Student Name: Shashank B V

Student No: R00224414

Title: Building a Cat/Dog Image Classifier

Subject: Applied Machine Learning

Declaration of Authorship

I, SHASHANK B V, declare that the work presented in this assignment, titled *Data Science and Visualisation*, is my own. I confirm that:

- This work was done wholly by me as part of my MSc. in Data Science & Analytics.
- Where I have consulted the published work and source code of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this assignment source code is entirely my own work.

18-05-2023

On ___DATE___

Signature:

SHASHANK B V

INTRODUCTION

The focal point of this report entails developing a machine learning-based cat/dog image classifier that operates with remarkable accuracy. It presents a somewhat challenging undertaking since machine learning algorithms encounter difficulties in classifying animal images accurately - it comes naturally to humans! Our project's primary aim was simple yet daunting - to distinguish between two classes containing distinct yet closely related creatures.

We relied on Dog vs. Cats's extensive database consisting of about 25k photos of both felines and canines dealing with our binary classification problem intending to create an algorithm capable enough to identify new pictures accurately. We've grouped our analysis results into three major categories - Pre-processing Phase, Training Phase, and Optimization Phase - each serving as a fundamental section that contributes significantly concerning producing an efficient model.

During pre-processing phases (Section A), we've emphasized preparing our data sets correctly by implementing full-scale procedures such as loading data set images onto relevant files while normalizing, resizing, and reducing dimensionality through PCA to ensure uniformity and consistency of the data.

Achieving consistency in pixel values is critical to effective image processing through normalization techniques that ensure standardization across diverse images. By contrast, resizing technique guarantees equal size dimensions for all images processed to prevent distortion effectively. PCA enables essential features extraction from vast datasets that minimize information loss while at it.

In the training phase of machine learning models of pre-processed datasets includes splitting them into two different sets: training and development sets with an objective of establishing the most efficient solution for classification tasks using various models such as Support Vector Machines (SVM), Random Forests or Convolutional Neural Networks (CNN). Performance evaluation metrics like accuracy rate measurement alongside precision score determination are used to evaluate model efficiency effectively. The optimization phase aims at refining selected models' performance through adjusting hyperparameters accordingly.

Hyperparameter optimization plays a crucial role in maximizing performance for machine learning classifiers such as ours. Through techniques like grid search, random search, or Bayesian optimization we systematically explore the combinations of values within the hyperparameter space so that we can find what provides high accuracy while minimizing errors in identifying animals as either dogs or cats. The final result of these three phases will be a robust machine learning classifier capable of accurately classifying animals. For evaluation of our model we will use a test dataset containing previously unseen images and measure performance metrics to determine its generalization capabilities. Once optimized this model can then be used for quick and accurate animal identification.

Part – A: Pre – Processing Phase

In this pre-processing phase, several critical steps are followed to prepare the image data for further use in model training and testing.

1. Setting Constants:

This phase starts by defining key constants that will determine the specifications of the images and the dimensionality of the feature set. The image size is set, ensuring all images conform to the same dimensions, and the number of principal components is determined for later dimension reduction using PCA.

2. Loading and Pre-processing Images:

The images are then loaded from a specified folder. During this process, each image is read and pre-processed as follows:

Resizing: To ensure uniformity, each image is resized to match the predefined constant image size. This is important as images in datasets often come in various sizes, and most machine learning models require input data of a fixed size.

Normalization: Each pixel in the image is represented by three integers (ranging from 0 to 255) that make up the RGB value. By dividing each value by 255, the data is normalized to a range between 0 and 1. This helps in speeding up the computation and makes the optimization process faster.

For training images, labels are also assigned based on the image filename. A binary label is used to classify the images into two categories: cats and dogs.

3. Displaying Sample Images:

Based on the below output we can see that the images are loaded and pre-processed. And printed the head of the data.

4. Image Flattening:

Since PCA requires a 2D array as input, the images (3D arrays considering width, height, and color channels) are flattened into 1D arrays. Each entry in this array represents a pixel from the original image.

5. Dimensionality Reduction using PCA:

Finally, dimensionality reduction is performed using Principal Component Analysis (PCA). PCA identifies the axes in the feature space along which the data varies the most and projects the original data onto these new axes. The axes are ordered according to the amount of variance each holds.

PCA helps in reducing the complexity and size of our data while retaining its essential elements. This makes it easier to process the data, reduces the computational cost and helps in removing noise and redundancy from the dataset.

Part – B: Training Phase

In preparation for building and educating our machine learning models using our refined dataset we kick off with an important stage called "data splitting." At this stage we divide our pre-processed data into two parts.

Namely: A training set which will be utilized for model education; And A validation set that helps in finely tuning parameters/hyperparameters for optimal efficiency while preventing overfitting.

