A Comparison Review of Optimizers and Activation Functions For Convolutional Neural Networks

Ahmad Awad
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP062406@mail.apu.edu.my

Tharun Muthukumaran Umadevi
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP066176@mail.apu.edu.my

Joey Ng Ceng Yi
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP060794@mail.apu.edu.my

Toh Jian En
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP061216@mail.apu.edu.my

Koo Susan
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP067306@mail.apu.edu.my

Zailan Arabee Abdul Salam
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
zailan@apu.edu.my

***Abstract*—Convolutional Neural Networks (CNN) are widely used in today's world for research on image classification and image identification. In this research, exploration is made into one type of CNN using transfer learning by implementing the DensNet-161 model into classifying 133 different types of dog breeds in a total of 8,351 images split between training, validation, and testing. Only 7,515 images will be used for training and validation of a ratio of 89:11 respectively. This research aims to identify the accuracies and performances of Rectified Linear Unit (ReLU), Leaky ReLU, and Exponential Linear Unit (ELU) activation functions along with Adaptive Moment Estimation (Adam), Adaptive Gradient Algorithm (Adagrad), and Stochastic Gradient Descent (SGD) optimization functions with learning rates (lr) of 0.001, 0.01, and 0.1.**

Keywords— dog breed classification, convolutional neural networks (cnn), artificial neural network (ann), transfer learning, optimization functions, activation functions, learning rate.

I. INTRODUCTION

The evolution of dogs since time immemorial has eventually led to hundreds of dog breeds in the world today. According to Federation Cynologique Internationale (2023), a total amount of 360 registered dog breeds exists in the modern world and some of them even share the same features despite being of different breeds (Oliver, 2023). Hence, it could be extremely arduous even for dog experts to identify and remember each dog breed. With the contemporary advancements in Artificial Intelligence and introduction of Machine Learning, more effective ways of identifying dog breeds specifically using Deep Learning (DL) approaches and Convolutional Neural Networks (CNN) were developed. Convolutional Neural Networks are highly endorsed for surveying visual images (Long K et al., 2022). However, training a CNN model has been a tough, complicated, and time-consuming task to do since it needs a large dataset to be fed into it until it can become a usable model with high accuracy (Castillo, 2023). For most of the research, the size and the number of images contained in the dataset used might be not sufficient to gain high accuracy (Koehrsen, 2021). Furthermore, most researchers have difficulty creating a new CNN model from scratch. This is where transfer learning techniques are used to reduce time consumption on creating multiple model architectures. Transfer learning is a technique by reusing a pretrained model to solve a new problem that is popular in the field of data science because it allows researchers to train or work with the deep neural network with only a relatively low amount of data (Sharma, 2021). By this, a better and more compatible model is published. In this research paper, the utility of transfer learning for identifying dog breeds is explored. The research uses DenseNet-161 as the pretrained model and initially uses the Rectified Linear Unit (ReLU) as the activation function along with the Adaptive Moment Estimation (Adam) optimization function. The paper also explores the employment of Stochastic Gradient Descent (SGD) and Adaptive Gradient Algorithm (Adagrad) optimization functions in comparison to Adam and the employment of Leaky Rectified Linear Unit (Leaky ReLU) and Exponential Linear Unit (ELU) in comparison to ReLU.

II. LITERATURE REVIEW

According to Naufal, Ema, and Gamma (2022), the research was on dog breeds classification using Convolutional Neural Network (CNN). That being said, CNN faces great difficulty in differentiating groups of dogs with similar physical characteristics, color, size, and shape. The purpose of the research was to find out the performance of using different ResNet architectures by examining the accuracy using F1-score value. Five dog classes from the sporting group were taken from the Tsinghua Dogs Dataset which contains 9558 images and were resized to a resolution of 224 x 224 pixels. Low- and high-resolution images were provided by this dataset. The images were then split randomly into 3 aspects which are training, validation, and testing with ratios of 70%, 10% and 20% respectively. The ResNet50 and ResNet 101 will be used as the pretrained models in the research to test out which is the better model. The difference between ResNet50 and ResNet101 is one has 50 neural network layers and another one has 101 neural network layers. Both are using convolutional layers in the variants while dropout, pooling layers, and dense were added. The

model was run on 25 epochs, batch-size of 32 and 4 neural network layers with 0.3 rate of two dropout layers, 2 dense layers. Based on the training results, the validation loss showed by the Resnet101 has less variation compared with ResNet50. Next, to test the F1-score, the highest validation accuracy from both variants based on the training results were chosen. The testing result shows most of the image classification was accurate. However, the results show that ResNet101 performed slightly better than ResNet50 with a superiority of 0.2. Moreover, the research also indicates that the number of layers is a key factor in determining the results. It does not necessarily mean that more layers are better; sometimes, more layers do not make a significant difference but can lead to longer training times.

III. MATERIALS

A. Dataset

The image dataset is provided in GitHub by the author where the folder was imported and unzipped. The dataset contains a total of 8351 dog images split into training, validation, and testing. The total combined number of training and validation dataset is 7,515 images split into a ratio of 89: 11 respectively. The images are re-sized to a resolution of 256 x 256 pixels and then are cropped into a size of 224 x 224 pixels. The dataset contains a total of 133 unique outputs or dog classifications.

B. Hardware Specification

The platform used to run the code was Google Colab as it is a widely popular platform that performs parallel processing and is utilized for machine learning and neural network training. The system hardware specifications used in Google Colab are provided in Table I.

TABLE I. SYSTEM HARDWARE SPECIFICATIONS

Hardware	Description
CPU	Intel Xeon 2-core, 2.2 GHz
RAM	13 GB
GPU	NVIDIA Tesla K80 with 24 GB of GDDR5 memory (12 GB per GPU)
Storage	64 GB

The model imports torch and NumPy as two of the most important libraries used for this research as they are widely used for mathematical and scientific purposes, especially in the domain of machine learning and neural network training.

