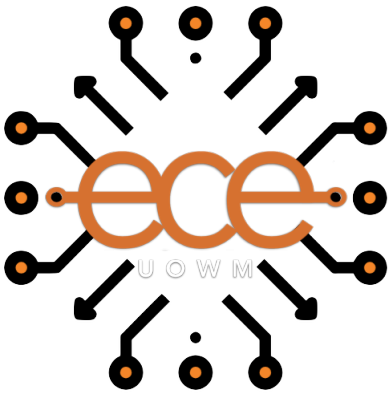


Πανεπιστήμιο Δυτικής Μακεδονίας

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

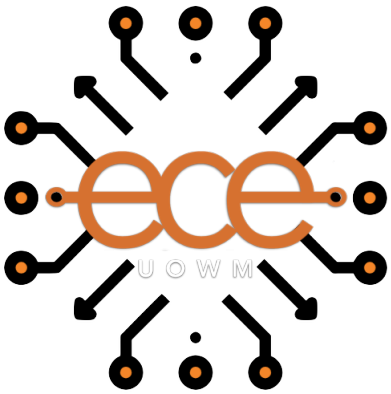
Μελέτη και σύγκριση συναρτήσεων ενεργοποίησης σε νευρωνικά δίκτυα

Ζαχαρής Γεώργιος (ΑΜ: 1323)

Επιβλέπων Καθηγητής: Νικόλαος Πλόσκας

Εργαστήριο Ευφυών Συστημάτων & Βελτιστοποίησης

February 28, 2024



University of Western Macedonia

Department of Electrical & Computer Engineering

Study and comparison of activation functions for neural networks

Georgios Zacharis (AM: 1323)

Supervisor: Nikolaos Ploskas

Intelligent Systems & Optimization Laboratory

February 28, 2024

Δήλωση Πνευματικών Δικαιωμάτων

Δήλωση Πνευματικών Δικαιωμάτων Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο ”Μελέτη και σύγκριση συναρτήσεων ενεργοποίησης σε νευρωνικά δίκτυα” καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Νικόλαου Πλόσκα αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Acknowledgement

Throughout the years in which I have completed the span of my degree studies, I have faced many obstacles and challenges, experienced joyful, insightful, and unexpected moments, and evolved not only in my field of study but also mentally. As this pluridimensional, colorful, and insightful journey comes to an end I stand optimistic in front of an unexplored future with a great desire to reach many more milestones and evolve even further. In this long journey, I could not but appreciate, cherish, and respect all the people who we of great company and support in both the darkest and brightest moments. I find myself thankful to the Institution, the University of Western Macedonia, and its faculty that guided me on this course. As such, I would first like to thank my supervisor, Associate Professor Nikolaos Ploskas, for his continuous support and insight he offered for this work and beyond. Secondly, I would like to thank PhD Candidate Christos Krallis for his insights and comments in this work. Finally, I would like to express my deepest gratitude to my mother and brother and all my friends.

Abstract

The flourishing of Artificial Intelligence (AI) in recent years, mainly due to the technological advancements resulting in high-performing hardware that boosted the field's rise, has led to the research and development of many real-world applications. By extension, since the field's theoretical beginnings are now applied to solve real-world problems, Machine Learning (ML), a subcategory of AI has been proven highly advantageous for computer vision tasks including image classification. This led to the development of various high-performing image classification Neural Networks (NN), each one with a different architectural approach. Through Transfer Learning (TL) these networks can be used for the development of real-world applications. However, such applications come with challenges that require an NN performing a task to be highly efficient, accurate, fast, stable generalized, and as less computational power-consuming as possible. There is constant research to improve models by designing innovative architectures through various tools and techniques, including activation functions. This work focuses on improving popular pre-trained image classification NNs of high architecture and performance by altering the activation functions they use in their core. The models are trained for five datasets, each time with a different activation function in their entirety of architecture. Nine activation functions were chosen for testing. The experiments show optimistic results as improvements in performance in terms of accuracy or training time are possible and in many cases to a high extent.

Keywords: Machine Learning, Neural Networks, Activation Functions, Deep Learning, TensorFlow, Keras

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A short story on image classification

There can be no doubt that image classification is one of the most researched topics of study in the field of computer vision. It is the process of classifying images into different classes based on their content. As a subject computer vision and by extension image classification are not state-of-the-art concepts as they have already been successfully tested and applied practically since the last decade.

However, radical advancements in the field of Artificial Intelligence (AI) and, by extension Machine Learning (ML), in recent years have significantly boosted every field of computer science including computer vision and image classification. Through ML and Deep Learning (DL), a wide variety of Convolutional Neural Networks (CNN) were developed for image classification tasks, each one with a different architecture and approach, in a constant attempt to develop image classification models as efficient, fast and accurate as possible so as to be implemented to applications for real-world solutions. Through these advancements, there have been developed many tools for machine learning and image classification such as different Neural Network (NN) architectures, various activation functions, various tools for development, etc.

Despite the flourishing of all the fields mentioned above, there is a constant need for improvement and better and faster performances in order to tackle real-world problems that require high and dynamic performance. For this reason, there is constant research to test different combinations of tools, architectures, and tunings for image classification models [8].

1.2 Motivation

Activation functions, which will be analyzed in detail in this work, play a big role in the function and performance of image classification models. In an attempt to improve established image classification models through experimenting with different activation functions inside the layers of the models, this work aims to provide not only a comprehensive analysis of the theoretical background of DL for image classification tasks but also optimistic results for further research. In this work, various established image classification networks are tested on their performance on various datasets, having their original activation functions altered to find out if there are any improvements. There are nine different activation functions tested in each network.

1.3 Thesis structure

This work consists of five chapters. Chapter 2 provides a detailed overview of the theoretical background of this work. In Chapter 3, there are discussed and analyzed the components of the experiments of this work and the tools and frameworks used. In Chapter 4 the results of the experiments are presented and discussed in detail as well as the improvements that were observed. Finally, Chapter 5 concludes this work and presents possible future extensions.

Chapter 2

Theoretical Background

2.1 Artificial Intelligence

Artificial intelligence is a field of computer science that focuses on the development of intelligent machines through studying data, planning, and learning in order to make decisions based on logic. Throughout the long history of the field, there have been given many definitions of AI thus there is no universal one. John McCarthy in 2004 stated that AI is "*the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable*".

2.2 Machine Learning and categories

Machine Learning is a subfield of AI that attempts for machines to learn automatically meaningful patterns and relationships from examples and observations so that they can give predictions based on the data it was trained. Since its theoretical beginnings, the field has known radical advancements over recent years leading to its vast expansion of subfields and applications [9]. The field of ML and its applications are divided into four primary categories, as illustrated in Figure 2.1.

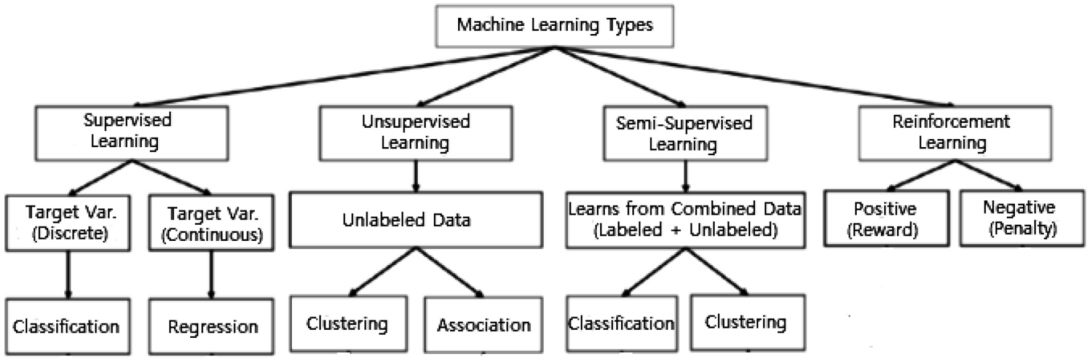


Figure 2.1: The categories of Machine Learning [1]

- **Supervised Learning**: Supervised Learning is the process of learning the way to correlate a possible input to an output based on a previously seen sample of labeled data. It is considered the most common ML category and is usually applied to practical tasks such as data classification (including image classification) and regression.
- **Semi-supervised Learning**: It is considered a hybrid of Supervised and Unsupervised Learning as it operates on both labeled and unlabeled data. There may be many possible outputs.
- **Unsupervised Learning**: Unsupervised Learning attempts to recognize and group unlabeled input data without the need for human supervision. It is widely used for feature extraction, identification of unknown patterns, and exploratory purposes. The most common unsupervised learning tasks are clustering, dimensionality reduction, feature learning, etc.
- **Reinforcement Learning**: It enables a machine to learn and improve itself through penalties or rewards so that it can evaluate the optimal process to carry out a task. In other words, the machine learns automatically by itself through experience. Reinforcement Learning is a powerful tool for increasing efficiency in tasks such as robotics, autonomous driving, and logistics.

2.3 Deep Learning

In the early years of ML and image classification, the traditional ML methods required a lot of human supervision and manual work so that the image classification

networks function up to a sufficient extent but by no means satisfactory. With the great advances in the field of ML, Deep Learning boosted the computer vision field. DL is a subset of machine learning that aims to teach machines to process, train, and understand patterns and make predictions mimicking the human brain. Deep learning essentially is a neural network of three or more layers in its architecture. It can be implemented in cases of both supervised and unsupervised learning. In the field of image classification, contrary to the traditional ML methods, it only requires raw data so that it can be analyzed by the network itself. Deep Learning gave a great boost and enabled many advancements in the field of image classification and nowadays it is almost entirely the way image classification networks are developed [10]. This is the field on which this thesis is focused on. Deep Learning networks consist of the following components:

- **Input Layer**: several nodes in the beginning of the architecture of the network that input the data into it.
- **Hidden Layer**: The hidden layer of an NN may consist of many layers. This is where the data is processed through different levels so the network can analyze it and learn patterns in order to give an output. There are many different types of layers in the hidden layer, which are explained further below.
- **Output Layer**: This layer consists of the nodes that output the data. This output may be True/False answer, the class of an image (e.g. cat/dog), etc.

2.4 Artificial Neural Networks and components

Artificial Neural Networks or NNs are mathematical-based artificial adaptive systems that, by mimicking the human brain and the way neurons work, are used for machine learning tasks. The way they work is that they consist of interconnected layers of neurons, each one containing given weights. Each layer modifies the layer and passes them to the other layers. The weights are essentially the knowledge the model has at that particular phase of the training the values of which are produced through the mathematical processes of each layer. The network contains a set of hyperparameters that basically define the way the network needs to function and play a crucial role

in its performance [11]. Examples of the hyperparameters a neural network contains are:

- **Learning Rate**: the rate at which the network updates estimates.
- **Epochs**: the number of times the network will process the entire datasets given as input.
- **Batch Size**: the size of the data each time the network processes in each epoch.

2.4.1 Activation functions

Activation functions are mathematical functions that add non-linearity to the network and affect the output of a neuron. They play a crucial role in the functionality and performance of the network. Depending on the input the function is given, it decides which neuron to activate or not so that the weights are updated with the new knowledge the model has gained at that particular stage of training. Depending on its architecture and task, a neural network usually uses many different activation functions in its layers.

