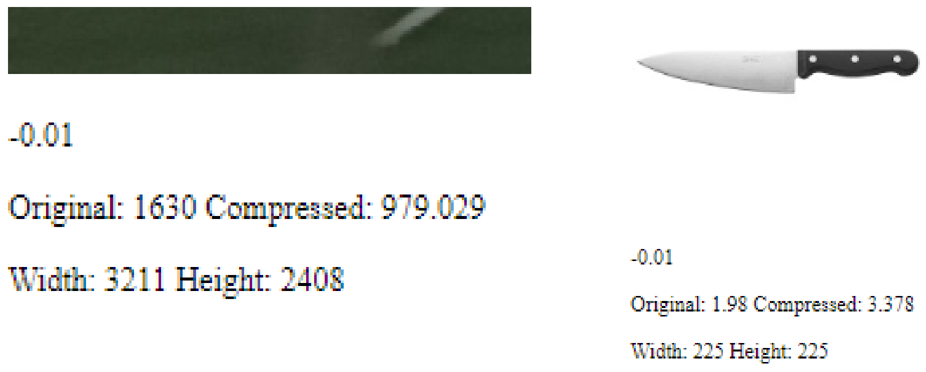
1.1

Original: 1.98 Compressed: 3.378

Width: 225 Height: 225

Figure 76: Result of 1.1 compression scale

A test run was done with a compression scale of 1.1. On the right is an image of size 3211x2408. A small part of it has been selected for illustration purposes. It can be seen that even though the compression scale is 1.1, the compressed size is still smaller than the original size. On the right is an image with a size of 225x225. In this scenario, it can be seen that the image size went up from 1.98 to 3.378, which is nearly double the original size due to the compression scale of 1.1


-0.01

Original: 1630 Compressed: 979.029

Width: 3211 Height: 2408

-0.01

Original: 1.98 Compressed: 3.378

Width: 225 Height: 225

Figure 77: Result of -0.01 compression scale

From these 2 images, it can be seen that the exact same file size occurs as when a scale of -0.01 is used. And the same phenomenon happens where small images end up becoming larger than their original size. From this, it can be seen that the canvas.toDataURL function processes scales above 1 or below 0 in an inconsistent manner. Previously it was thought that the function would just automatically make every image bigger if an unsuitable scale was used. More testing needs to be done on this section to truly determine the cause of this phenomenon

Initially, it was thought that the loading time of the website would be impacted in some way. It could either increase due to additional time spent on compression, or decrease since there is less information to render onto the website. However, the results obtained were inconclusive and do not point to any clear results. ReLU loaded the website 5 seconds later compared to the non-compressed variant for that run. Elu and ReLU6 both loaded the website 10 seconds later, SeLU loaded the website 5 seconds faster, Mish loaded the website 10 seconds later and Swish loaded the website 5 seconds faster. Mish, which was thought to be the best throughout the entire test run, still resulted in a 10 second delay when loading the website. Swish, which is said to be a more inferior version of Mish in terms of prediction, ended up loading the website 5 seconds faster than Mish. It can be theorized again that it had something to do with the way the neurons are fired in the neural network, which contributes to computational power and therefore affects loading time. But the previous theory concluded that Mish had the upper hand since it was the more efficient activation method that leads to efficient neurons firing. Ultimately, it can be concluded that the processing done by the neural network is not significant enough to offset processing time of the website by an amount noticeable enough. Thus, the different loading times for the test result can be chalked up to 1 reason. The reason being network fluctuations when loading the images from Imgur. As previously mentioned, 10 images were loaded from Imgur 10 times for both compressed and uncompressed variants. Since this step of the process deals with the network, which is known to not be 100% stable and constant in terms of speed due to bandwidth and traffic within the internet. In short, lag could be the reason for different website loading time.

Lastly, it must be mentioned that the dataset was not normalized. This method of action was indeed explored, however it was found that the idea was not feasible as of now due to limited data. Unwanted results would start to appear when attempting to predict compression scales for images that are not within the testing data. The scale predicted was not desirable and the results were overall worse than using a non-normalized dataset.