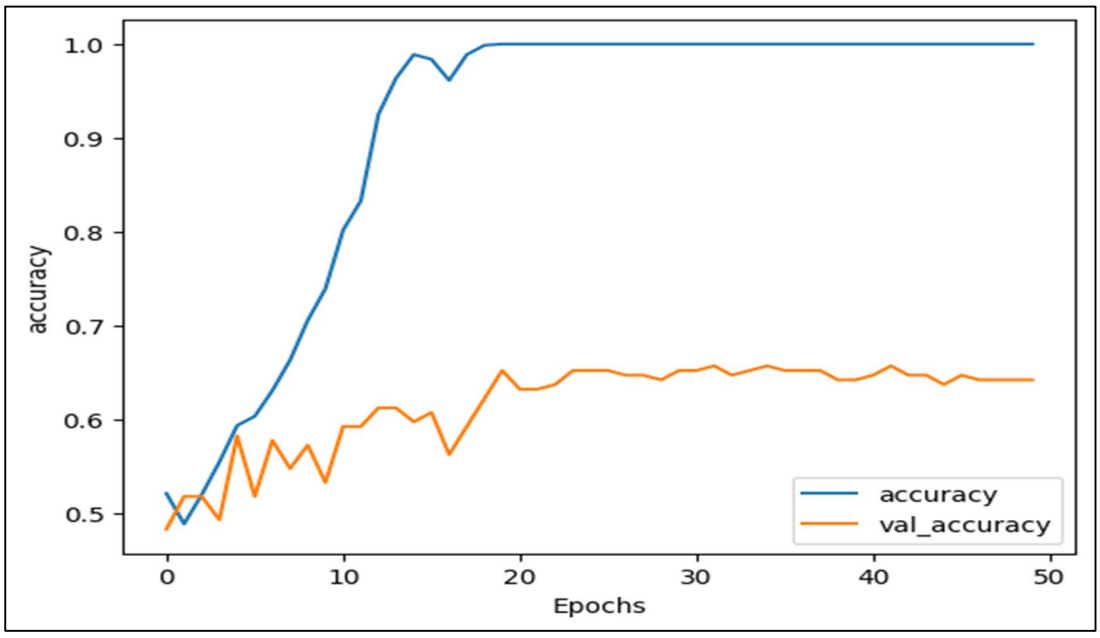
Once this step has been completed satisfactorily, we move on to "one hot encoding." Herein lies another vital part of our process as image labels undergo transformation into one hot encoded vector. This process becomes necessary because it allows for appropriate comparisons between predicted probabilities per class versus real values.

Finally comes the development/training phase involving three distinct Convolutional Neural Network (CNN) models, MobileNet Model, EfficientNetB0 Model:

Convolutional Neural Network (CNN):

Our very first model involves crafting a customized CNN consisting of four convolutional layers with each layer being succeeded by a max pooling layer before completing with dense layer after flattening out the convolutional layers' output. For this multi-class classification problem, we use a softmax activation function in the output layer. To tackle this task efficiently while taking into account mobile and embedded vision applications, Google developed MobileNet as a lightweight model architecture.