Since their contribution to the network is of high significance it is only natural that there have been explored, improved, and developed various activation functions. It is worth mentioning that through years of testing and research, it seems that there is not a one-fits-all solution for sure, but it has been observed that many functions perform better for specific tasks than others. Choosing the right activation function requires testing to see its performance in a network since it does not only need to add non-linearity but also to not hamper the gradient flow during training [12]. For the experiments of this work there were gathered 9 activation functions popular for image classification tasks both traditional and state-of-the-art:

- **ReLU**: The most popular activation function is the Rectified Linear Unit or ReLU activation function. It is a simple calculation that returns the value provided as input directly or 0 if the value is 0 or less. It is mathematically represented as:

$$F(x) = max(0, x)$$

- **Sigmoid**: The logistic activation function, commonly known as Sigmoid, is a non-linear activation function and takes any real value as input and outputs values within the range of 0 to 1. The higher the input value is the closer to 1.0 the output value will be. Because of this, it is commonly used for prediction models, since a probability value exists between the range of 0 and 1. However, it is vulnerable to the gradient decent problem, because as the input values recede from 0 the gradient value approaches zero, making a model unstable [13]. It is mathematically represented as:

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

- **Tanh**: Similar to the Sigmoid function, the hyperbolic tangent activation function, commonly known as Tanh, also has an S-shape form. It differs in the output value range as its output values range from -1 to 1, and the larger the input, the closer the output to 1.0. It is usually used in the hidden layers of the models as it helps centering the data which makes learning for the next layers easier. However, it also faces the vanishing gradients problem [14]. It is mathematically represented as:

$$F(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- **ELU**: The Exponential Linear Unit (ELU) is one of the many variants of the ReLU activation function. It performs better than ReLU as it avoids the dead ReLU problem but increases the computational time of the model because of the exponential operation included [12]. It is mathematically represented as:

$$f(x) = \begin{cases} \alpha(e^{-x}) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

  where α is a constant value initialized to 1

- **GELU**: The Gaussian Error Linear Unit (GELU) activation function is a smooth approximation of the ReLU function that is mainly used for NLP models al-

though it is also used for image classification [15]. It uses the cumulative distribution function (CDF) of the standard normal distribution, denoted Φ(x). Thus it provides much better non-linearity than ReLU and ELU in theory. It is mathematically represented as:

$$f(x) = xP(X \leq x) = x\Phi(x) =$$

$$= 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

- **SELU**: The Scaled Exponential Linear Unit (SELU), also a variant of the ReLu function, provides the model with internal normalization. This means that each one of the layers preserves the mean and variance from the previous layers. This makes the network converge faster and reduces training time. [16]. It is mathematically represented as:

$$f(x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

where $\lambda \approx 1.0507009$ and $\alpha \approx 1.6732632$

- **Swish**: The Swish activation function is a state-of-the-art highly performing activation function especially used when dealing with large datasets. It is a smooth, non-monotonic function and avoids significantly the problems of other activation functions like the gradient decent problem. Its development focuses on improving models for image classification tasks [17]. It is mathematically represented as:

$$f(x) = x * sigmoid(x)$$

- **Mish**: Motivated by the work performed by Swish, Mish is a self-regularized non-monotonic activation function that shares the beneficial properties of Swish and GELU. Mish outperforms the latter but also avoids the dying ReLU phenomenon due to the better optimization and stability it provides to the model

[18]. It is mathematically represented as:

$$f(x) = x * \tanh(\ln(1 + e^x))$$

- **SMish**: Smish is a state-of-the-art activation function that offers the model good gradient-based optimization improving its performance. Basically, it computes the input using the sigmoid and tanh functions mentioned earlier. Although it is costly in terms of computational resources, making its use less practical for large-scale applications, the fact that it outperforms in theory many activation functions has led many researchers to explore its potential improvements [19]. It is mathematically represented as [18]:

$$f(x) = x * tanh(ln(1 + \frac{1}{1 + e^{-x}}))$$

2.4.2 Vanishing and exploding gradients

In many cases, the gradients, which are the derivatives of a function that has more than one outputs, get more and more low and approach zero which eventually leads to the weights of the initial or lower layers being nearly unchanged or unchanged at all. This is known as the vanishing gradients or the gradient decent problem. In some other cases, the gradients keep on getting larger and larger causing very large weight updates and thus the gradient descent to diverge. This is commonly known as the exploding gradients problem. These two problems are the ones that a network needs to avoid in order to function or else there will be training issues [20].

2.4.3 Convolutional layers

Convolutional layers are used for detecting features within an input, commonly an image. As the name suggests, they use they perform a convolution to an input and then pass the result to the next layer. In this procedure, they use filters also known as kernels. The most common type of convolutional layer is the 2D convolutional layer, in which the kernel processes the 2D array of the pixels of an image, sums up the multiplication of the elements and then transforms it to a single output pixel [21].

2.5 Literature review on comparison of activation functions

It is worth noting that other researchers have attempted to carry out experiments of a similar nature by testing different combinations of datasets, activation functions, and image classification NNs. Dubey et al. [12] in 2021 made a comprehensive survey of different activation functions and their effect on image classification NNs.The experiments were mainly on the ReLU variations and showed optimistic results for the Mish, ELU, and GELU functions amongst others used with MobileNet, VGG-16, GoogleNet, ResNet50, SENet18, and DesneNEt121. The tests were conducted on the CIFAR-10 and CIFAR-100 datasets.

In 2023, in a work published by Verma et al. [22] various activation functions were tested in different NNs with different datasets to observe their impact and highlight the significance of choosing the right activation function. The experiments showed that the SwAT activation function outperformed the others tested. The experiments were conducted on ReLu, ELU, Swish, SELU, LReLU, Tanh, DSiLU, TSiLU, ATSiLU, and SwAT functions, used with the VGG-19, ResNet50, DenseNEt121, and InceptionV3 on the CIFAR-10, CIFAR-100, and MNIST datasets.

In 2021, Zhang et al. [23] made a similar study. The Swish, GELU, ReLU, Sigmoid, and SELU activation functions were tested on different types of NNs to test their performance. The tests were carried out on the MNIST and Fashion MNIST datasets.

Chapter 3

Implementation

This section provides detailed coverage of each step of the implementation process. To begin with, there will be complete coverage of the tools, libraries, and frameworks used for conducting the experiments. Later, there will be discussed in detail the pre-trained models, the datasets, and the activation functions chosen for the experiments of this work. Finally, there will be a detailed analysis of the code and the conducting of the experiments.

3.1 Tools, libraries, and frameworks

In order for these experiments to be implemented efficiently but also with high accuracy so that they can provide a clear and comprehensive view of the results, several environments, and frameworks are used, the nature and use of which are explained in detail below. By understanding the tools used, we can have a clearer understanding of the process and stages of the implementation and the way the experiment was carried out. There is a plethora of good tools and frameworks for ML applications. The tools described in this section are the ones chosen that fit the needs of this work and are highly recommended for image classification.

Figure 3.1: Python logo

- **Python**: Python is one of the oldest and most popular programming languages that are used to date and was created by Guido van Rossum and was first released in 1991. Due to its easy-to-learn and user-friendly syntax and structure, it became radically popular amongst new and experienced programmers, which led to the development of many tools, libraries, and applications of various natures due to the ever-growing and diverse community of programmers. However, its high popularity is not only due to the simplicity of its syntax relative to other programming languages but also due to its efficient approach to object-oriented programming, its fast and powerful capabilities, and its compatibility with many platforms, APIs, and other programming languages. Needless to say, Python is constantly updated, improved, and implemented in various tasks. In recent years it has become a go-to programming language for machine learning programming, especially for beginners, but also provides many powerful tools to accommodate its experienced users [24][25].

Figure 3.2: TensorFlow logo

- **TensorFlow**: In 2017 Google released the initial version of TensorFlow, a free, open-source software library aiming to improve, but also make more accessible and user-friendly the development of data analysis, machine learning, and artificial intelligence applications at large scale. TensorFlow has a particular focus on the training and inference of deep neural networks. It provides an end-to-end platform as well as tools to consolidate, clean, and preprocess data at scale. Furthermore, it provides datasets for training machine learning models

for many tasks like natural language processing (NLP), image classification, audio processing, etc with TensorFlow Datasets (TFDS) as mentioned later. It also supports GPU acceleration for heavy computations across clusters with CUDA support, which is an important feature, especially for ML tasks. At the time of writing this thesis, TensorFlow is one of the most popular machine-learning frameworks due to its wide range of capabilities, ease of use, strong community support, and compatibility with many programming languages such as JavaScript, C++, Java, and, in this particular case, Python. Evidently, it is more than suitable to be of assistance in the implementation of this thesis. However, depending on the nature of the research and its needs, there may be other frameworks that match a particular case or application like PyTorch and scikit-learn [26].

- **TensorFlowDatasets (TFDS)**: TensorFlow Datasets (TFDS) is a library of the TensorFlow platform that provides a collection of ready-to-use datasets for ML and handles downloading and preparing the data deterministically. This is highly useful as the user only needs to install this library and load a dataset without using third-party software or downloading and uploading a dataset for training manually as it is barely efficient and highly time-consuming. For the needs of the experiments of this work, the CIFAR-10, CIFAR-100, SVHN, STL-10, and Food 101 datasets are used which are analyzed later [27].

Figure 3.3: Keras logo

- **Keras**: Keras is a free, open-source library that was designed to enable fast experimentation with artificial neural networks and hence deep learning models. It contains numerous implementations of commonly used neural network building blocks such as dense layers, convolutional layers, dropout layers, pooling layers, optimizers, and activation functions. Moreover, it provides users with the most popular and highest-performing pre-trained networks, such as the ones

used in this thesis, for experimentation and development of machine learning applications. Since the release of its 2.4 version, Keras has acted as a high-level API exclusively for the TensorFlow platform and the two of them are highly compatible together and provide an approachable, highly productive interface for solving machine learning problems. Furthermore, it provides many popular pre-trained deep-learning networks for the user to easily utilize such as Xception, InceptionV3, ResNet, DenseNet, EfficientNet, and MobileNEt that are used in this work. The models and their architecture are further analyzed later in this thesis [28][29].

Figure 3.4: Anaconda logo

- **Anaconda**: The development of machine learning applications can be tricky, incomprehensible, and time-consuming for beginners both in theory and in practice. The tools mentioned prior are no exception and the common factor is that they attempt to make machine learning programming not only better but also more and more user-friendly and accessible. Hence, Anaconda is no exception as well. Like other environments, frameworks, and libraries, TensorFlow and Keras can be tricky to install and set up manually, especially in Linux environments as they require the user to have more than a novice hands-on experience, or else they may encounter many obstacles and functionality issues.

  Anaconda is a free, research-focused integrated development environment (IDE) that provides the user with many tools that ease the development of code, especially for ML and data analysis tasks in either Python or R. It is a simple, well-supported graphical user interface (GUI) that includes the most popular and important software, libraries and IDEs in its installation and is a popular IDE, especially for ML due to the tools provided for such purposes like Jupiter Notebook, Jupiter Lab, Spyder etc. Furthermore, it spares the user of the diffi-

cult manual installation process of each tool, which is an important feature as all software and libraries have specific dependencies in terms of compatibility and software versions that must be met in order to be functional and efficient. Likewise, for the needs of this thesis, Anaconda provides an easy way to install TensorFlow, TFDS, and Keras[30].

Figure 3.5: Weights and Biases logo

- **Weights and Biases (Wandb)**: Weights and Biases (Wandb) is a free API for data visualization providing the user with a wide range of tools to graphically represent performance, accuracy, power consumption, and much other useful information for research purposes. It is a useful tool as it is not only a user-friendly and easy-to-use GUI but also it provides high-quality data visualization which is vital not only for testing and improving the performance of a model but also for having a clear and comprehensive depiction of it for presentation or research purposes. Furthermore, an also useful feature is that it provides real-time monitoring of a model's performance as well as a simple way of configuring the hyperparameters of the training process. The way it works is that after installation the user only needs to create a repository in Wandb and then include a line of code in his code that specifies the repository where the data is sent for visualization and monitoring and the hyperparameters for training [31].

3.2 Pretrained models

In practice, in very few particular cases there is a need to develop and train an entire convolutional neural network from scratch as it is a complicated, time-consuming, and costly process. Furthermore, to train a highly efficient and generalized model there is a need for an extreme amount of data and computational resources that in most cases is simply unachievable. Entities of research who can afford the implementation, but also the research for the development of highly efficient, accurate, generalized models have introduced many pre-trained models over the years. Each model provides a different architecture and approach for specific tasks like NLP, Image classification, audio processing, and object tracking. In addition, the weights of its training can be loaded so they can be used for implementation on unseen data.

In the case of this thesis, over the years there have been developed a plethora of pre-trained models that have been established for image classification tasks. These networks have been trained on the largest dataset, the ImageNet dataset, which contains 1,000 object classes and contains 1,281,167 training images, 50,000 validation images, and 100,000 test images. ImageNet is the established benchmark for large-scale image classification models. Each model requires inputs of specific size and scale as explained in the Implementation chapter of this work [32][33]. In order to test the effectiveness of changing the activation functions of a model, the following popular image classification pre-trained models are used:

- **Xception**: Xception is a convolutional neural network, and it is built on the concept of depthwise separable convolutional layers. This means that instead of using convolutional filters that alter the depth of an image, the filter is applied to each channel separately, which significantly improves the efficiency and effectiveness of the model.Its architecture consists of three types of layers, each one having a specific role in its functionality. To begin with, there is the entry flow, where the image given as input is resized and preprocessed in preparation for the main architecture. The entry flow contains different layers such as convolutional, batch normalization, activation, and pooling layers all of which are involved in processing the input image.The second part of the Xception architecture is the middle flow, which also contains multiple convolutional layers,

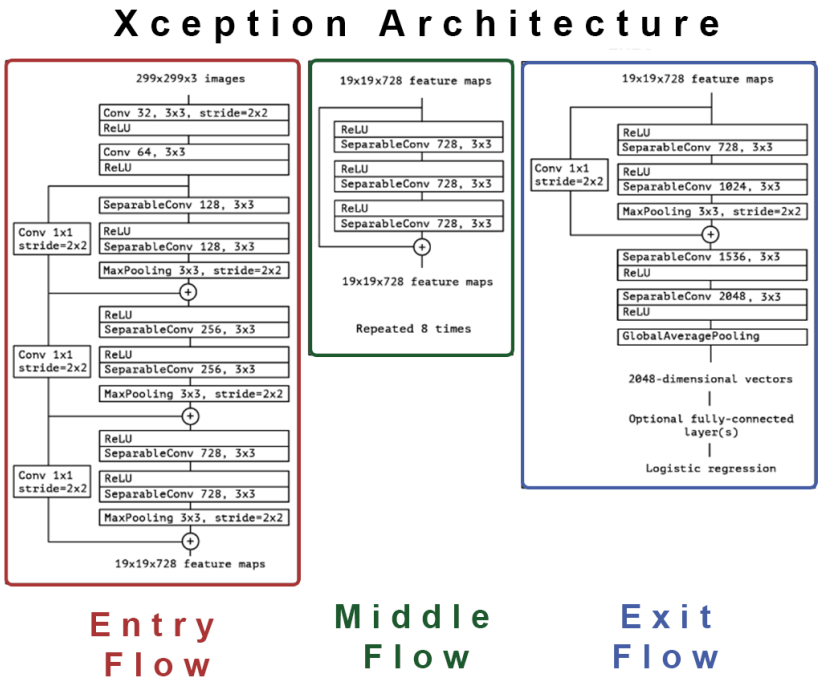


Figure 3.6: Xception architecture, *Xception: deep learning with depthwise separable convolutions* [2]

batch normalization, and activation functions. It is designed to maintain the resolution of the image in order to prevent loss of information. Finally, the exit flow reduces the image resolution and channels through a set of convolution layers and pooling. By this process, the network dimensions are reduced which is crucial in the reduction of the computational requirements and memory footprint of the network, making it highly efficient and effective [34]. Figure 3.6 illustrates the architecture of the Xception Network.