IV. PROPOSED METHOD

The research aims to survey the performances of different activation functions and optimization functions with various learning rates. This gives an intuition as to which parameters are ideal for the given problem. While the dataset size can be considered relatively small, the research provides substantial intuition at the implementation of high-performance activation functions and optimization functions with respect to each other.

A. Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep learning algorithm and a subset of Artificial Neural Network (ANN) that classifies an image by implementing convolutions, down sampling, and other pre-processing procedures. CNN takes the input image, allocates the appropriate weights and biases for classification (Saha, 2018). CNNs are discerned from other neural networks due to their prevailing performance with image and audio inputs (IBM, n.d). Every CNN contains 3 types of layers: Convolutional, Pooling and Fully Connected (IBM, n.d). For each layer, the CNN expands in its complexity, thus navigating bigger sections of an image. CNN is modified in a format such that it begins identifying small patterns into classifying the required object. For each layer, the CNN expands in its complexity, thus navigating bigger sections of an image. CNN is modified in a format such that it begins identifying small patterns into classifying the required object (Abdo et al., 2022).

Recent developments in Convolutional Neural Networks combine advancements from the deep learning field. They have been a common choice for image recognition tasks, such as handwritten numeric recognition. It is a popular discourse that CNNs were the first to provide a robust implementation which was successful using multilayer hierarchical networks. CNNs can reduce the number of trainable network parameters while improving the efficiency of backpropagation. (Seng et al., 2021).

For our project, CNN will demonstrate the solution to the problem by segmenting dog images based on the RGB color planes (Saha, 2018). The dataset is first designated to the convolution and pooling layers for feature extraction whilst the fully connected layer draws the extracted features onto the final output. The convolutional layers demonstrate vital importance in the CNN to convolute the images using kernels and matrix multiplication to extract high level features from an image whereas the pooling layers are utilized as down sampling computations, therefore reducing the processing power necessitated (Yamashita. R et al., 2018). Pooling and convolution calculations are continuously repeated where the output of one layer dispenses as an input for the next layer as the network becomes more complex (Yamashita. R et al., 2018). In this research, a pre-defined model is implemented with its own weights and biases already set for classification of dog breeds.

B. Artificial Neural Networks

An Artificial Neural Network (ANN) consists of 3 layers: input layer, hidden layer(s), and output layer. Each layer is made up of neurons (nodes). The input layer is where data is fed into the network, transformed into a format that can be processed by the subsequent layers, and then passed to the first hidden layer. Each node in the input layer represents a feature of the input data (Raj, n.d). In the case of dog breed identification for instance, features include shape of a dog's face, texture of the fur, color patterns, presence of certain markings, etc.

The hidden layer(s) are where the heavy computations will be performed to solve the target problem. The number of hidden layers depend on the architecture of the ANN that is being used, and this number can have a significant impact on the model's performance. Generally, more complex problems require more neurons for processing and hence more layers. Each neuron in the hidden layer(s) has an activation function, which is used to determine the output of a neuron (i.e., whether to fire a neuron or not). Following are the steps that are usually involved in computing the output of a neuron (forward propagation):

1. The computation starts off at the input layer of the network and progressively propagates towards the output layer. To propagate forward, every neuron in the same layer will receive inputs from the neurons in the preceding layer to calculate the *weighted sum of inputs*.
2. The weighted sum is then passed into the activation function. If the value exceeds a certain threshold, the neutron will be activated. Otherwise, the neutron will be inactive.
3. Outputs from the activated neurons will be passed to the neurons in the succeeding layer. On the other hand, an inactive neuron will have output equal to 0 since it is being "turned off". Therefore, inactive neurons have no impact on the final prediction of the model.

During model training, however, inactive neurons will still play a role in the network. Loss or error is calculated at the output layer and is then passed back to the hidden layers to update the model's weights and biases. This process is also known as backpropagation (Seth, 2021).

In this project, the Rectified Linear Function (ReLU) has been used as the activation function for neurons in the hidden layers. ReLU is a piecewise linear function that will output a neuron's weighted sum directly if the value is positive, and outputs 0 otherwise (Brownlee, 2020).

The output layer is the layer that produces the result of the ANN. The number of nodes in the output layer depends on the number of output variables (or classes). For instance, the dog breed identification problem has 133 different types of dog breeds, that is, 133 output variables. Hence, the ANN may predict a dog's breed among these 133 possible classes. Depending on the type of problem to be solved, an activation function in the output layer may be necessary (Raj, n.d). In this project, the Logarithm SoftMax (Log SoftMax) activation function is applied to the output layer since it is typically used for multi-class classification problems. SoftMax will first take a vector of real numbers from the outputs of the hidden layers. These numbers will then be converted into a probability distribution over the output variables, such that each number in the vector will be within the interval of [0,1] and the sum of all numbers will be 1.

This way, the numbers can now be interpreted as the probabilities of the classes. For example, a probability distribution of [0.34201, 0.02413, 0.09031, …] means that there is a 34.201% chance of the dog being a Chihuahua breed, 2.413% chance of being a Papillon breed, 9.031% chance of being a Blenheim Spaniel breed, and so on. The purpose of Log is simply to achieve numeric stability by representing the SoftMax probabilities on a logarithmic scale (Abhirami, 2021).

C. Vanishing Gradient Problem

In the backpropagation process of the network and as the number of layers in the network increases, the rate of the product of derivative decreases towards a point where the partial derivative of the loss function merges towards zero, thus, the partial derivative vanishes (Jacob T., 2022). This means that the network is unable to learn effectively or at all. This problem is common in activation functions such as Sigmoid.

D. Dying ReLU Problem

The dying ReLU problem is a situation where numerous ReLU neurons that are in the scope of negative values output 0 when activating (Leung, 2021). This issue becomes very strenuous when most of the inputs to the ReLU neurons are negative. Furthermore, this makes the entire network inoperable when every node is a negative, by breaking down the gradient during backpropagation and hence, failing to update weights (Leung, 2021). The dying ReLU problem is caused by a high learning rate and/or a high bias (Leung, 2021).