5.2 Limitations

One of the limitations of measuring power consumed with Firefox's power profiler is that it does not purely measure the power consumed for Firefox only. Rather it measures the power consumed for the entire time frame. This means that if there are multiple applications running in the background when the test is performed, Firefox will pick up on those power readings and factor them into the final result. To circumvent this, a number of constant factors were established.

The second limitation would be the amount of data present in the dataset. Under an ideal scenario, at least 1000 rows in the dataset would be ideal. However, due to time constraints, only 400 rows were able to be collected. In hindsight, it would seem that only the accuracy of the neural network's prediction would be changed, but since various amounts of activation methods were used, it is hard to accurately say what changes might happen. Since different activation methods process data differently, different amounts of data might affect the prediction in a big way. Case in point, the result with ReLU6. ReLU6 ended up predicting negative values for one of the image sizes. Since there is no effective way to look into the thought process and the information learned by the weights in the neural network, there is no definitive explanation for why ReLU6 ended up with a negative value. If more data were to be added to the dataset, there is no guarantee that ReLU6 or other activation methods will not predict any negative values, and the chance that more negative values pops out might even be higher than right now.

Another limitation regarding the dataset is that not all data within the dataset is 100% accurate or desirable. Because the compression scale of the images gathered are judged by humans, it may not be ideal. Humans have different perceptions on what exactly is a "clear" image. One person might think the image passes the test after being compressed, while the other might think that artifacts can be spotted in the image which ends up being unacceptable for them. All of these little factors will contribute to the final dataset and affect the prediction results. Ideally, a computer program with image recognition should be used in order to determine which image passes the criteria, but due to time constraints this idea could not be executed. Furthermore on the topic of quality, image size is realistically not the only factor that should be looked at when performing compression. Image noise, image grain and how much quality loss is already present must also be factored in before compressing. An image with a lot of noise and an image with no noise will handle compression differently, meaning the image with noise might end up being over-compressed and quality will suffer to the point where it cannot be recognized. Different settings and environment with respect to lighting will also affect how much the image can be compressed.

The fourth limitation is the testing scenario. Since Firefox picks up on all power used by the CPU during webpage load, it is impossible to know exactly how much power did the webpage alone use. Which is the reason why all apps were terminated to prevent too many uncontrollable variables. However, in a real world scenario, users will most likely run their webpage with other applications running in the background as well. These applications might also influence the power consumption, and processing time, which might affect hidden factors previously not known or observed. Additionally, the project was only tested on one device with an AMD processor. It is not known how the neural network will function with other processors, and whether or not older processors will be able to give good results like the one obtained during the current test cycle.

The fifth limitation stems from the nature of neural networks. Neural networks will always end up with different results even if the same dataset along with the same number of training iterations are used. This is simply the nature of neural networks and activation functions which introduces randomness to prevent linearity [15]. This means that there might be better versions of results that can be obtained from the activation methods used above. However, this requires setting a special parameter called seed. The seed parameter is a parameter used to ensure that randomly generated values within the system will always return a set, non-random value [48]. If the best possible seed can be found for each activation method, then randomness can be eliminated. However, this approach takes an unreasonable amount of time.

5.3 Implementations in the real world

The current approach of this project is only a proof of concept. However, there are ways to implement the proposed method into the real world. A JavaScript interceptor can be added to a part of the code in the website. Javascript interceptors are code chunks that intercept a HTTP Request and modify it.