- **InceptionV3**: The Inception Network got its name due to its Inception module, an innovative architecture that was introduced in the first version of this model. The motivation for its development was to increase the capability of a deep neural network but also with efficient use of computational resources.

  The Inception module uses multiple filters of different sizes on the same level thus making the network wider but not deeper, so that overfitting of the data is avoided. The inception module consists of four parallel layers: 1x1,3x3 and 5x5 convolution layers and one 3x3 max pooling layer. At the end of the Inception module, the output of all these kernels is stacked together. Although the first version of the Inception Network had a high performance, modifications

Inception Architecture

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure	35×35×288
5×Inception	As in figure	17×17×768
2×Inception	As in figure	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

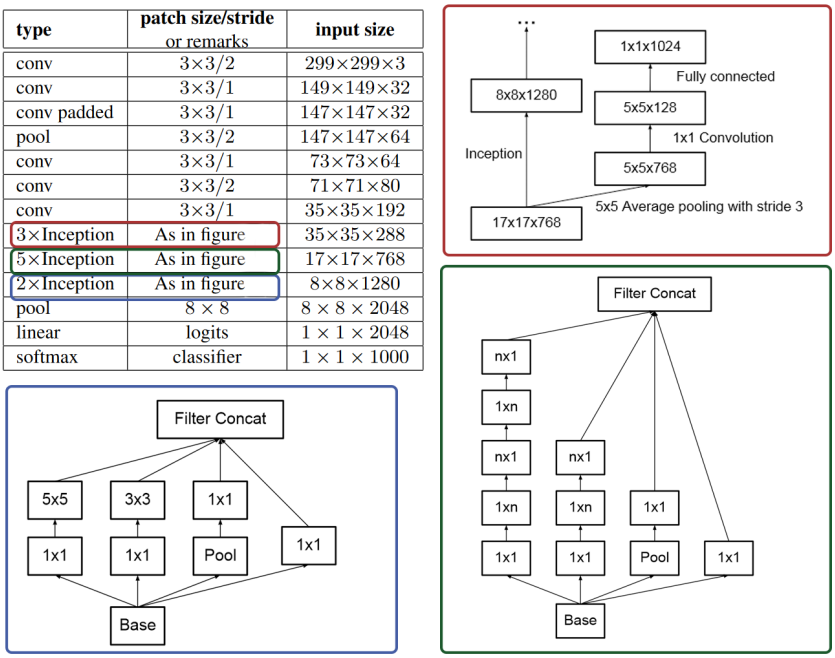


Figure 3.7: Inception architecture and blocks [3]

needed to be made in its next versions so that it could be more power efficient. Such modification resulted in the architecture of InceptionV3, by factorizing the convolutions of the model into smaller convolutions, adding auxiliary classifiers to combat the vanishing gradient problem, and adding max and average pooling layers in order to reduce the grid size of the feature maps. The Inception V3 model consists of 42 layers and is illustrated in Figure 3.7 [3].

- **ResNet**: The Residual Network (ResNet) is a convolutional neural network designed to support hundreds or thousands of convolutional layers. As explained before, the more the layers in a convolutional neural network the more the chances for the vanishing gradient problem to appear.However, the model uses the many convolutional layers in its favor. In order to avoid the vanishing gradient problem, the Residual block (ResBlock) is introduced, otherwise known as "skip connections". In this block, multiple convolutional layers that do nothing at first (called identity mappings) are stacked, then skipped, and then the activations of the layer before are reused. This process speeds up the initial training by compressing the network into fewer layers. An example of the residual block as well as the architecture of ResNet are illustrated in Figure

ResNet Architecture

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

The ResBlock

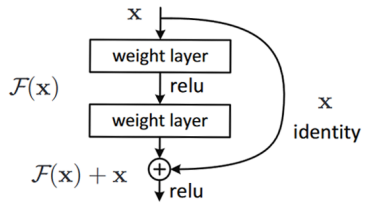


Figure 3.8: ResNet architecture and the ResBlock [4]

3.8. Since its innovative release, there have been many variations of the ResNet model, including DenseNet, which is discussed later [4]

- **DenseNet**: As previously mentioned, the DenseNet model is a popular variation of the ResNet model. It attempts to resolve the vanishing gradient problem by creating more connections and ensuring the maximum flow of information between the network layers by connecting each layer directly to all the others. This implementation is parameter efficient and allows the network to reuse features, with every layer behaving as a separate state, reading from one preceding state and writing to one subsequent layer. This feed-forward capability reduces feature redundancy and provides an efficient information flow. However, the model distinguishes the difference between preserved and newly added information. With a final classifier layer, all the network's feature maps are used to make decisions [5]. It is depicted in further detail in Figure 3.9.

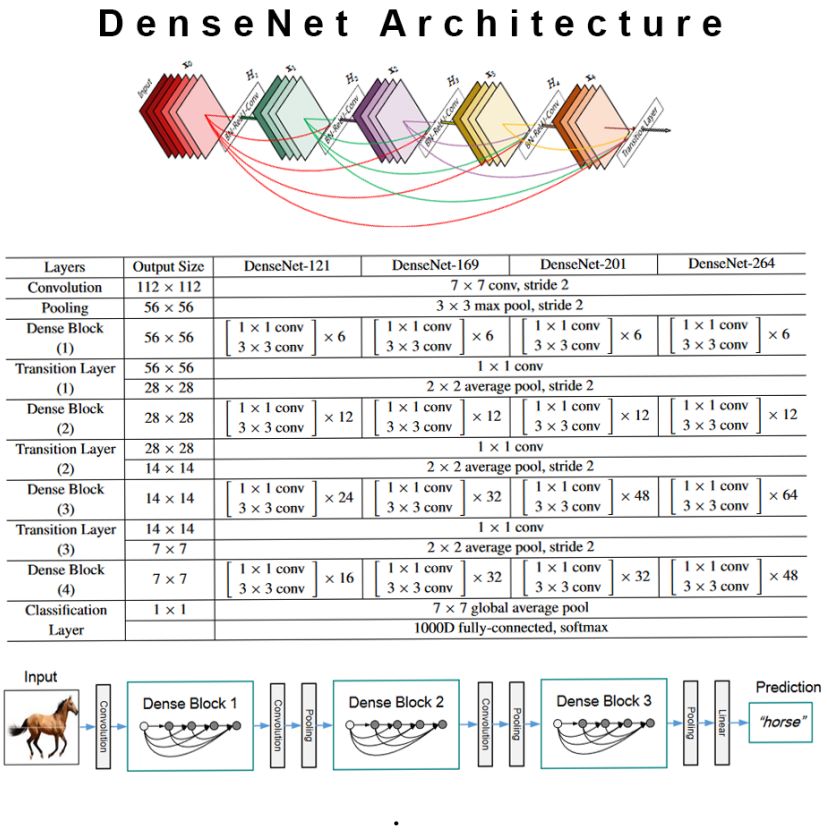


Figure 3.9: DenseNet architecture [5]

- **EfficientNet**: EfficientNet is a convolutional neural network that is built upon a concept called compound scaling. Compound scaling attempts to bring a balance to a model in terms of size, accuracy, and reduced computational requirements. To achieve this, it uses Mobile Inverted Bottleneck (MBConv) layers, which are a combination of depth-wise separable convolutions and inverted residual blocks. These layers are fundamental to the model's architecture.

  The MBConv layer begins with a depth-wise convolution and then is followed by a point-wise convolution expanding the number of channels, and finally, another convolution reducing the channels back to their original number. This bottleneck design allows the model to have a highly efficient learning process while still maintaining a high degree of representational power. Furthermore, in order for the model to filter the essential features a Squeeze-and-Excitation (SE) optimization block is used. This block uses global average pooling to reduce the spatial dimensions of the feature map to a single channel, followed by two fully connected layers, as seen in Figure 3.10 [35].

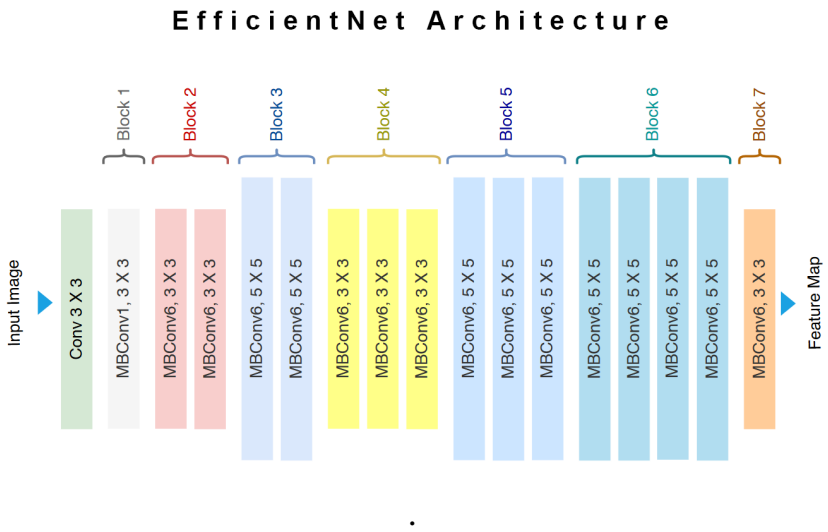
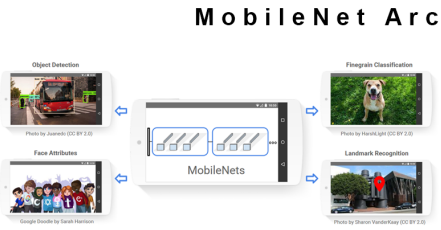


Figure 3.10: EfficientNet architecture [6]

- **MobileNet**: MobileNet is a lightweight but highly efficient convolutional neural network developed for use on devices with limited computational resources, such as smartphones. It uses simplified architectures in order to reduce the calculations and parameters. This is achieved using primarily depth-wise and point-wise convolution techniques. In addition, the model introduces two innovative global hyperparameters that enable the model's tuning to be flexible in trading off latency or accuracy for speed and lower computational requirements depending on the requirements of its use. The convolution is essentially divided into two layers where the first layer is used to filter the input channels while the second layer combines the generated result and creates a new feature. Then another additional layer computes the linear combination of the output from the depth-wise convolution. A visual representation of the MobileNet architecture is provided in Figure 3.11. Since its release, there have been many revisions to the architecture of the model. For this work, the MobileNetV2 model is used, which through some significant changes in its architecture achieves higher accuracy scores than its predecessor [36].



Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

.

Figure 3.11: MobileNet architecture [7]

3.3 Datasets

This section covers the datasets used in this work for the experiments. Figure 3.12 shows a preview of the images each dataset contains.

Figure 3.12: Datasets preview

- **CIFAR-10**: The CIFAR-10 dataset is one of the most famous datasets for machine learning image classification consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and

10,000 test images. The 10 classes of the dataset are the following: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset's size is 163 MB. It is divided into five training batches and one test batch, each one with 10,000 images. The test batch contains exactly 1,000 randomly selected images from each class. The training batches contain the remaining images in random order, although some training batches may not contain an equal number of images as others [37].

- **CIFAR-100**: This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. The images' dimensions and specs are the same as the CIFAR-10 dataset. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 super classes. Each image comes with a label indicating the class to which it belongs and a second label indicating the superclass to which it belongs. The dataset's size is 161 MB.