E. DenseNet-161

Transfer learning is a technique that uses a pre-trained model and fine-tunes it for a new problem. In this project, a pre-trained CNN model with the DenseNet-161 architecture has been used to solve the dog breed identification problem. As the name suggests, DesnseNet-161 has 161 layers, which makes it a very deep neural network. The key idea behind DenseNet is to connect every layer to every other layer in a feed-forward fashion (Rao, 2020).

DenseNet is generally better than other deep neural networks architectures like AlexNet, VGG, GoogLeNet, MobileNet and SqueezeNet. Firstly, DenseNet has more diversified features because every layer receives input from all preceding layers. This introduces data with more variety into the network, which can enhance the network's performance. On the contrary, network architectures like AlexNet and VGG have very limited feature reuse because the connections between layers are relatively simple, such that a layer is only connected to its immediate layers. Besides, DenseNets has a strong gradient flow. Dense connections among layers help to preserve the gradients during backpropagation, enabling the network to have a faster and more stable learning process (Tsang, 2018). Meanwhile, other network architectures except for ResNet, DenseNet, LSTM, and Highway Networks are all susceptible to the vanishing gradient problem, which can impede model learning (Rao, 2020).

F. Activation Functions

1) Rectified Linear Unit (ReLU)

ReLU is a simple mathematical function that acts as the identity function for all positive inputs and equals zero for all negative inputs (Banerjee et al., 2019) as seen by equation (1). It has been shown to decrease training time and increase accuracy compared to other common activation functions like the Sigmoid activation function, and Tanh function. ReLU also has several variations such as Leaky Rectified Linear Unity (Leaky ReLU) and Exponential Linear Unit (ELU) (Banerjee et al., 2019).

$$ReLU(x) = \begin{cases} \max(0, x) & , x >= 0 \\ 0 & , x < 0 \end{cases} \quad (1)$$

2) Exponential Linear Unit (ELU) and Leaky Rectified Linear Unit (Leaky ReLU)

The Exponential Linear Unit (ELU) and Leaky Rectified Linear Unit (LeakyReLU) are both activation functions that are subsets of ReLU and designed to solve the dying ReLU problem. ELU, however, is slower than LeakyReLU to its non-linearity (Singh S., n.d). Both LeakyReLU and ELU work on evading the vanishing gradient problem using the

same activation techniques as ReLU for positive values (Clevert et al., 2016). However, ELU and LeakyReLU include negative values as well that enables the mean of the ELU activation function move towards 0. This enables the training model to coincide quicker than other activation functions (Singh S., n.d). The main difference between ELU and LeakyReLU activation function is that ELU uses an exponential slope as seen by equation (2) while LeakyReLU uses a linear slope as seen by equation (3) for negative input values (Liu, 2017).

$$ELU(x,\alpha) = \begin{cases} x & ,x >= 0 \\ \alpha(e^x - 1) & ,x < 0 \end{cases} \quad (2)$$

$$LeakyReLU(x,\alpha) = \begin{cases} \text{x} & ,x >= 0 \\ \alpha x & ,x < 0 \end{cases} \quad (3)$$

G. Optimization Functions

1) Stochastic Gradient Descent (SGD)

The first modification that has been done is by changing the optimization function to **Stochastic Gradient Descent (SGD)**. One of the benefits of having SGD as the optimizer is that the model can be trained much faster. This is because SGD only uses a small subset of the training data to update the model parameters at every iteration. This subset is also known as a *mini batch*.

Besides that, SGD has smaller memory requirements because it only needs to store the gradients for a single mini-batch data. On the other hand, Adam optimizer needs to store additional quantities such as the first and second moments of gradients. Sometimes, SGD can provide better generalizations compared to Adam optimizer because it is more prone to finding flat minima in the optimization landscape, making it less likely to overfit the training data. It is also easier to tune the hyperparameters in SGD than in Adam optimizer. In SGD, the only hyperparameter that is involved is the learning rate, but Adam optimizer needs all learning rates beta1, beta2, and epsilon.

2) Adaptive Moment Estimation (Adam)

Adam is a method for efficient stochastic optimization that only requires first-order gradients with relatively lower memory requirements. It computes individual adaptive learning rates for different parameters based on the estimate of the first and second moment of the gradients. This method aims to combine the benefits of the 'AdaGrad' method (Duchi et al, 2011) and RMSprop (Tieleman & Hinton, 2012). The AdaGrad function works well with sparse gradients, whereas RMSprop works well in non-stationary settings. As stated by the original paper, some of Adam's advantages include: the magnitudes of parameter updates are invariant to rescaling of the gradient, its step sizes are approximately bounded by the step size hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing. (Kingma & Ba, 2015).

3) Adaptive Gradient Algorithm (Adagrad)

Adagrad is an optimization algorithm that adjusts the learning rate for each parameter based on gradient history. This is accomplished by dividing the current gradient in the update rule by the sum of the previous gradients. As a result, when the gradient is large, the learning rate is reduced, and when it is small, the learning rate is increased. Adagrad can speed up the learning process for sparse datasets because it updates parameters associated with infrequent features more quickly (Mayanglambam, 2020). It also eliminates the need for manual learning rate tuning. However, one drawback is that the learning rate can become too small over time, resulting in slow convergence (Databricks, n.d.).

While Adagrad and Adam are better suited for complex and sparse datasets, SGD is often more suitable for larger datasets. This is because SGD is computationally more efficient and requires less memory than Adagrad and Adam. Adagrad and Adam are more adaptive and can handle different learning rates for each parameter, while SGD requires a carefully chosen learning rate that works well for all parameters as it needs to set the learning rate manually.

V. Discussion On Implementation

The model implementation seeks to identify how DenseNet-161 works with the Adaptive Moment Estimation (Adam), Stochastic Gradient Descent (SGD) and Adaptive Gradient Algorithm (Adagrad) with respect to different learning rates, along with how extensions of Rectified Linear Unit (ReLU) activation function such as Exponential Linear Unit (ELU) and Leaky ReLU can be implemented with respect to different learning rates.