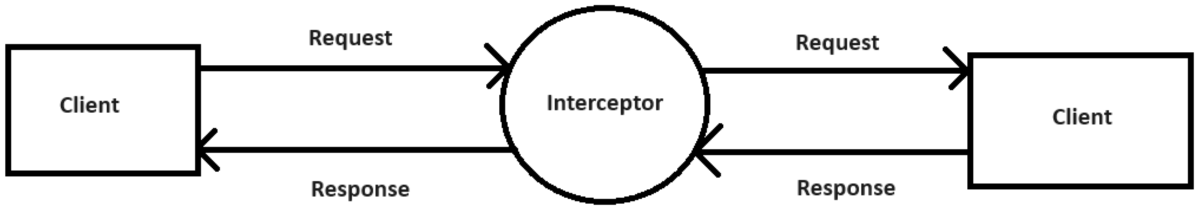


Figure 78: Proposed solution of interceptor

In theory, the interceptor can intercept the image response from the server that the website is requesting from, and then make changes to it before parsing the image back to the client for display in the user interface. To do this, it would mean that the particular chunk of code that intercepts and processes the image must always be on regardless of which part of the website the user is in. To do this, @Injectable declaration must be used before the class that contains the intercept function. By using the @Injectable declaration, it enforces all other classes to have the @Injectable class as a dependency, which means that the @Injectable class will be loaded in every part of the website as an instance. By doing this, it allows the compression function to automatically run and intercept all HTTP Requests. Now, the code just has to check if the request itself is an image, and the function will use the neural network model to predict compression scales to compress the image, as well as compressing all images to JPEG file format to further reduce file size.

5.4 Challenges

Various challenges were faced throughout the lifecycle of this project. The first challenge being the initial setup of the code's runtime environment. JavaScript does not contain a specific IDE that can be used. Furthermore, no virtual machine exists that can be run within an application like Eclipse. Therefore, the runtime environment as well as the packages imported must be self installed through the help of a command prompt in conjunction with Node Package Manager. After extensive research, Webpack was selected as the runtime for JavaScript. Without this, various problems would arise when trying to incorporate the installed modules into JavaScript.

Halfway through development, it was discovered that the initially proposed model: sequential model, could not be used due to the nature of the project. This ultimately resulted in additional research being needed to be carried out since the original plan did not account for this. This obstacle also set development back by a few days since up to this point, most of the code was optimized for sequential models. Thankfully, this setback led to the discovery of a better and more customizable model called the functional model.

The next challenge comes from Tensorflow. Tensorflow has an impressive amount of features built into it. However, the documentation behind it is obscured by large amounts of technical terms that will confuse and intimidate a new user. This issue is further amplified when using the JavaScript variant of Tensorflow. Most communities and forums are used to answering questions for the Python variant of Tensorflow, which misleads search engines into finding solutions for the Python variant of Tensorflow which is not applicable in the JavaScript variant.

Lastly, inaccurate readings were present during the test phase. Occasionally, background apps that were built into Windows 11 caused massive surges in power consumption when testing. This led to inaccurate readings since the power consumed did not come from the tab being monitored. To circumvent this, task manager was used to force stop each background application.

6.0 Conclusion

6.1 Findings

In conclusion, this project managed to answer the pressing question of whether or not a neural network driven solution to image compression could actually reduce power consumption of websites when rendering images. Additionally, a few interesting findings were made. The first being that negative scales can still be predicted despite the fact that every subsequent activation method counteracts the very existence of negative values. However, this phenomenon only occurred once. The difference between neural network models was also understood. Previously, it was thought that despite sequential models having a recommended input of 1, 2 inputs could still be forced with some degree of acceptable accuracy. However, it was instead discovered that sequential models will accurately predict values of inputs conforming to the training dataset, but utterly fail when predicting values outside of the training dataset. Another important finding was that power consumption during the initial training of the neural network is not a big factor when calculating. During the literature review, close attention was paid to the amount of power consumed between each activation function. ReLU6 was theorized to be better at the time due to claims about reduced power consumption over the original ReLU. The same can be said about Swish over Mish. Mish was said to be more computationally heavy since it was said to be more efficient over Swish. Ultimately, the activation method that utilized most modern techniques was the one that had an advantage. This meant that Mish yielded a better result over Swish. The reason for this lies in efficiency. Although being more computationally intensive to train, a better activation method meant better compression scales overtime which meant more power saved in the long run. In the end, the project managed to achieve all 3 objectives. The power consumption for loading pictures was reduced, the best activation method for compression was found with conclusive evidence and power consumption was lowered without drastically impacting image quality. Despite a small difference in power consumption, power consumption was still lowered. Mish ultimately reduced power consumption by 4.1% for 100 images. The only counter-productive aspect about this approach is that the testing method is engineered and tested for 100 images. Although it was found out that the breakpoint where net energy consumed by compressed versions becomes lesser than their non-compressed counterpart at 50 images and above, which can be a deal breaker when considering whether or not to implement the proposed method into a real-world website.