- **Street View House Numbers (SVHN)**: The SVHNdataset is an image dataset consisting of over 600,000 real-world images of digits and numbers in natural scenes. The images were obtained from house numbers in Google Street View images. The detection and classification of images in real-world conditions is a much harder task for models to carry out than images of high resolution. The dataset has 10 classes, one for each digit from 0 to 9. The size of the dataset is approximately 1.47GB [38].

- **STL-10**: Inspired by the CIFAR-10 dataset, the STL-10 dataset is also an image recognition dataset consisting of 10 classes, similar to the CIFAR-10 dataset, of 96x96 100.000 unlabeled images. Contrary to its predecessor, STL-10 contains much more images and fewer labeled training examples. The primary challenge is for the models to attempt to classify the unlabeled data, which increases the difficulty of the task the model has to carry out. The size of the dataset is 2.46 GB [39].

- **Food 101**: The Food-101 dataset is a challenging dataset consisting of 10,100 images of food and dishes images of 101 categories. Each class contains 250 manually reviewed images. The dataset images purposefully contain noise and

Input Specifications of Models		
Model	Input Shape	Input pixels
Xception	71,71,3 - 299,299,3	[-1,1]
InceptionV3	75,75,3 - 299,299,3	[-1,1]
DenseNet121	32,32,3 - 224,224,3	[0,1]
EfficientNetV2B0	32,32,3 - 224,224,3	[-1,1]
MobileNetV2	32,32,3 - 224,224,3	[-1,1]
ResNet50V2	32,32,3 - 224,224,3	[0,1]

Table 3.1: Input requirements of networks

some of them are wrongly labeled, so it can be challenging for a model to classify them. The size of the dataset is 5GB [40].

3.4 Setting up the experiments

Using Tensorflow, the process of loading a pre-trained model only requires a few lines of code as in the example given in the images below. The model is initialized without loading its pre-trained weights since the aim of this work is to see how well these models can be trained by changing the activation functions in its architecture. As previously mentioned, each pre-trained model accepts inputs with specifications as minimum and maximum size and input pixels. Table 3.1 specifies the requirements of the inputs of each pre-trained model. That is why the dataset after being loaded to the model is preprocessed using the function below before training. Finally, the models are trained for each dataset, each time with a different activation function in their entirety of architecture. It is important to note that the models are trained also with their original architecture for reference. This means that each model is trained 10 times for each dataset. The experiments of this work consist of 300 runs in total.

Chapter 4

Results

The results of the experiments were interesting and surely optimistic since there were many cases of improved performance. As discussed further in this section, there were many notable cases and thus many conclusions to extract. To begin with, it is necessary to address the performance of the models on each one of the datasets. As explained in the last section, for each dataset each pre-trained model is trained 10 times (no pre-trained weights included), each time with a different activation function in its entirety (original model included for reference).

Since many of the datasets are made from images from the ImageNet dataset and the pre-trained models have been trained to the ImageNet dataset it is natural that the general performance in most cases will be high. The main focus is to gather the tests and data to find out if any changes in the activation functions will result in better performance relative to the original model. Although this work's experiments can be labeled as small-scale image classification, any improvements, even if minor, might be of vital importance when it comes to large-scale tasks. Better performance can be in terms of accuracy, training time, or reduced power efficiency. Graphical depictions for accuracy and GPU usage as well as training time data are included in this section for a more comprehensive analysis.

Before analyzing the results of the experimentations, it is vital to note the activation functions that are used in the original architectures of the NNs.

- Xception uses primarily the ReLU activation function in its architecture.
- InceptionV3 and DenseNet121 primarily use the ReLU function as a backbone

but also use the ReLU6, the Sigmoid and LReLU functions for the internal needs of their architectures.

- EfficientNetV2 primarily uses the Swish activation function.
- MobileNetV2 primarily uses the ReLU6 activation function.
- ResNEt50V2 primarily uses the ReLU activation function.

4.1 Performance results for each dataset

4.1.1 Cifar-10 tests

The accuracy of the Xception model was improved significantly, in comparison to the original performance, with the Swish, Mish, GELU, and ELU activation functions, as seen in Figure and Table 4.1. The highest accuracy of 0.967 was achieved with Swish. The original model achieved a lower accuracy of 0.87 but had the fastest runtime with a difference of more than 2 minutes from the top-performing activation functions.

The InceptionV3 model saw also an improvement in both accuracy and training time using the Sigmoid function, outperforming the original model. Figure and Table 4.2 depicts in detail the performance of the model. Using sigmoid, InceptionV3 had a 0.7941 accuracy in 9.12 minutes whereas the InceptionV3 model did not fall much behind in terms of accuracy with a score of 0.7894 and a runtime of only 10 seconds more than sigmoid. This is obviously a minor improvement but can be proven significant in large-scale applications. However, the rest of the activation functions could not outperform the original model.

The DenseNet121 also performed better using the SMish and Mish activation functions, outweighing the original model's 0.9279 accuracy by scoring 0.9411 and 0.934 respectively. However, the Smish and Mish had the worst runtimes among all the other runs, including the original. This can be seen in more detail in Figure and Table 4.3.

Finally, in Figure and Table 4.4 it can be seen that EfficientNetV2B0 was the model that was improved the most using the Sigmoid function significantly outperforming

the original model's performance in both accuracy and runtime. For the rest of the models, MobileNet and ResNEt, as seen in Figures and Tables 4.5 and 4.6, the activation functions did not improve or outperform the originals.

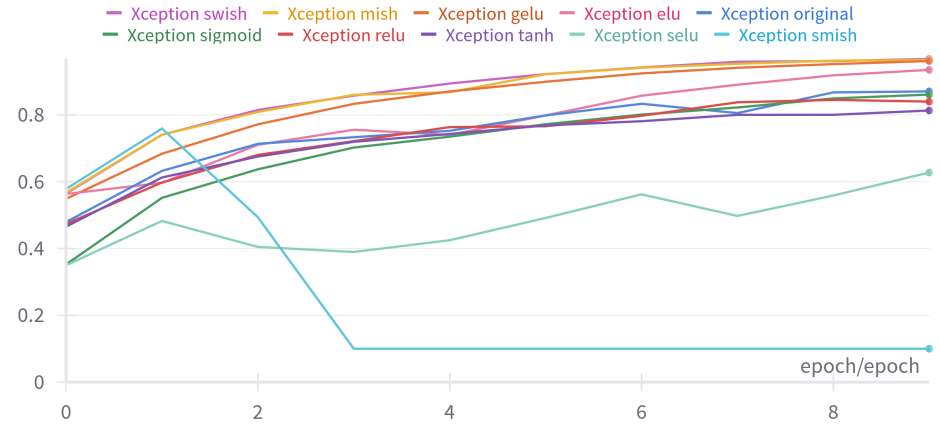


Figure 4.1: CIFAR-10 Xception performance

Xception	Runtime	Accuracy
swish	9m 23s	0.9676
mish	9m 11s	0.9642
gelu	9m 58s	0.9613
elu	7m 15s	0.9348
original	6m 48s	0.8704
sigmoid	7m 11s	0.8609
relu	7m 5s	0.8398
tanh	7m 12s	0.8131
selu	7m 20s	0.6275
smish	12m 5s	0.1

Table 4.1: CIFAR-10 Xception results

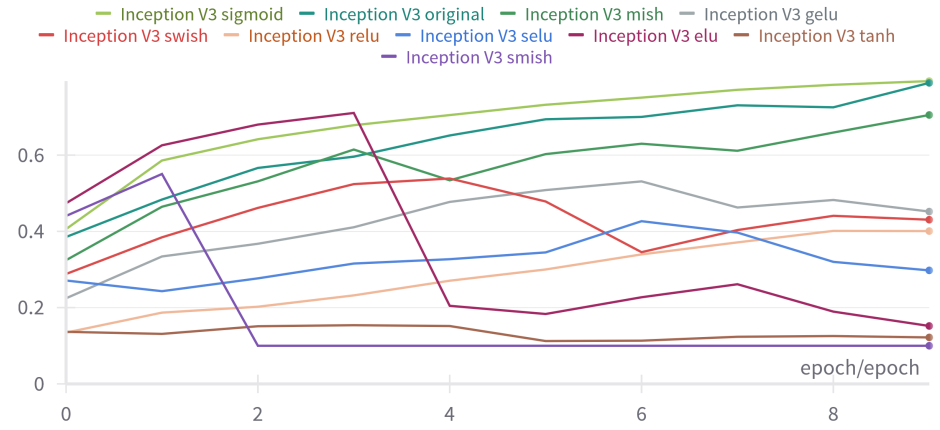


Figure 4.2: CIFAR-10 InceptionV3 performance

InceptionV3	Runtime	Accuracy
sigmoid	9m 12s	0.7940
original	9m 22s	0.7894
mish	11m 17s	0.7052
gelu	12m 8s	0.4519
swish	12m 58s	0.4305
relu	9m 15s	0.4007
selu	9m 45s	0.2976
elu	9m 24s	0.1518
tanh	9m 12s	0.1218
smish	13m 48s	0.1

Table 4.2: CIFAR-10 InceptionV3 results

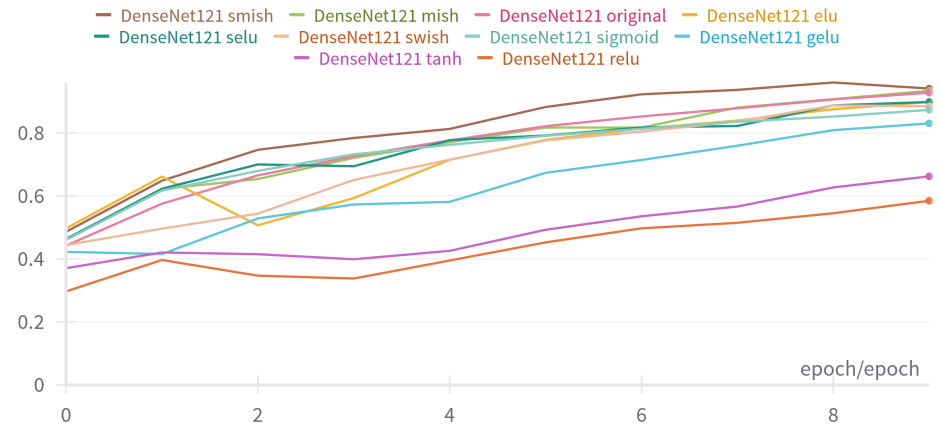


Figure 4.3: CIFAR-10 DenseNet121 performance

DenseNet121	Runtime	Accuracy
smish	24m 27s	0.9410
mish	17m 41s	0.9339
original	17m 9s	0.9279
elu	15m 22s	0.8985
selu	15m 19s	0.8984
swish	20m 4s	0.8850
sigmoid	15m 5s	0.8733
gelu	19m 5s	0.8304
tanh	15m 2s	0.6623
relu	14m 57s	0.5847

Table 4.3: CIFAR-10 DenseNet121 results

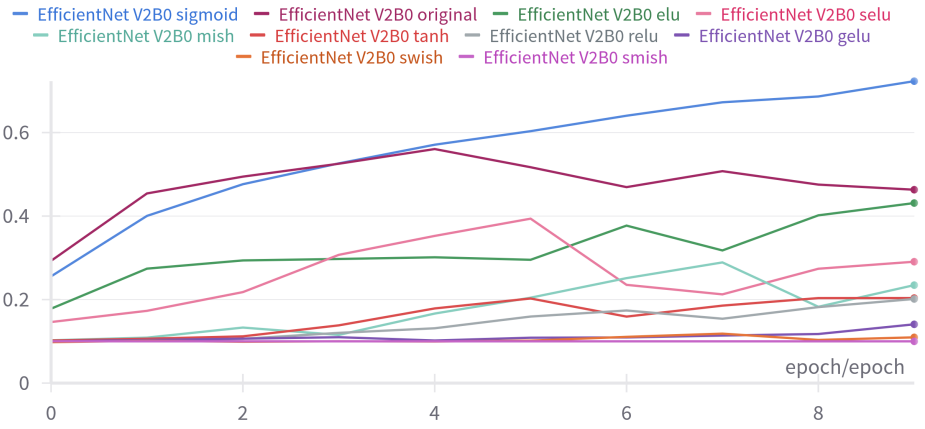


Figure 4.4: CIFAR-10 EfficientNetV2 performance

EfficientNetV2	Runtime	Accuracy
sigmoid	9m 22s	0.7231
original	10m 30s	0.4631
elu	9m 26s	0.4311
selu	9m 30s	0.2906
mish	10m 44s	0.2346
tanh	9m 22s	0.2036
relu	9m 17s	0.2014
gelu	11m 26s	0.1407
swish	11m 59s	0.1095
smish	12m 28s	0.1