VI. Results

A. Result (Optimization Function)

In Fig 1 and 2., at learning rate 0.001, the default Adam optimizer provides the best performance, reaching 85% validation accuracy and 75% training accuracy. We also see that Adagrad converges faster than its counterpart SGD towards the performance achieved by Adam for validation. Furthermore, Adam achieves the lowest loss both in training and validation, followed by AdaGrad, which is then followed by SGD.

As for Fig 3 and 4., with a learning rate of 0.01, Adagrad provides the best results among all three optimizers that were experimented with, as it has the highest accuracies (>70% in training, >80% in validation) as well as the lowest loss scores for both training and validation datasets. It can be observed that SGD has very sharp increase in both training and validation accuracies and very sharp decrease in training and validation loss.

The accuracies and losses for training and validation on Adam optimizer are stable even as the number of epochs increases. This could be explained by 2 reasons:

1. The model has reached its optimal performance. In other words, the model has already learned everything that it can learn from the data, and there is nothing to improve further (To make further progress, more data may be required, or a different model architecture could improve results).
2. Since the validation accuracy is higher than the training accuracy and that both accuracies are quite stable, it may suggest that the model might be overfitting the data. The model is perhaps too complex that it starts learning the noise in the training data.

As compared to Adam optimizer, SGD has a higher rate of change in both accuracy and loss scores.

In Fig 5 and 6., the training and validation accuracies of AdaGrad starts off low but improves over number of epochs towards 60% for training and 76% for validation which is the highest compared to the SGD and Adam, this could be due to AdaGrad's adaptive learning rate mechanism, which adjusts the learning rate for each parameter based on its historical gradients, allowing it to converge faster and more accurately in this case.

SGD showed a consistent improvement in training accuracy over epochs, and this could be due to its use of a fixed learning rate for all parameters, which can make it less effective when the gradients for different parameters have very different magnitudes. The default Adam optimizer seems to have a lowest training and validation accuracy and highest training and validation loss overall. Thus, Adam does not perform well in steep learning rates.

The performance of optimizer functions can be sensitive to the choice of hyperparameters. With Adam optimizer, if the learning rate is too high, the optimizer overshoots the optimal weight and oscillates around the minimum, resulting in poor training and validation accuracy. On the other hand, if the learning rate is too small, it could result in slow convergence. It's important to choose a learning rate in the range of 0.001 or lower. For AdaGrad, its ability to adjust the learning rate for each parameter based on its historical gradient information could help it converge faster and avoid overshooting the optimal solution (Pramoditha, 2022).

B. Results (Activation Functions)

In Fig 7 and 8., when the learning rate is set to 0.001, compared to the other functions, the Leaky ReLU function begins from a lower loss value immediately from the first epoch. It remains relatively non-variant, while also providing the lowest training loss. Default ReLU and the ELU on the other hand begin with training loss values of approximately 2.5 and 3.5 in training respectively and stabilize to nearly one towards the final epochs. The training accuracy with Leaky ReLU also remains relatively non-variant, rising from ~75% to ~80% for training.

The other two functions reach a maximum training accuracy around ~75%. Comparing validation loss, all three functions converge to ~0.4. Like the previous case, the Leaky ReLU starts with the lowest loss from the first epoch and remains largely invariant. The default ReLU is the slowest to stabilize, but nonetheless reaches equivalent accuracy. All three functions perform roughly the same when comparing validation accuracy.

In Fig 9 and 10., with the learning rate increases to 0.01, ELU achieves higher training and validation loss. While the default ReLU converges to the lowest training loss. In this case, Leaky ReLU displays some fluctuations deviating towards inferior performance towards the end of the iteration in training and validation. The result indicates that ELU is more sensitive to the learning rate. It is worth noting that ELU provides the highest training accuracy despite achieving higher training and validation loss. The default ReLU conversely does not outperform the default ReLU and ELU activation functions despite having the lowest training loss.

In Fig 11 and 12., by setting the learning rate to an adverse value of 0.1, the behavior of the functions can be seen on a poorer scale for training and validation loss being at excessive values. Leaky ReLU, however, manages to gain a significant upward trend in accuracy despite the steep learning rate, reaching 50% and 60% in training and validation accuracy, respectively. The default ReLU is rendered completely ineffective at this learning rate for both training and validation accuracies.

Hence, the comparison of the three activation functions suggests that Leaky ReLU provides the highest performance across a range of learning rates.

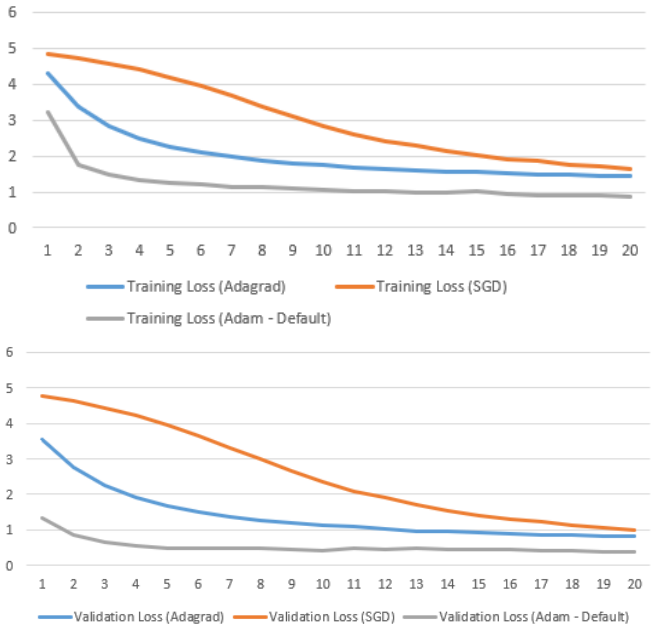

Fig. 1. (Results – Training and Validation Loss for different optimization functions of lr 0.001)