6.2 Summary of Objectives

The best activation method found was Mish. This conforms with all the literature reviews done above. Mish takes into account past problems like vanishing gradient problems and bias shifting. It then counteracts all of these issues by deviating from the standard activation formulas at the cost of computational power consumption. Mish counteracts vanishing gradient problems by allowing itself to be continuously differentiated, and eliminates bias shifts by preserving negative weights throughout training. Lastly, image quality. Overall, image quality was preserved throughout the entire project. All of the images were visible and preserved a good amount of quality while effectively cutting down on the file size. In a sense, the compression scale and effectiveness can be summed up to the lower the scale the better. However, there is a limit for each image size. Mish manages to reach the perfect breakpoint where the compression scale is low enough where the file size is small, but not low enough to the point where the image starts producing visible noise and quality deterioration. This paper achieved the objectives that it set out to achieve, the paper also presented an interesting comparison between activation methods in a field that has yet to be thoroughly explored. Using neural networks to attempt image compression as a website is being loaded has rarely been experimented on. Furthermore, the effects of different activation methods on the resulting scale is also yet to be fully understood. This paper contributes useful findings that can help achieve better power efficiency in the future.

6.3 Future works

In the future, various things can be done to enhance the testing method of the research. Firstly, different loss and optimizers can be experimented on. Although the only acceptable loss function for regression is Mean Squared Error, there exists different variants of Squared Error loss functions that can be used for regression. Furthermore, different optimizers should also be experimented on. Additionally, different activation methods can be used within the input layers. Combinations of different activation methods can be experimented with to find out if it will positively or negatively impact the result of the prediction. More images should be added into the testing pool, along with certain amounts of quality deterioration and noise added into the image. By doing this, the robustness of the neural network can be tested to see how well it handles images that are already bad in quality. To add on to the topic at hand, more input features can also be added to the dataset. The neural network can be trained to account for noise and other deterioration features. However, to accomplish this, a computer vision based criteria judging system must first be made. Humans cannot reliably judge noise and quality deterioration within an image. The way which data has been gathered can also again be improved. The current dataset consists of compression scales judged by the human eye. Furthermore, different people were used to judge and decide which compressed image passes the test. It is common knowledge that humans are not the most objective creatures on earth. Everybody has varying opinions on the clarity of images. To remedy this, computer vision should be used for the image acceptance criteria. By doing this, a criteria for judging image clarity level can be established, and along with it a constant and stable method of testing can be declared to better generate a training dataset for the neural network. By having more data, normalization can be reattempted on the dataset. When having adequate data in the dataset, normalization is said to yield better results than a non-normalized counterpart. Finally, the method on how power is measured can also be improved in the future. A new power profiler that specifically measures the power consumption for only one tab should be used over the current method. This reduces random uncontrollable variables from interfering with the reading, which is a problem prevalent in the current testing method.

7.0 References

[1] Uzair, A., Beg, M. O., Mujtaba, H., & Majeed, H. *Weec: Web energy efficient computing: A machine learning approach*. Sustainable Computing: Informatics and Systems, 22, 230-243.

[2] Chaqfeh, M., Asim, R., AlShebli, B., Zaffar, M. F., Rahwan, T., & Zaki, Y. Towards a World Wide Web without digital inequality. Proceedings of the National Academy of Sciences, 120(3), e2212649120.