Table 4.4: CIFAR-10 EfficientNetV2 results

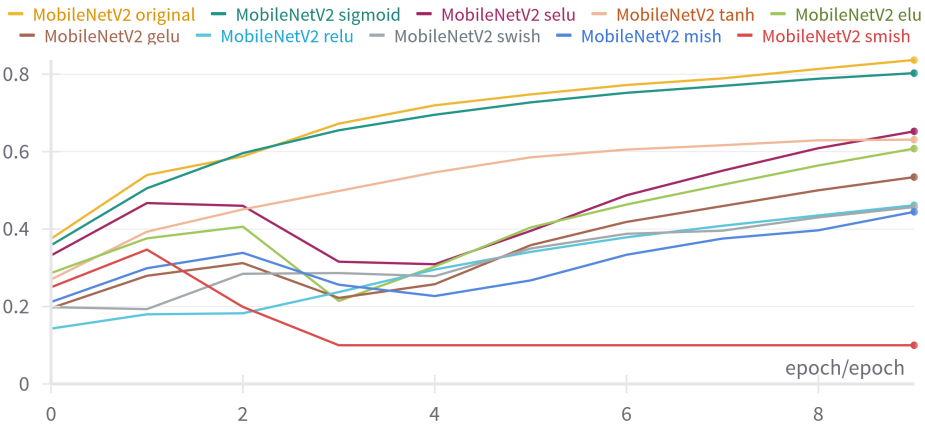


Figure 4.5: CIFAR-10 MobileNetV2 performance

MobileNetV2	Runtime	Accuracy
original	6m 23s	0.8367
sigmoid	6m 50s	0.8027
selu	5m 56s	0.6523
tanh	5m 53s	0.6309
elu	5m 58s	0.6078
gelu	6m 59s	0.5342
relu	6m 14s	0.4615
swish	7m	0.4569
mish	6m 39s	0.4445
smish	7m 50s	0.1

Table 4.5: CIFAR-10 MobileNetV2 results

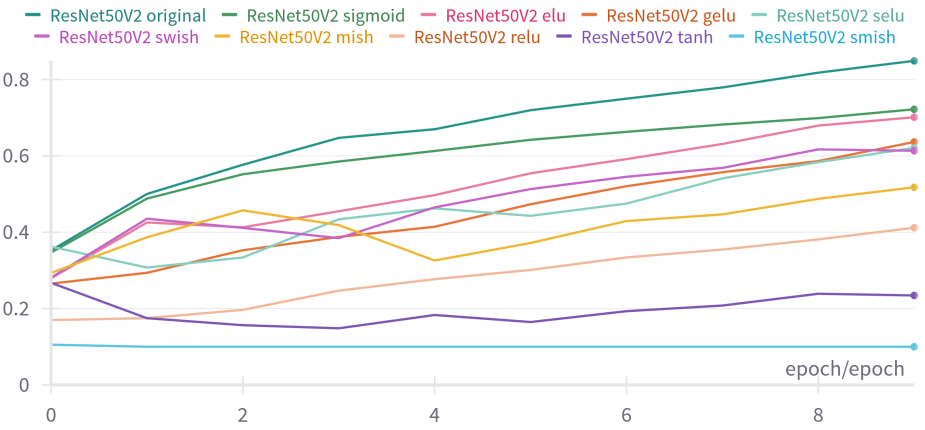


Figure 4.6: CIFAR-10 ResNet50V2 performance

ResNet50V2	Runtime	Accuracy
original	7m 20s	0.8489
sigmoid	7m 33s	0.7220
elu	7m 26s	0.7012
gelu	9m 50s	0.6363
selu	7m 26s	0.6209
swish	9m 40s	0.6133
mish	8m 59s	0.5176
relu	6m 59s	0.4117
tanh	7m 32s	0.2344
smish	11m 43s	0.1

Table 4.6: CIFAR-10 ResNet50V2 results

4.1.2 Cifar-100 tests

When stressing the models by increasing the classes to 100 with the Cifar-100 dataset, most models performed best in their original architectures. However, DenseNet's accuracy was improved from 0.8978 accuracy (original model) to 0.9135 with the ELU function but with a slightly higher training time. This can be observed in Figure and Table 4.9. Moreover, EfficientNet's accuracy and runtime were improved using the Sigmoid function in both terms of accuracy and runtime, as illustrated in Figure and Table 4.10. In the same case, SELU and ELU managed to score higher than the original as well. The other networks did not seem to improve with the experiments as seen in Figures and Tables 4.7, 4.8, 4.11, 4.12. This can be due to the increased number of classes and number of images, it seems that the large-scale focused architecture of the originals performed better. However, it is worth mentioning that in cases such as MobileNet, EfficientNet, and InceptionV3, although the sigmoid function did not always manage to outperform the original's accuracy, it did achieve slightly less accuracy but significantly faster.

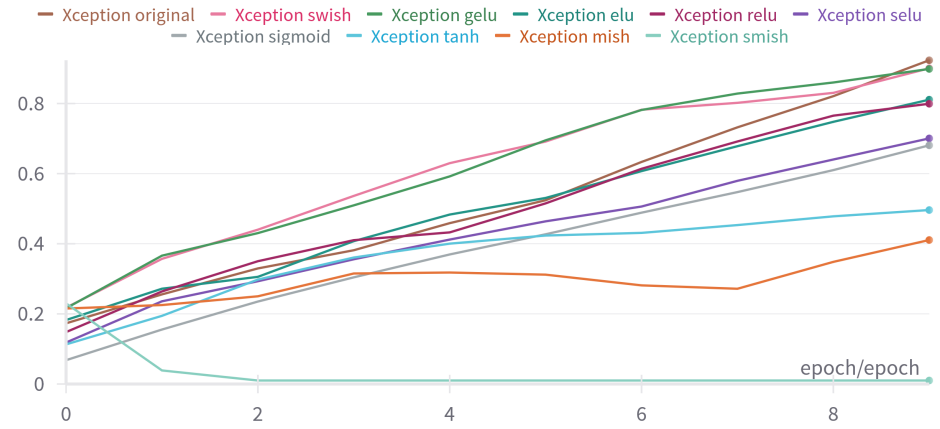


Figure 4.7: CIFAR-100 Xception performance

Xception	Runtime	Accuracy
original	8m 40s	0.9233
swish	9m 26s	0.9004
gelu	9m 58s	0.8985
elu	7m 3s	0.8109
relu	7m 5s	0.7993
selu	7m 15s	0.7004
sigmoid	7m 9s	0.6807
tanh	7m 4s	0.4961
mish	9m 12s	0.4106
smish	12m 4s	0.1

Table 4.7: CIFAR-100 Xception Results

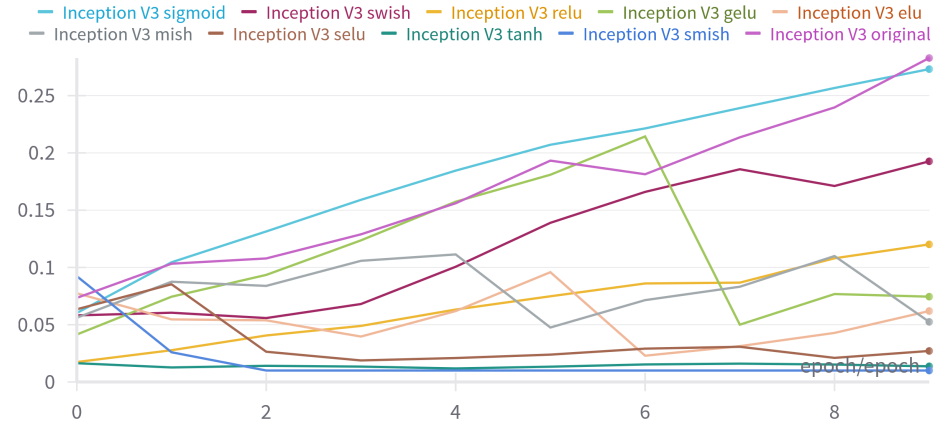


Figure 4.8: CIFAR-100 InceptionV3 performance

InceptionV3	Runtime	Accuracy
original	13m 13s	0.2829
sigmoid	9m 29s	0.2731
swish	12m 58s	0.1926
relu	9m 26s	0.1201
gelu	12m 15s	0.0744
elu	9m 28s	0.0619
mish	17m 16s	0.0524
selu	9m 39s	0.0270
tanh	9m 20s	0.0136
smish	13m 21s	0.1

Table 4.8: CIFAR-100 InceptionV3 Results

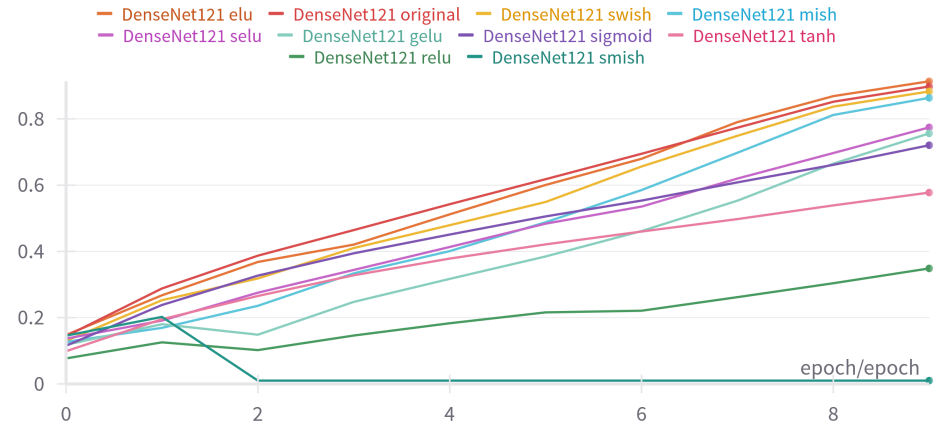


Figure 4.9: CIFAR-100 DenseNet121 performance

DenseNet121	Runtime	Accuracy
elu	15m 17s	0.9135
original	14m 51s	0.8978
swish	20m 15s	0.8829
mish	52m 49s	0.8634
selu	15m 45s	0.7743
gelu	18m 59s	0.7562
sigmoid	14m 40s	0.7204
tanh	14m 57s	0.5775
relu	14m 56s	0.3486
smish	20m 42s	0.1

Table 4.9: CIFAR-100 DenseNet121 Results

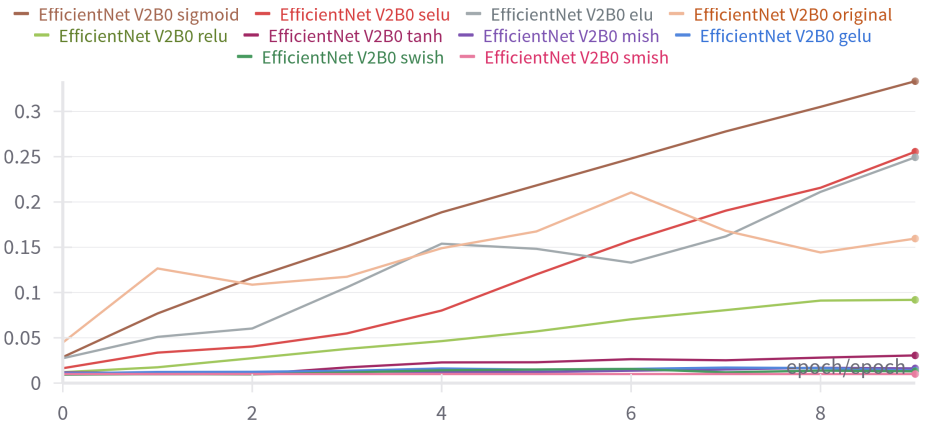


Figure 4.10: CIFAR-100 EfficientNetV2 performance

EfficientNetV2	Runtime	Accuracy
sigmoid	9m 9s	0.3333
selu	9m 32s	0.2554
elu	9m 20s	0.2493
original	10m	0.1595
relu	9m 12s	0.0919
tanh	9m 6s	0.0305
mish	10m 37s	0.0162
gelu	11m 20s	0.0138
swish	11m 55s	0.0130
smish	12m 33s	0.1