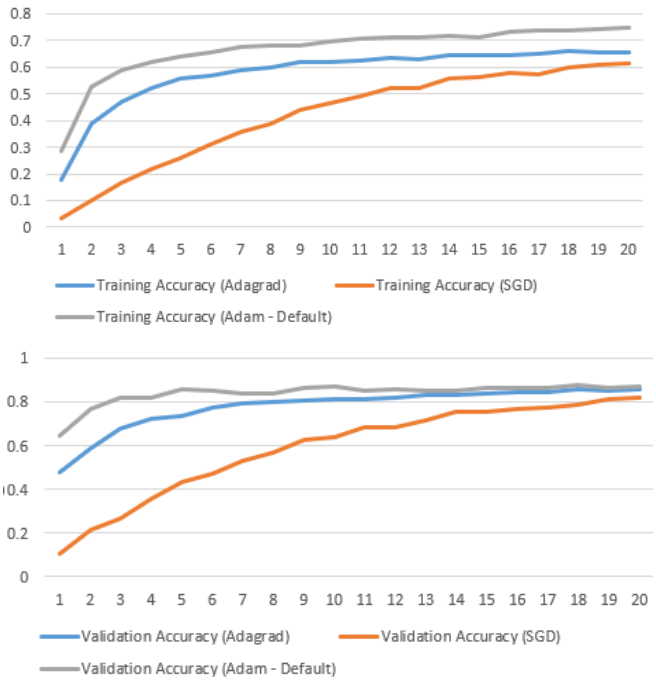

Fig. 2. (Results – Training and Validation Accuracy for different optimization functions of lr 0.001)

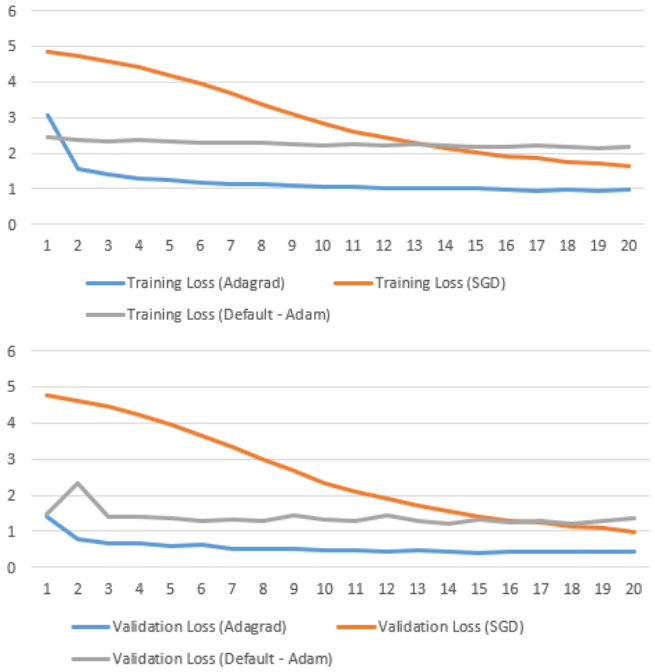

Fig. 3. (Results – Training and Validation Loss for different optimization functions of lr 0.01)

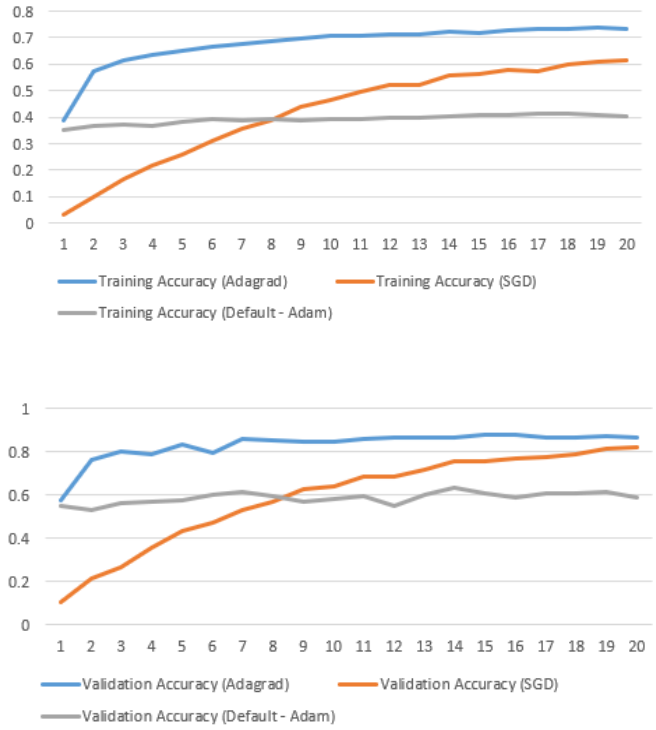

Fig. 4. (Results – Training and Validation Accuracy for different optimization functions of lr 0.01)

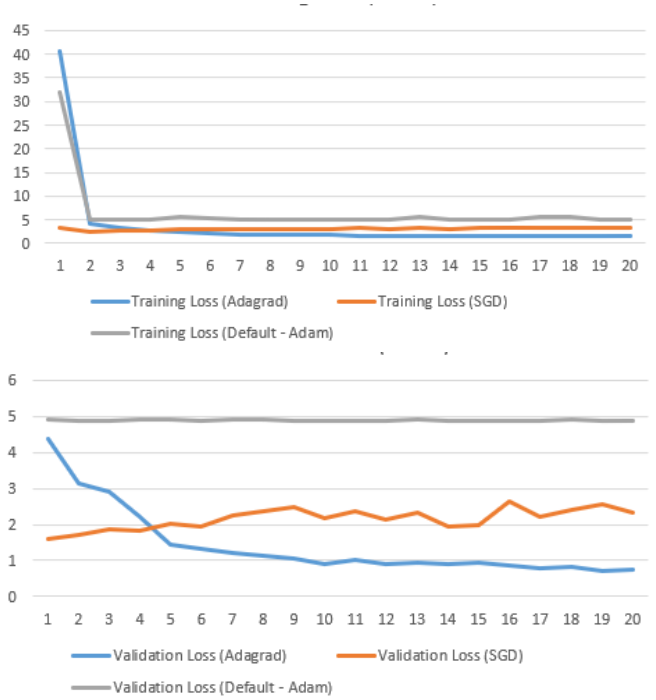

Fig. 5. (Results – Training and Validation Loss for different optimization functions of lr 0.1)