[3] SPQR, (2021). "Image resolution and human perception". Crafting Pixels. https://pixelcraft.photo.blog/2021/03/16/image-resolution-and-human-perception/ (accessed 3rd Dec, 2023)

[4] Turing, A. M. *Mind*. Mind, 59(236), 433-460.

[5] Das, S., Dey, A., Pal, A., & Roy, N. *Applications of artificial intelligence in machine learning: review and prospect.* International Journal of Computer Applications, 115(9)

[6] Mahesh, Batta. *Machine Learning Algorithms -A Review.* 10.21275/ART20203995.

[7] Maulud, D., & Abdulazeez, A. M. *A review on linear regression comprehensive in machine learning*. Journal of Applied Science and Technology Trends, 1(4), 140-147.

[8] Gechev, M., Zats, D., Li, N., Yu, P., Ramesh, A., & Gupta, S. *Speed-up your sites with web-page prefetching using Machine Learning*. TensorFlow Blog.

[9] Shyamala, K., Kalaivani, S., & Murugan, A *A Framework to improve the Web Performance using Reorganization, Optimized Prediction and Prefetching.* International Journal of Advanced Technology and Engineering Exploration, 8(3), 2277-3878.

[10] Jiang, S., Qin, S., Pulsipher, J. L., & Zavala, V. M. Convolutional Neural Networks: Basic Concepts and Applications in Manufacturing. arXiv preprint arXiv:2210.07848.

[11] MathWorks. *What is a Convolutional Neural Network?* https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html#:~:text=A%20convolutional%20neural%20network%20(CNN,%2Dseries%2C%20and%20signal%20data.

[12] Marhon, S. A., Cameron, C. J., & Kremer, S. C. *Recurrent neural networks*. Handbook on neural information processing, 29-65.

[13] Sysel, M., & Doležal, O. *An educational http proxy server*. Procedia Engineering, 69, 128-132.

[14] Qiong, G., & Li, W. *An optimization method of javascript redundant code elimination based on hybrid analysis technique*. In 2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP) (pp. 300-305). IEEE.

[15] Roelofs, R., Shankar, V., Recht, B., Fridovich-Keil, S., Hardt, M., Miller, J., & Schmidt, L. A meta-analysis of overfitting in machine learning. *Advances in Neural Information Processing Systems, 32*.

[16] Minsky, M., & Papert, S. A. *Perceptrons, reissue of the 1988 expanded edition with a new foreword by Léon Bottou: an introduction to computational geometry*. MIT press.

[17] Bebis, G., & Georgiopoulos, M. Feed-forward neural networks. IEEE Potentials, 13(4), 27–31. doi:10.1109/45.329294

[18] Cardot, H. (Ed.). *Recurrent neural networks for temporal data processing*. BoD–Books on Demand.

[19] AUDA, G., & KAMEL, M. MODULAR NEURAL NETWORKS: A SURVEY. International Journal of Neural Systems, 09(02), 129–151. doi:10.1142/s0129065799000125

[20] fchollet, "The Functional API". TensorFlow. https://www.tensorflow.org/guide/keras/functional_api (Accessed Nov. 30, 2023)

[21] Rumelhart, D. E., Durbin, R., Golden, R., & Chauvin, Y. Backpropagation: The basic theory. In *Backpropagation* (pp. 1-34). Psychology Press.

[22] Clevert, D. A., Unterthiner, T., & Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

[23] He, J., Li, L., Xu, J., & Zheng, C. ReLU deep neural networks and linear finite elements. arXiv preprint arXiv:1807.03973.