Table 4.10: CIFAR-100 EfficientNetV2 Results

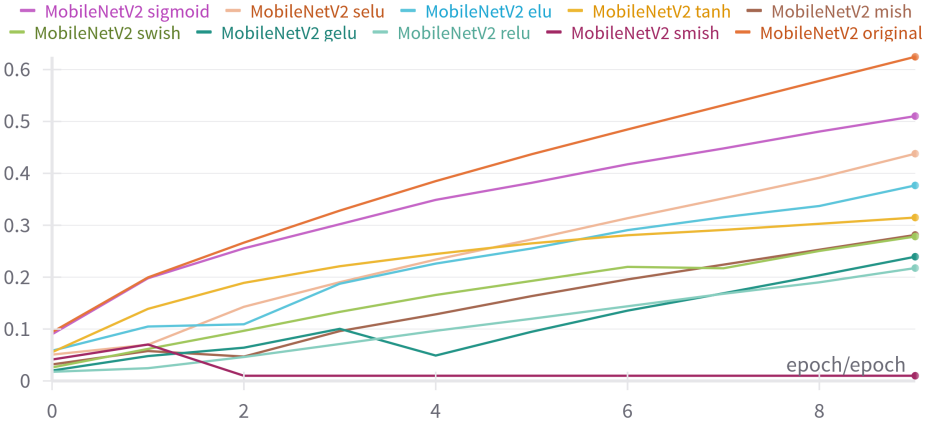


Figure 4.11: CIFAR-100 MobileNetV2 performance

MobileNetV2	Runtime	Accuracy
original	9m 14s	0.6246
sigmoid	6m 1s	0.5102
selu	5m 56s	0.4379
elu	6m 1s	0.3767
tanh	6m 3s	0.3148
mish	6m 43s	0.2813
swish	7m	0.2782
gelu	7m 7s	0.2393
relu	5m 57s	0.2175
smish	7m 46s	0.1

Table 4.11: CIFAR-100 MobileNetV2 Results

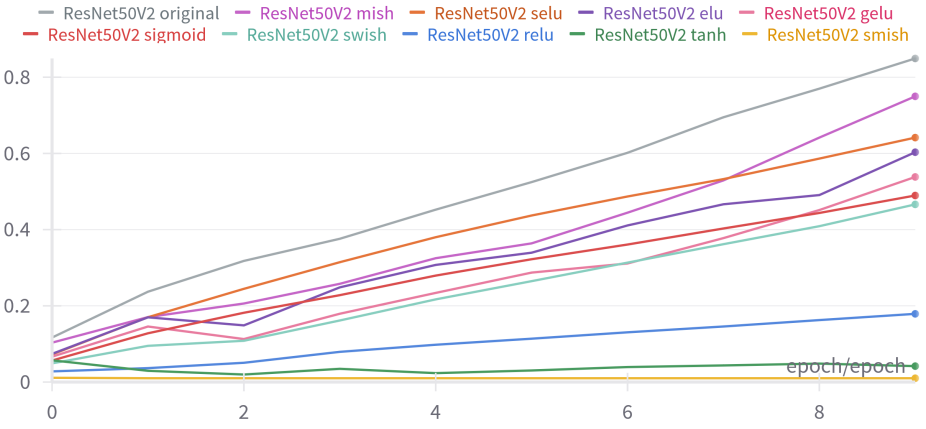


Figure 4.12: CIFAR-100 ResNet50V2 performance

ResNet50V2	Runtime	Accuracy
original	7m 24s	0.8492
mish	8m 56s	0.7497
selu	7m 27s	0.6414
elu	7m 29s	0.6031
gelu	9m 54s	0.5382
sigmoid	7m 29s	0.4895
swish	9m 34s	0.4661
relu	7m 25s	0.1789
tanh	7m 27s	0.0417
smish	11m 43s	0.1

Table 4.12: CIFAR-100 ResNet50V2 Results

4.1.3 STL-10 tests

This time all the models performed better than with their original activations. Xception's accuracy was improved using the Smish, swish, and ELU activation functions with scores of 0.8952, 0.8934, and 0.8844 respectively, contrary to the original model's score of 0.8768. However, only ELU had both better accuracy and runtime than the original model since the other 2 were slower by 1 minute in comparison to the original model's runtime. The above can be observed in further detail in Figure and Table 4.13.

Figure and Table 4.14 depict the InceptionV3 network's performance. InceptionV3 was significantly improved using other activation functions, since all except ReLU and Tanh scored a higher accuracy than the original, with the highest scoring being ELU, Smish, and Sigmoid with 0.7178, 0.6474 and 0.5738 accuracy scores respectively. For a second time, ELU was the highest scoring with the lowest runtime.

A similar case was also the one of DenseNet,which is shown in Figure and Table 4.15, the accuracy of which was improved majorly using the Smish, ELU, and Swish functions, with accuracy scores of 0.887, 0.8014, and 0.7701 respectively. The ELU function also in this case was the one with the lowest runtime.

Furthermore, both EfficientNEt and MobileNEt were improved using the Sigmoid function, the performance of which not only outpaced the original performances but also with the lowest runtime in both cases. The above can be seen in further detail in the Figures and Tables 4.16, 4.17.

Finally, ResNet's performance was improved using the Mish, SELU, ELU, and Sigmoid functions, with the highest being Mish with an accuracy of 0.6162. Except for Mish, all the functions that outperformed the original also had a lower runtime with the fastest being ELU, as analyzed in Figure and Table 4.18.

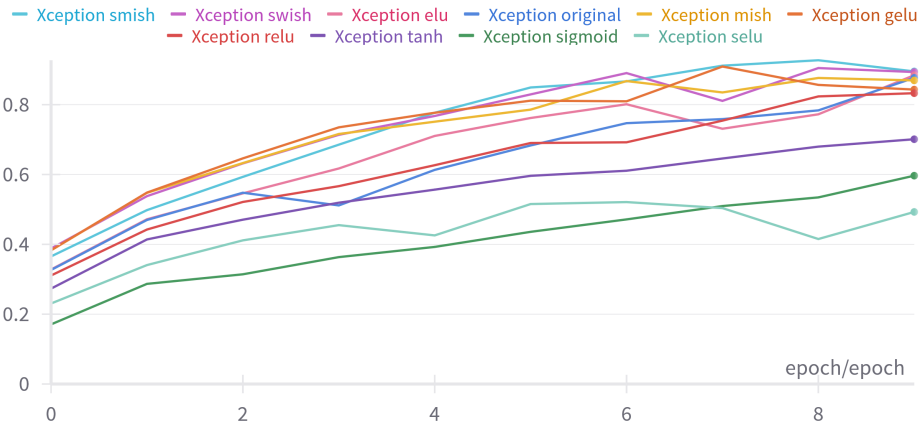


Figure 4.13: STL-10 Xception performance

Xception	Runtime	Accuracy
smish	3m 51s	0.8952
swish	3m 48s	0.8934
elu	2m 50s	0.8844
original	2m 51s	0.8768
mish	3m 9s	0.8694
gelu	3m 21s	0.8432
relu	2m 46s	0.8327
tanh	2m 51s	0.7009
sigmoid	2m 50s	0.5965
selu	2m 50	0.4927

Table 4.13: STL-10 Xception Results

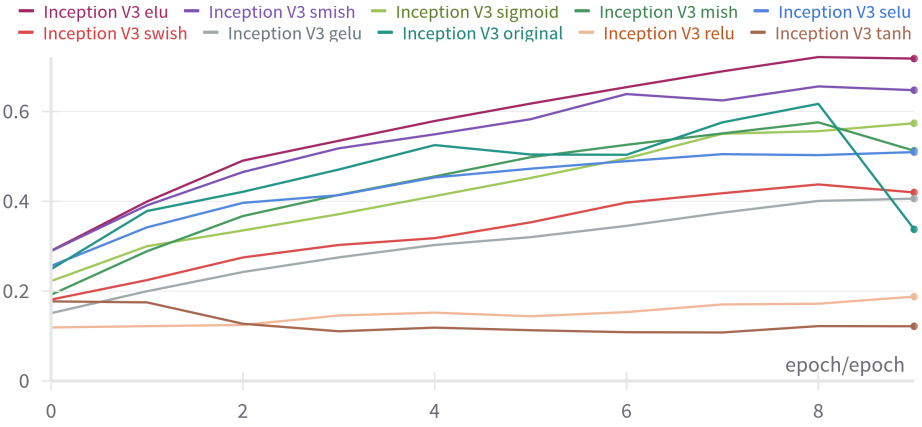


Figure 4.14: STL-10 InceptionV3 performance

InceptionV3	Runtime	Accuracy
elu	4m 30s	0.7178
smish	5m 59s	0.6474
sigmoid	4m 27s	0.5738
mish	4m 53s	0.5126
selu	4m 30s	0.5095
swish	6m 42s	0.4198
gelu	5m 10s	0.4059
original	4m 30s	0.3375
relu	4m 26s	0.1878
tanh	4m 28s	0.1217

Table 4.14: STL-10 InceptionV3 Results

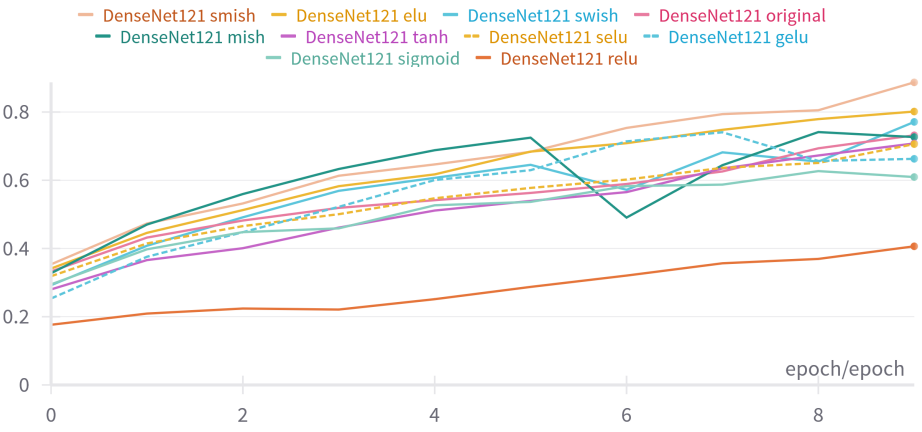


Figure 4.15: STL-10 DenseNet121 performance

DenseNet121	Runtime	Accuracy
smish	8m 45s	0.8870
elu	6m 59	0.8014
swish	10m	0.7710
original	7m 14s	0.7318
mish	7m 21s	0.7265
tanh	6m 56s	0.7080
selu	6m 44s	0.7059
gelu	7m	0.6628
sigmoid	7m	0.6093
relu	6m 55s	0.4063

Table 4.15: STL-10 DenseNet121 Results

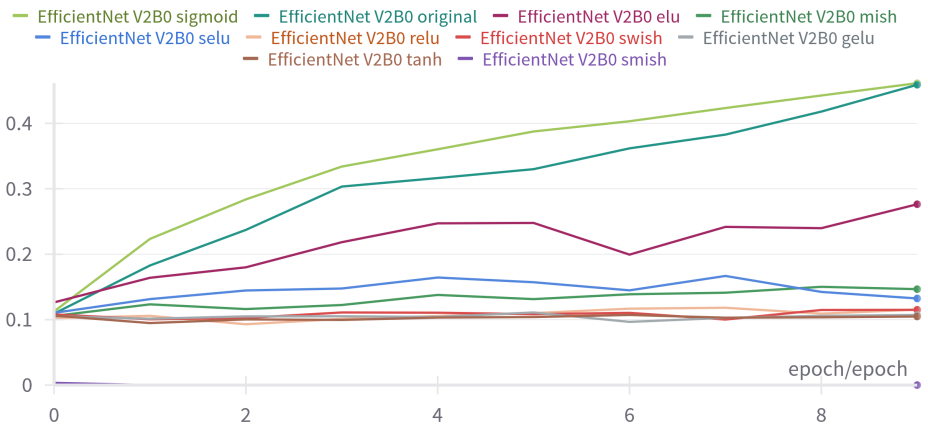


Figure 4.16: STL-10 EfficientNetV2 performance

EfficientNetV2	Runtime	Accuracy
sigmoid	4m 17s	0.4614
original	4m 55s	0.4589
elu	4m 22s	0.2763
mish	4m 38s	0.1465
selu	4m 22s	0.1324
relu	4m 22s	0.1154
swish	5m 47s	0.1150
gelu	4m 54s	0.1071
tanh	4m 22s	0.1048
smish	5m 12s	0.1

Table 4.16: STL-10 EfficientNetV2 Results

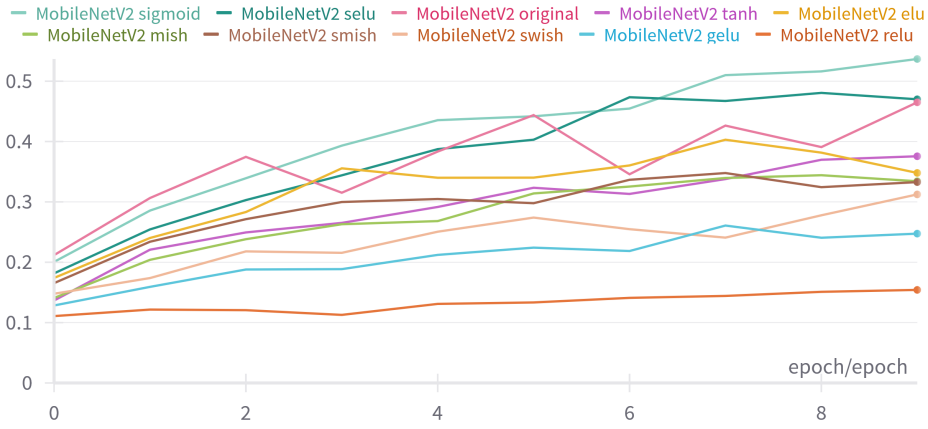


Figure 4.17: STL-10 MobileNetV2 performance

MobileNetV2	Runtime	Accuracy
sigmoid	2m 48s	0.5368
selu	2m 53s	0.4699
original	2m 54s	0.4650
tanh	2m 52s	0.3756
elu	2m 52s	0.3479
mish	3m 3s	0.3341
smish	3m 14s	0.3328
swish	3m 25s	0.3124
gelu	3m 2s	0.2474
relu	2m 50s	0.1542