Fig. 6. (Results – Training and Validation Accuracy for different optimization functions of lr 0.1)

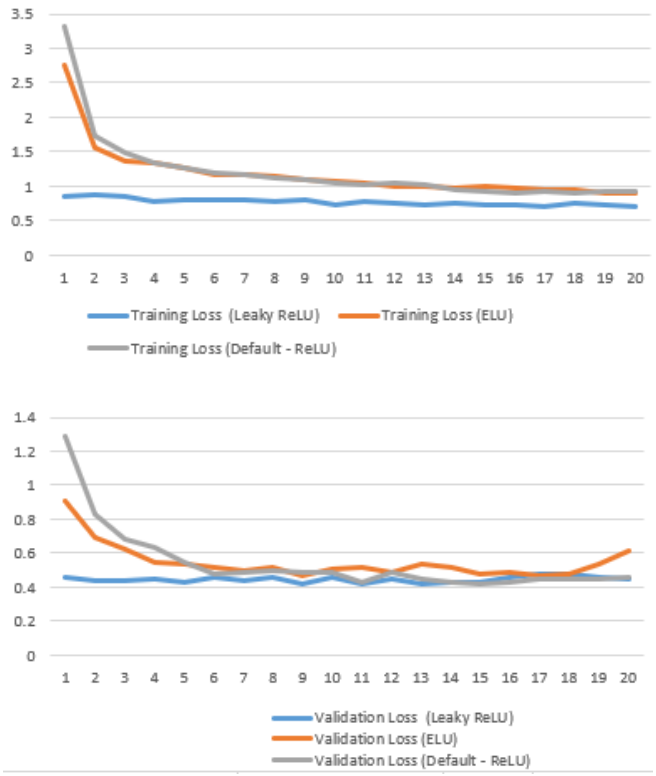

Fig. 7. (Results – Training and Validation Loss for different activation functions of lr 0.001)

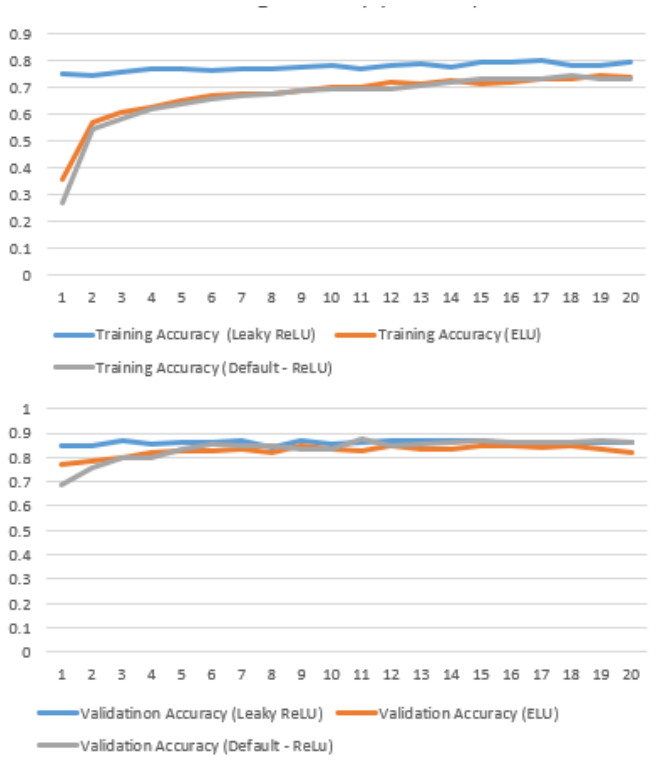

Fig. 8. (Results – Training and Validation Accuracy for different activation functions of lr 0.001)

Fig. 9. (Results – Training and Validation Loss for different activation functions of lr 0.01)

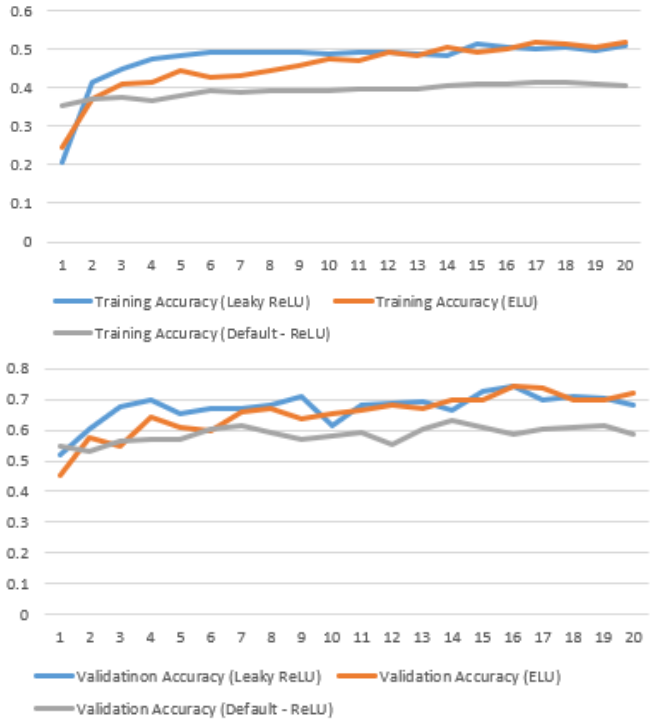

Fig. 10. (Results – Training and Validation Accuracy for different activation functions of lr 0.01)