[24] Rasamoelina, A. D., Adjailia, F., & Sincak, P. A Review of Activation Function for Artificial Neural Network. 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI). doi:10.1109/sami48414.2020.9108717

[25] Basodi, S., Ji, C., Zhang, H., & Pan, Y. Gradient amplification: An efficient way to train deep neural networks. Big Data Mining and Analytics, 3(3), 196–207. doi:10.26599/bdma.2020.9020004

[26] Nworu, C. C., Ekpenyong, J. E., Chisimkwuo, J., Okwara, G., Agwu, O. J., & Onyeukwu, N. C. The Effects of Modified ReLU Activation Functions in Image Classification. *J Biomed Eng Med Dev*, *7*, 237.
[27] Han, J., & Moraga, C. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International workshop on artificial neural networks* (pp. 195-201). Berlin, Heidelberg: Springer Berlin Heidelberg

[28] Pratiwi, H., Windarto, A. P., Susliansyah, S., Aria, R. R., Susilowati, S., Rahayu, L. K., ... & Rahadjeng, I. R. Sigmoid activation function in selecting the best model of artificial neural networks. In Journal of Physics: Conference Series (Vol. 1471, No. 1, p. 012010). IOP Publishing.

[29] Hao Zheng, Zhanlei Yang, Wenju Liu, Jizhong Liang, & Yanpeng Li. Improving deep neural networks using softplus units. 2015 International Joint Conference on Neural Networks (IJCNN). doi:10.1109/ijcnn.2015.7280459

[30] Courbariaux, M., Bengio, Y., & David, J. P. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems, 28*.

[31] Touretzky, D. S., & Pomerleau, D. A. What's hidden in the hidden layers. *Byte, 14*(8), 227-233.

[32] Misra, D. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*.

[33] Bishop, C. M., & Nasrabadi, N. M. *Pattern recognition and machine learning* (Vol. 4, No. 4, p. 738). New York: springer.

[34] Hao Zheng, Zhanlei Yang, Wenju Liu, Jizhong Liang, & Yanpeng Li. Improving deep neural networks using softplus units. 2015 International Joint Conference on Neural Networks (IJCNN). doi:10.1109/ijcnn.2015.7280459

[35] Lin, G., & Shen, W. Research on convolutional neural network based on improved Relu piecewise activation function. Procedia Computer Science, 131, 977–984. doi:10.1016/j.procs.2018.04.239

[36] "Webpack Concepts", Webpack. https://webpack.js.org/concepts/ (accessed Nov 30, 2023)

[37] L. Maldonado, "All you need to know about CORS & CORS errors", Telerik. https://www.telerik.com/blogs/all-you-need-to-know-cors-errors#:~:text=CORS%20errors%20happen%20when%20a,by%20the%20server's%20CORS%20configuration. (accessed Nov 30, 2023)

[38] "Friendly Machine Learning for the Web". ML5js. https://ml5js.org/ (accessed 30th Nov, 2023)

[39] B. Jackson, "JPG vs JPEG: Understanding the Most Common Image File Format". Kinsta. https://kinsta.com/blog/jpg-vs-jpeg/#:~:text=There%20is%20no%20difference%20between,it%20came%20to%20file%20names (accessed 30th Nov, 2023)

[40] Lucci, S., Kopec, D., & Musa, S. M. *Artificial intelligence in the 21st century*. Mercury learning and information.

[41] Bickel, P. J., & Doksum, K. A. Mathematical statistics: basic ideas and selected topics, volumes I-II package. CRC Press.

[42] "Accuracy and Loss". Ai Wiki. https://machine-learning.paperspace.com/wiki/accuracy-and-loss. (accessed 3rd Dec, 2023)

[43] Zhang, X.-S. Neural Networks in Optimization. Nonconvex Optimization and Its Applications. doi:10.1007/978-1-4757-3167-5

[44] fchollet, "The Sequential Model". TensorFlow Core. https://www.tensorflow.org/guide/keras/sequential_model (accessed 30th Nov, 2023)

[45] Janocha, K., & Czarnecki, W. M. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*

[46] Sharma, S., Sharma, S., & Athaiya, A. Activation functions in neural networks. *Towards Data Sci*, *6*(12), 310-316.

[47] Fehm, H. L., Kern, W., & Peters, A. The selfish brain: competition for energy resources. *Progress in brain research*, *153*, 129-140.

[48] Héam, P. C., & Nicaud, C. Seed: An Easy-to-Use Random Generator of Recursive Data Structures for Testing. 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. doi:10.1109/icst.2011.31