Table 4.17: STL-10 MobileNetV2 Results

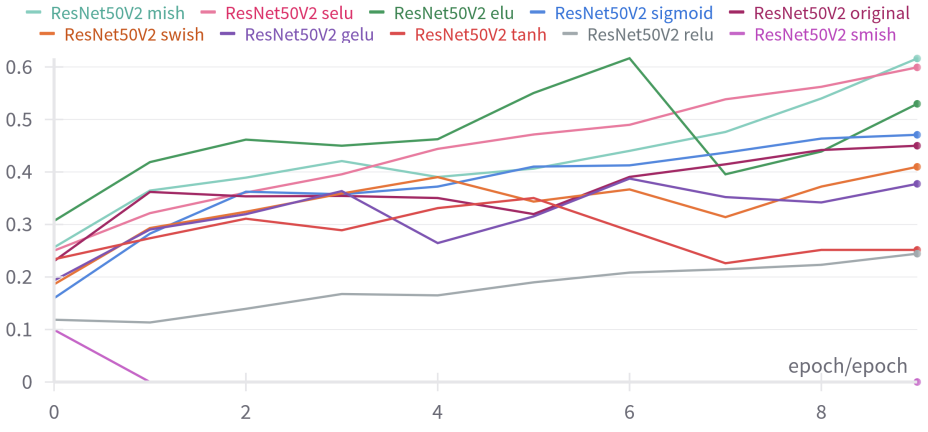


Figure 4.18: STL-10 ResNet50V2 performance

ResNet50V2	Runtime	Accuracy
mish	3m 31	0.6161
selu	3m 21	0.5992
elu	3m 16s	0.5297
sigmoid	3m 20s	0.4708
original	3m 26s	0.4499
swish	11m 27s	0.4097
gelu	3m 53s	0.3774
tanh	3m 17s	0.2515
relu	3m 20s	0.2443
smish	4m 12s	0.1

Table 4.18: STL-10 ResNet50V2 Results

4.1.4 SVHN tests

In the case of SVHN the original architectures of all the models, except Xception and EfficientNet, performed the best, as seen more comprehensively in Figures and Tables 4.20, 4.21, 4.23, 4.24. In Xception's case, the Swish and GELU functions performed the highest with accuracy scores of 0.9871 and 0.9841 respectively, and managed faster runtimes than the original's. Figure and Table 4.19 depicts Xception's performance. In EfficientNet's case which is shown in Figure and Table 4.22, the ELU function slightly outperformed the original but essentially their performance was equal as well as their Runtimes. However, the Sigmoid function in most cases came close to the original's performance again.

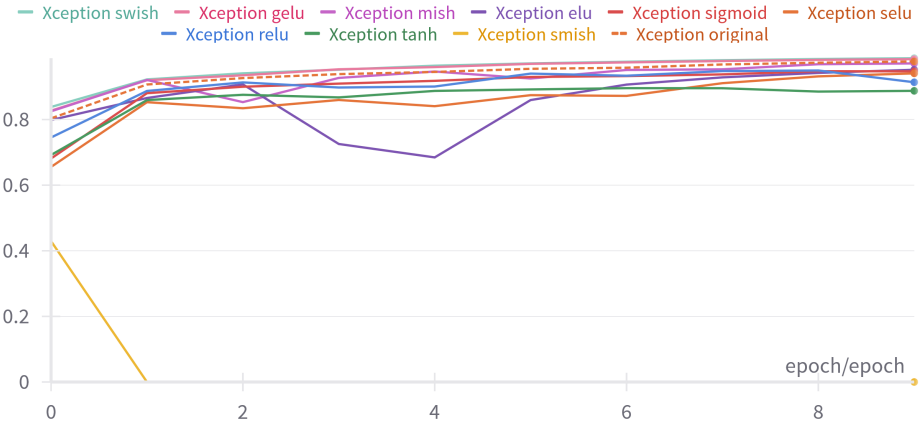


Figure 4.19: SVHN Xception performance

Xception	Runtime	Accuracy
swish	13m 42s	0.9870
gelu	14m 46s	0.9841
original	15m	0.9775
mish	13m 42s	0.9711
elu	10m 2s	0.9518
sigmoid	9m 57s	0.9469
selu	10m 1a	0.9407
relu	11m 34s	0.9136
tanh	10m	0.8874
smish	17m 51s	0.1

Table 4.19: SVHN Xception Results

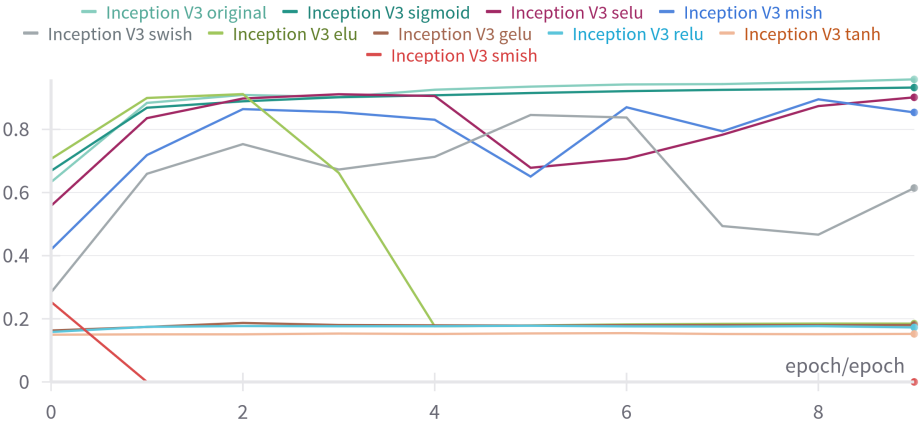


Figure 4.20: SVHN InceptionV3 performance

InceptionV3	Runtime	Accuracy
original	12m 46s	0.9585
sigmoid	12m 45s	0.9325
selu	13m 20s	0.9015
mish	16m 39s	0.8538
swish	16m 29s	0.6143
elu	13m 14s	0.1847
gelu	16m 29s	0.1804
relu	12m 35s	0.1727
tanh	12m 44s	0.1522
smish	19m 22s	0.1

Table 4.20: SVHN InceptionV3 Results

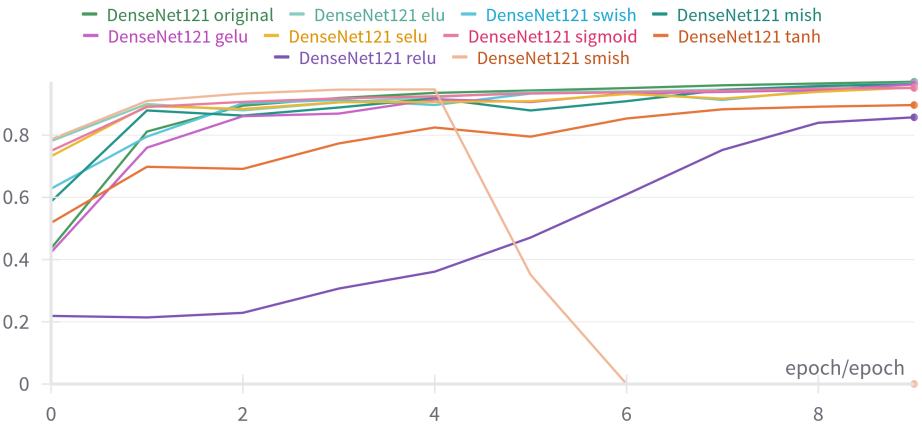


Figure 4.21: SVHN DenseNet121 performance

DenseNet121	Runtime	Accuracy
original	12m 46s	0.9718
sigmoid	12m 45s	0.9516
selu	13m 20s	0.9530
mish	16m 39s	0.9638
swish	16m 29	0.9645
elu	13m 14s	0.9663
gelu	16m 29s	0.9632
relu	12m 35s	0.8574
tanh	12m 44s	0.8968
smish	19m 22s	0.1

Table 4.21: SVHN DenseNet121 Results

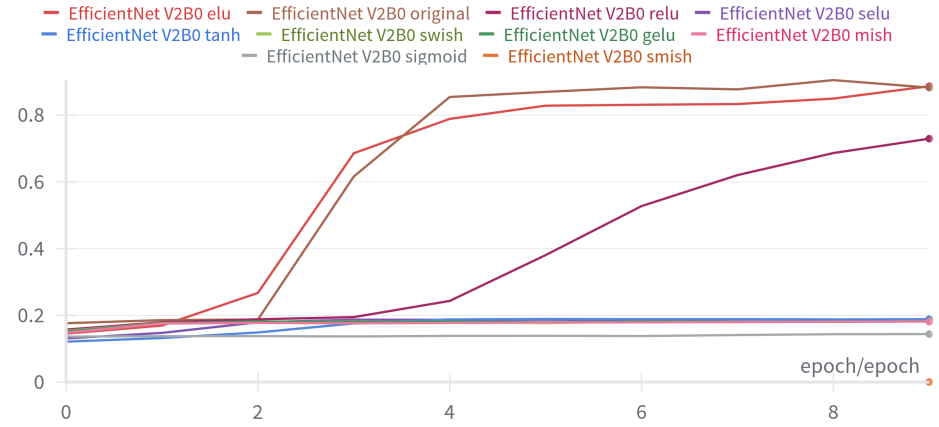


Figure 4.22: SVHN EfficientNetV2 performance

EfficientNetV2	Runtime	Accuracy
elu	13m 9s	0.8867
original	13m 8s	0.8826
relu	12m 35s	0.7294
selu	22m 31s	0.1884
tanh	12m 51s	0.1878
swish	15m 20s	0.1822
gelu	15m 6s	0.1816
mish	14m 56s	0.1812
sigmoid	12m 54s	0.1439
smish	16m 37s	0.1

Table 4.22: SVHN EfficientNetV2 Results

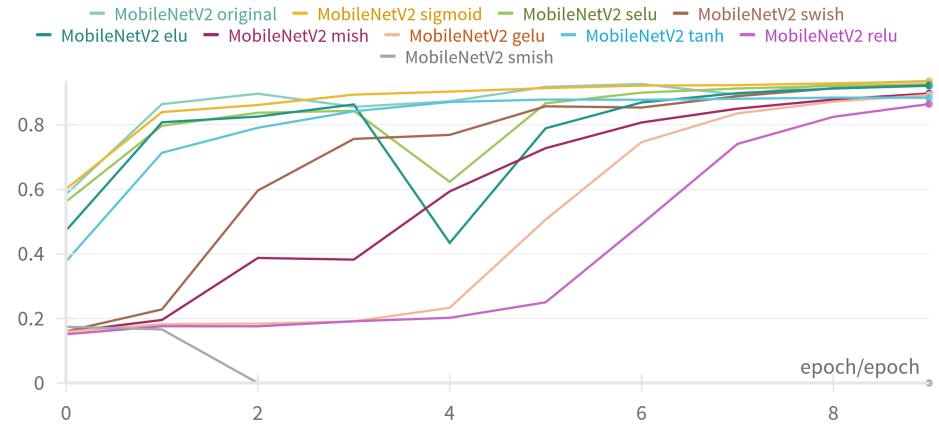


Figure 4.23: SVHN MobileNetV2 performance

MobileNetV2	Runtime	Accuracy
original	8m 54s	0.9359
sigmoid	8m 43s	0.9340
selu	8m 25s	0.9255
swish	9m 20s	0.9221
elu	8m 13s	0.9209
mish	9m 29s	0.8970
gelu	11m 45s	0.8925
tanh	8m 28s	0.8838
relu	8m 49s	0.8643
smish	11m 15s	0.1

Table 4.23: SVHN MobileNetV2 Results

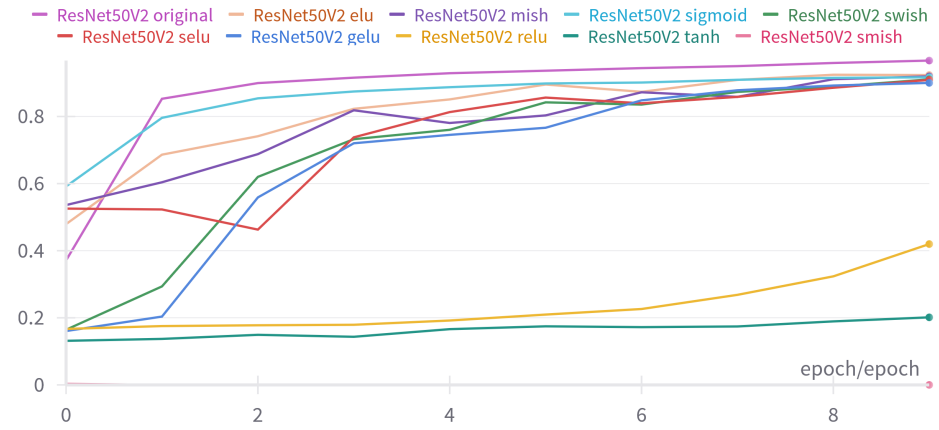


Figure 4.24: SVHN ResNet50V2 performance

ResNet50V2	Runtime	Accuracy
original	9m 28s	0.9666
elu	10m 34s	0.9235
mish	13m 54s	0.9196
sigmoid	9m 53s	0.9167
swish	13m 51s	0.9106
selu	10m 48s	0.9091
gelu	15m 3s	0.9001
relu	10m 27s	0.4197
tanh	10m 42s	0.2013
smish	20m 58s	0.1

Table 4.24: SVHN ResNet50V2 Results

4.1.5 Improvements over the original models

Many models' performance was improved in many cases to both low and high extents. As is apparent from the results, Swish, Mish, GELU, ELU, Sigmoid, and SMish (only in STL-10 and CIFAR-10) were the functions that were proven to be of great use in this work. RELU and Tanh did not seem to be an improvement, which is expected since they have many vulnerabilities as mentioned earlier in this work.