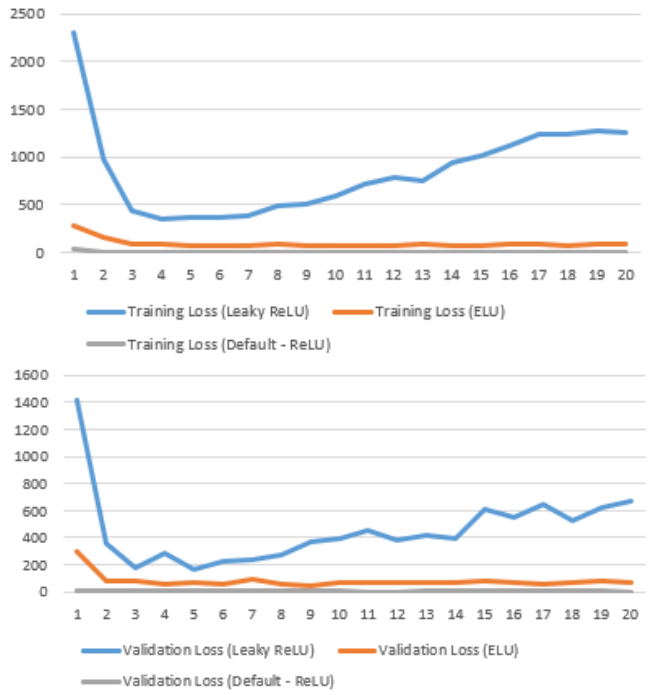

Fig. 11. (Results – Training and Validation Loss for different activation functions of lr 0.1)

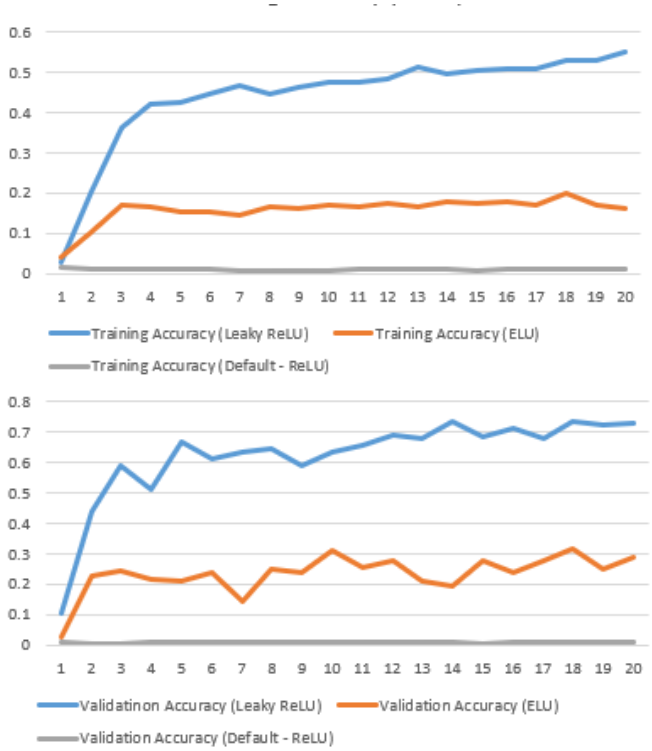

Fig. 12. (Results – Training and Validation Accuracy for different activation functions of lr 0.1)

VII. Conclusion

The key findings of this research proposes that the default optimizer Adam is more sensitive to higher learning rates than Adagrad and SGD. Furthermore, Adagrad and SGD both demonstrated relatively good results despite higher learning rate and relatively small dataset size as well. Adam nevertheless showed slightly superior performance when using the default learning rate of 0.001. Thus, we can conclude that Adam is only effective when lower learning rates are implemented, while Adagrad is ideal for a wide range of learning rates within the DenseNet architecture. As for the activation functions, we saw multiple flaws with the ReLU activation function when the learning rate increased. This exposed the Dying ReLU problem which caused neurons to be non-active because of large negative inputs. LeakyReLU on the other hand, showed superb performance despite the changes in learning rates. While ELU also showed relatively high performance, the accuracies of LeakyReLU signified that it is ideal for use with the DenseNet architecture and large output classification problem.

VIII. Limitations

The number of training and validation examples presented with respect to the classification output are considered to be relatively small for the given problem. The model could be improvised by using a larger dataset such as ImageNet or implementing data augmentation and fine tuning to increase dataset size and prevent overfitting. It is also worth noting that CNNs are susceptible to the noise in the images of the training dataset (Lau, et al., 2021). Furthermore, recent advancements in GPU hardware using dedicated accelerators for machine learning has facilitated the development of faster ML libraries. The training and inference performance could be improved significantly by using hardware acceleration using libraries like Nvidia TensorRT, rather than the general purpose CUDA GPU architecture used in this model.

Acknowledgment

We would like to thank the pratyakshajha on GitHub for creating the model and supplying the dataset needed for this research to be made.

References

Abdo, N. A., Yusop, R. B., & Abdulla, R. (2022). *Obstacle Avoidance Robot Using Convolutional Neural Network*. Journal of Applied Technology and Innovation, 6(4), 27-33. https://dif7uuh3zqcps.cloudfront.net/wp-content/uploads/sites/11/2022/09/19105551/Volume6_Issue4_Paper6_2022.pdf

Abhirami, V. S. (2021, October 10). *Softmax vs LogSoftmax*. Medium. https://medium.com/@AbhiramiVS/softmax-vs-logsoftmax-eb94254445a2.

Banerjee, C., Mukherjee, T., & Pasiliao, E. L. (2019). An Empirical Study on Generalizations of the ReLU Activation Function. ACM Southeast Regional Conference. https://doi.org/10.1145/3299815.3314450.

Brownlee, J. (2020, August 20). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Machine Learning Mastery. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

Castillo, D. (2023, February 21). Transfer Learning for Machine Learning. Seldon. https://www.seldon.io/transfer-learning#:~:text=Transfer%20learning%20is%20generally%20used,categorisation%20or%20natural%20language%20processing.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2016). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *ArXiv:1511.07289 [Cs]*, *5*(arXiv:1511.07289). https://arxiv.org/abs/1511.07289v5.

Databricks. (n.d.). *What is AdaGrad?* Databricks. https://www.databricks.com/glossary/adagrad.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, *12*. https://dl.acm.org/doi/10.5555/1953048.2021068.

i2tutorials. (2019, September 9). *What is the difference between Adagrad, Adadelta and Adam?* I2tutorials. https://www.i2tutorials.com/what-is-the-difference-between-adagrad-adadelta-and-adam/.

Jacob, T. (2022, February 25). *Vanishing Gradient Problem, Explained*. KDnuggets. https://www.kdnuggets.com/2022/02/vanishing-gradient-problem.html#:~:text=When%20there%20are%20more%20layers .

Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. ArXiv (Cornell University). https://doi.org/10.48550/arxiv.1412.6980.

Koehrsen, W. (2021, December 6). Transfer Learning with Convolutional Neural Networks in PyTorch. Medium. Retrieved March 8, 2023, from https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce#:~:text=Most%20categories%20only%20have%2050,trained%20model%20applying%20transfer%20learning.

Lau, Y., Sim, W., Chew, K., & Ng, Y. (2021). Understanding how noise affects the accuracy of CNN image classification. *Journal of Applied Technology and Innovation (JATI)*, *5*(2), 2600–7304. #84-JATI-Understanding How Noise Affects The Accuracy of CNN Image Classification (dif7uuh3zqcps.cloudfront.net)

Leung, K. (2021, July 27). *The Dying ReLU Problem, Clearly Explained*. Medium. https://towardsdatascience.com/the-dying-relu-problem-clearly-explained-42d0c54e0d24.

Liu, D. (2017, November 30). A Practical Guide to ReLU - Danqing Liu - Medium. Medium. Retrieved March 15, 2023, from https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7.

Long, K., Nataraj, C., & Susiapan, Y. (2022). Autonomous Garbage-Collecting Robot For Beaches With Deep Learning Approach and Improved Cleaning Technique handwritten digit recognition with different CNN architectures. Journal of Applied Technology and Innovation, 5(1), 2600–7304. *Journal of Applied Technology and Innovation*, *6*(2), 2600-7304. https://dif7uuh3zqcps.cloudfront.net/wp-content/uploads/sites/11/2022/03/06191655/Volume6_Issue2_Paper1_2022.pdf

Mayanglambam, G. (2020, November 18). *Deep Learning Optimizers*. Medium. https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f.

Oliver. J. (2023, February 9). How Many Dog Breeds Are There in the World? (2023 Update). Pet Keen. Retrieved March 8, 2023, from https://petkeen.com/how-many-dog-breeds-in-the-world/.

Pramoditha, R. (2022, October 6). *How to Choose the Optimal Learning Rate for Neural Networks*. Medium. https://towardsdatascience.com/how-to-choose-the-optimal-learning-rate-for-neural-networks-362111c5c783#:~:text=Code%20by%20author)-.

Pratama, N. H., Rachmawati, E., & Kosala, G. (2022, November). Classification Of Dog Breeds From Sporting Groups Using Convolutional Neural Network.Researchgate.net. https://www.researchgate.net/publication/368822454_classification_of_dog_breeds_from_sporting_groups_using_convolutional_neural_network.

Raj, R. (n.d.). *Components of an Artificial Neural Network*. Enjoy Algorithms. https://www.enjoyalgorithms.com/blog/components-of-ann.

Rakhecha, A. (2019, July 7). *Understanding Learning Rate*. Medium. https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de#:~:text=If%20the%20learning%20rate%20is.

Rao, S. S. (2020, May 9). *Exploring DenseNets and a comparison with other Deep Architectures*. Medium. https://medium.com/analytics-vidhya/exploring-densenets-and-a-comparison-with-other-deep-architectures-85f02597400a.

Saha, S. (2018, December 16). *A comprehensive guide to Convolutional Neural Networks-the eli5 way*. Medium. Retrieved March 7, 2023, from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

Seng, M., Chen Chiang, B. B., Abdul Salam, Z. A., Yih Tan, G., & Tong Chai, H. (2021). MNIST handwritten digit recognition with different CNN architectures. Journal of Applied Technology and Innovation, 5(1), 2600–7304. MNIST Handwritten digit recognition with different CNN architectures (dif7uuh3zqcps.cloudfront.net)

Seth, N. (2021, June 8). *How does Backward Propagation Work in Neural Networks?* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/how-does-backward-propagation-work-in-neural-networks/.

Sharma, P. (2021, October 30). Understanding Transfer Learning for Deep Learning. Analytics Vidhya. Retrieved March 8, 2023, from https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/.

Singh, S. (2020, June 5). *ELU as an Activation Function in Neural Networks* . Deep Learning University. https://deeplearninguniversity.com/elu-as-an-activation-function-in-neural-networks/.

Stanford Dogs Dataset. (2019, November 13). Kaggle. Retrieved March 8, 2023, from https://www.kaggle.com/datasets/jessicali9530/stanford-dogs-dataset.

Tsang, S.-H. (2018, November 25). *Review: DenseNet — Dense Convolutional Network (Image Classification)*. Medium. https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803.

Turing. (2022, May 28). *Softmax: Multiclass Neural Networks*. Turing. https://www.turing.com/kb/softmax-multiclass-neural-networks.

What are Convolutional Neural Networks? | IBM. (n.d.). Www.ibm.com. Retrieved March 7, 2023, from https://www.ibm.com/my-en/topics/convolutional-neural-networks.

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, *9*(4), 611–629. https://doi.org/10.1007/s13244-018-0639-9.