However, the ones that seemed to be the highest achieving of all were the Sigmoid and ELU functions. The reason is that in most cases not only did they improve the model's accuracy but also when used the training time of the model was reduced. The same cannot be said about most of the other activation functions. Although they improved the model's performance hardly could they reduce the original model's training time. This can be due to the fact that generally, the ReLU variations such as the ones used in this work, are computationally expensive. This can also be seen in Figure 4.25 which shows the Power Usage Graph of the Top functions that were the most power costly. ELU certainly seems to be the exception to this rule because although it is considered resource-costly, it does not require computationally expensive batch normalization, thus resulting in faster training times.

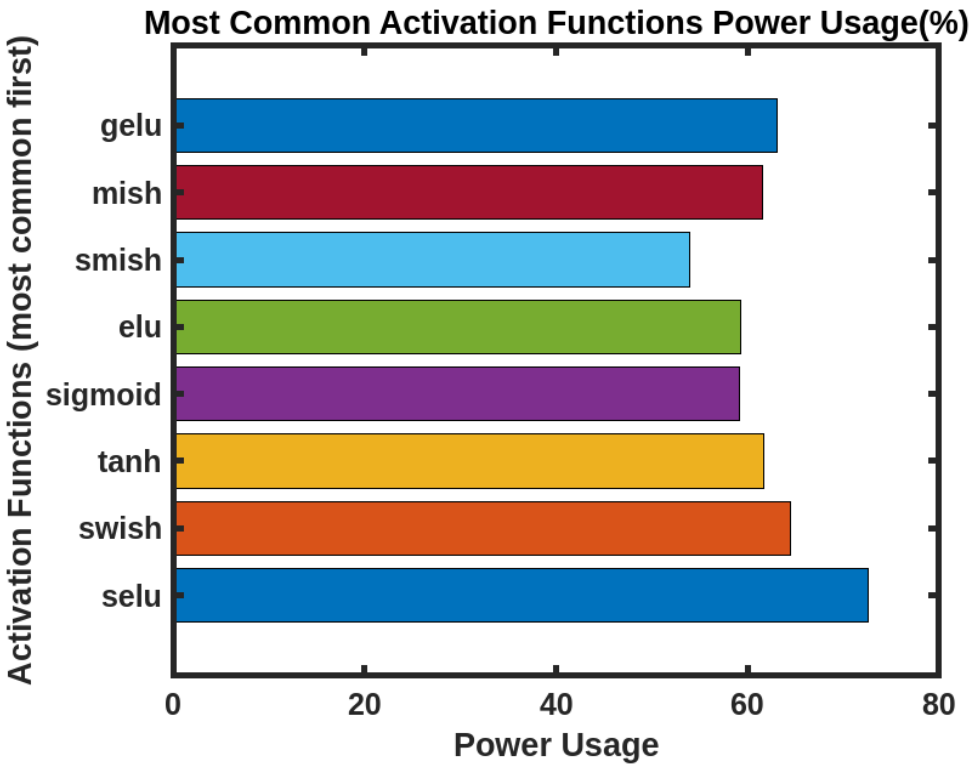


Figure 4.25: Power usage graph of the activation functions that improved the original models

4.1.6 The SMish problem

The case of the performance of the Smish activation function was special. Despite the fact that the function for most cases did not perform at all, as the model did not train completely, thus the 0.1 accuracy, it did perform highly for the STL-10 and the CIFAR-10 (only for the DenseNet model) datasets. Such cases can be attributed to many causes such as:

- Initialization Issues, meaning that the initial weights and biases are not suited or even applicable for the specific activation function.
- The learning rate might be not appropriate for the specific function.
- Model complexity, which in this case can be high. Since the architectures of the pre-trained models are already of large scale and complex, the Smish function in the entirety of the model may increase the complexity, thus leading to the Vanishing Gradients Problem.
- Most models do not use the same activation function in the entirety of the model. Rather, each layer may use different activation functions depending on the architecture, the layer role, the flow of data, etc. This way many functional problems or unwanted scenarios can be avoided such as the cases mentioned above

The first possible explanation, the initialization issues, may not be apt to every case of this work since in many cases the model using the SMish function performs well during the first epoch of training and then fails. The other possible factors of the problematic performance require further research as, at their core, suggest tuning issues that might be resolved through testing different hyperparameters. However, the root of the problematic performance can be accounted to tunning issues, such as the ones suggested in the possible factors of the problem above, which can be tackled through testing

Chapter 5

Conclusions

5.1 Challenges

The field of image classification, while practically yet in its early stages, has surely been radically advanced in recent years providing optimistic results. However, there are still challenges that need to be addressed in order for image classification to become a powerful tool that can be applied to real-world solutions.

There is a constant need for more and better datasets, especially for real-world datasets that consist of images with noise, distortion, and more complexity so that the models can be trained in more everyday scenarios instead of perfect conditions. This can lead to more generalized models, thus more accurate and trustworthy.

On the software side, there is constant experimentation and research to develop better and more efficient models, through innovative architectures, activation functions, or better tuning. Improvements can also be made through testing different combinations of the already existing tools such as the task of this work. This can lead to models that can use the hardware resources more efficiently and perform better and faster as required for real-world applications.

5.2 Conclusions

The development of more improved image classification models is a challenging task that constantly is researched. The need for more efficient, accurate, fast, and generalized image classification models is higher than ever before. The technological

advances made it possible for the hardware to catch up with the requirements of theoretical concepts. Nowadays when the requirements are met theoretical concepts such as AI and ML can be practically researched and applied to innovative real-world solutions. Image classification can be undoubtedly a powerful tool to be implemented in many everyday applications. However, there are many challenges that need to be tackled in order for it to reach its full potential.

In this work, different image classification neural networks were modified to use a different activation function than their original to seek performance improvements. The experiments were surely optimistic as in several cases using a different activation function resulted in not only a better performance but also faster. In the overall 300 runs, the most outperforming functions were the ELU, Sigmoid, and Mish. Although ELU and Sigmoid managed to improve the networks not only in accuracy but also in execution time, Mish was one of the slowest-performing functions. There were also some cases in which the original architecture was proven best.

5.3 Future work

For future research, more activation functions and different combinations of them need to be tested in more pre-trained models in order to find out if more improvements can be made to pre-trained models. Furthermore, more datasets need to be tested in this process to find out if the current results are only in this work's case scenario or if there are similarities in performance with other datasets.

Bibliography

[1] I. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, 03 2021.

[2] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," Apr. 2017.

[3] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 2015.

[5] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016.

[6] T. Ahmed and N. H. N. Sabab, "Classification and understanding of cloud structures via satellite images with efficientunet," *SN Comput. Sci.*, vol. 3, dec 2021.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[8] Ò. Lorente, I. Riera, and A. Rana, "Image classification with classic and deep learning techniques," *CoRR*, vol. abs/2105.04895, 2021.

[9] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *CoRR*, vol. abs/2104.05314, 2021.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[11] E. Grossi and M. Buscema, "Introduction to artificial neural networks," *European journal of gastroenterology hepatology*, vol. 19, pp. 1046–54, 01 2008.

[12] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "A comprehensive survey and performance analysis of activation functions in deep learning," *CoRR*, vol. abs/2109.14545, 2021.

[13] S. Narayan, "The generalized sigmoid activation function: Competitive supervised learning," *Information Sciences*, vol. 99, no. 1, pp. 69–82, 1997.

[14] A. Namin, K. Leboeuf, R. Muscedere, H. Wu, and M. Ahmadi, "Efficient hardware implementation of the hyperbolic tangent sigmoid function," pp. 2117 – 2120, 06 2009.

[15] D. Hendrycks and K. Gimpel, "Bridging nonlinearities and stochastic regularizers with gaussian error linear units," *CoRR*, vol. abs/1606.08415, 2016.

[16] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *CoRR*, vol. abs/1706.02515, 2017.

[17] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *CoRR*, vol. abs/1710.05941, 2017.

[18] D. Misra, "Mish: A self regularized non-monotonic activation function," 2020.

[19] X. Wang, H. Ren, and A. Wang, "Smish: A novel activation function for deep learning methods," *Electronics*, vol. 11, p. 540, 02 2022.

[20] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, pp. 107–116, 04 1998.

[21] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *CoRR*, vol. abs/1511.08458, 2015.

[22] S. Verma, A. Chug, and A. P. Singh, "Revisiting activation functions: empirical evaluation for image understanding and classification," *Multimedia Tools and Applications*, vol. 83, pp. 18497–18536, Feb 2024.

[23] X. Zhang, D. Chang, W. Qi, and Z. Zhan, "A study on different functionalities and performances among different activation functions across different anns for image classification," *Journal of Physics: Conference Series*, vol. 1732, p. 012026, jan 2021.

[24] A. J. Dhruv, R. Patel, and N. Doshi, "Python: The Most Advanced Programming Language for Computer Science Applications:," in *Proceedings of the International Conference on Culture Heritage, Education, Sustainable Tourism, and Innovation Technologies*, (Medan, Indonesia), pp. 292–299, SCITEPRESS - Science and Technology Publications, 2020.

[25] J. Lakshmi, "Machine learning techniques using python for data analysis in performance evaluation," *International Journal of Intelligent Systems Technologies and Applications*, vol. 17, no. 1/2, p. 3, 2018.

[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for large-scale machine learning," 2016.

[27] "TensorFlow Datasets, a collection of ready-to-use datasets." `https://www.tensorflow.org/datasets`.

[28] N. Ketkar, *Introduction to Keras*, pp. 97–111. Berkeley, CA: Apress, 2017.

[29] F. J. J. Joseph, S. Nonsiri, and A. Monsakul, "Keras and TensorFlow: A Hands-On Experience," in *Advanced Deep Learning for Engineers and Scientists* (K. B. Prakash, R. Kannan, S. Alexander, and G. R. Kanagachidambaresan, eds.), pp. 85–111, Cham: Springer International Publishing, 2021.

[30] D. Rolon-Mérette, M. Ross, T. Rolon-Mérette, and K. Church, "Introduction to Anaconda and Python: Installation and setup," *The Quantitative Methods for Psychology*, vol. 16, pp. S3–S11, May 2020.

[31] L. Biewald, "Experiment tracking with weights and biases," 2020. Software available from wandb.com.

[32] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, Y. Yao, A. Zhang, L. Zhang, W. Han, M. Huang, Q. Jin, Y. Lan, Y. Liu, Z. Liu, Z. Lu, X. Qiu, R. Song, J. Tang, J.-R. Wen, J. Yuan, W. X. Zhao, and J. Zhu, "Pre-trained models: Past, present and future," 2021.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[34] X. Wu, R. Liu, H. Yang, and Z. Chen, "An xception based convolutional neural network for scene image classification with transfer learning," in *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*, pp. 262–267, 2020.

[35] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114, PMLR, 09–15 Jun 2019.

[36] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018.

[37] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[38] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[39] A. Coates, A. Ng, and H. Lee, "An Analysis of Single Layer Networks in Unsupervised Feature Learning," in *AISTATS*, 2011. `https://cs.stanford.edu/~acoates/papers/coatesleeng_aistats_2011.pdf`.

[40] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101 – mining discriminative components with random forests," in *European Conference on Computer Vision*, 2014.