Model Accuracy and validation accuracy

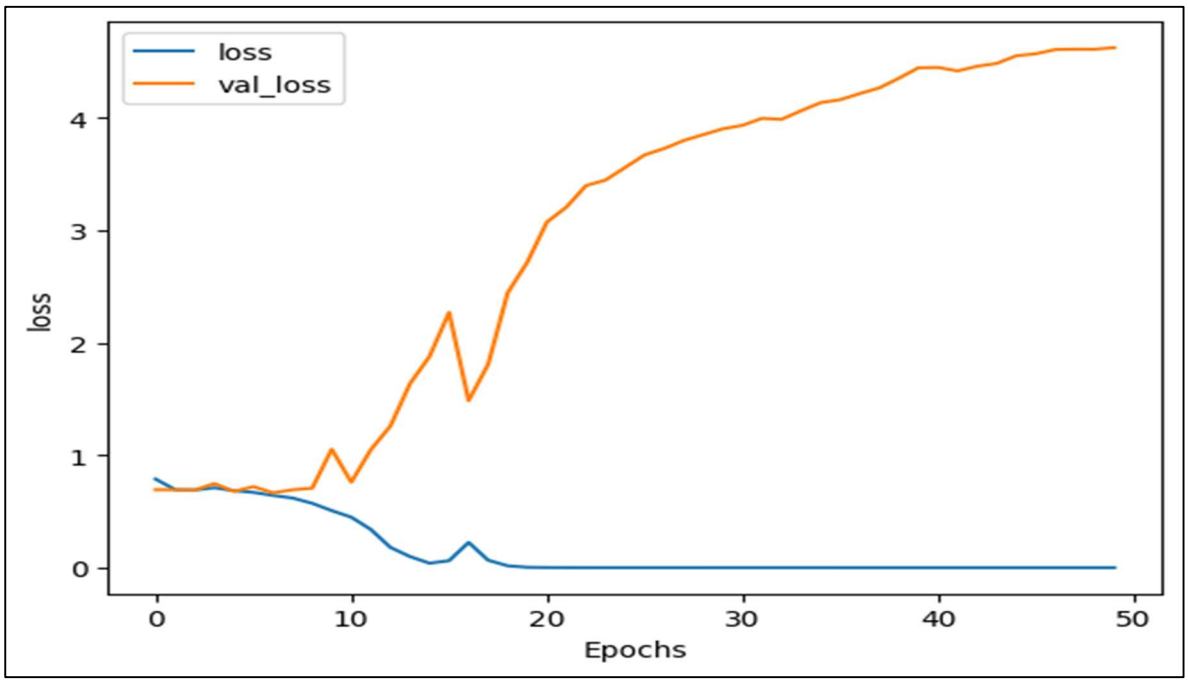


- In order to gauge how effectively our model was functioning we looked at its accuracy - which indicates how often it makes correct predictions - as a key

performance indicator. Generally speaking, models with higher accuracy rates perform more effectively than those with lower rates do. In this particular case study example, we noted that our training accuracy experienced significant gains throughout its lifecycle: starting at a modest rate of just 0.5206 during its first epoch and progressing upward until it reached an impressive perfection rate of 1 (or 100%) during its final epoch! These results clearly indicate that our model was capable of flawlessly classifying all instances present within its given training set.

- The Validation Accuracy refers to the precision rating calculated when testing a model using validation data. If a model was ideal, it should produce similar results for its performance using both training and validation sets. Unfortunately, in this instance, after an initial climb in accuracy, it plateaus before decreasing to 64.18% at the end of its training phase; this grounding out suggests overfitting.

Model Loss and validation Loss

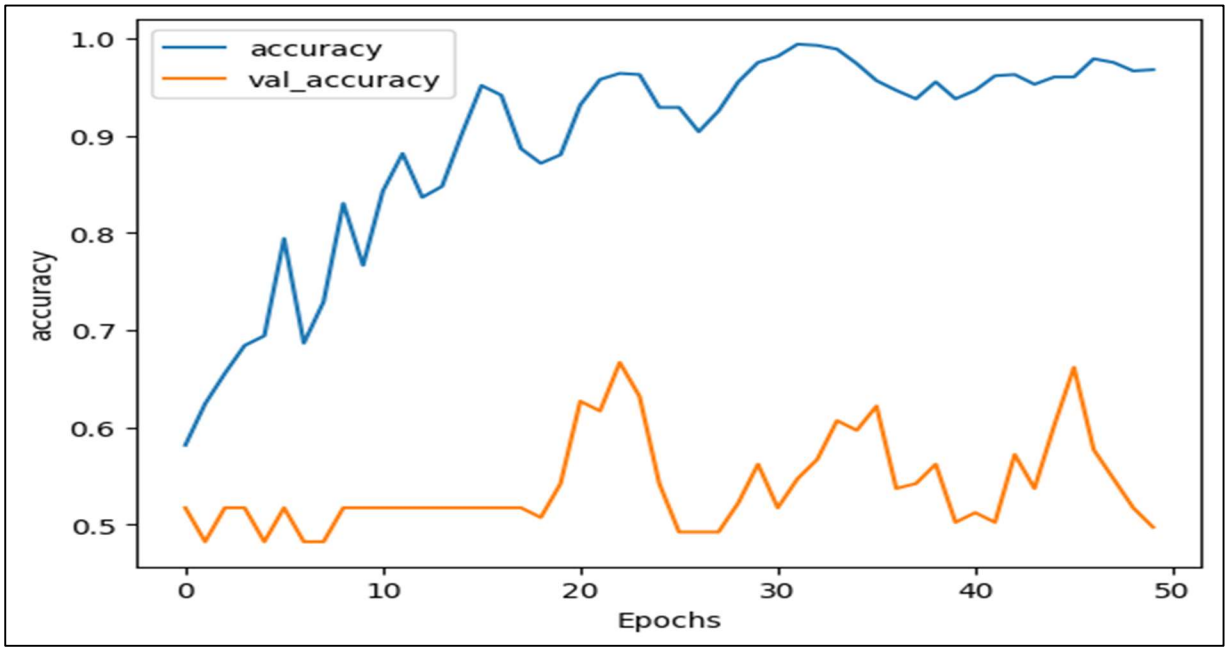


- To evaluate whether a neural network's predictions are close to actual values, analysts use "loss" as a key metric. Lower losses indicate more accurate models - and we want accuracy! In this case, we can see our training loss steadily dropping from its start point of 0.7876 in epoch one all the way down to nearly zero (1.2219e-06) by epoch fifty: evidence our system is doing well at learning from our dataset.
- To measure loss on the validation data -a set of data not used during training- we use validation loss. Our primary goal is to keep this value as low as possible. However, in this scenario, initially we witness a decrease in validation loss that starts increasing after some epoch until it reaches 4.6272 at epoch 50th. This points towards overfitting of model towards training data where good learning results are obtained but poor performance on new and unseen datasets

MobileNet Model:

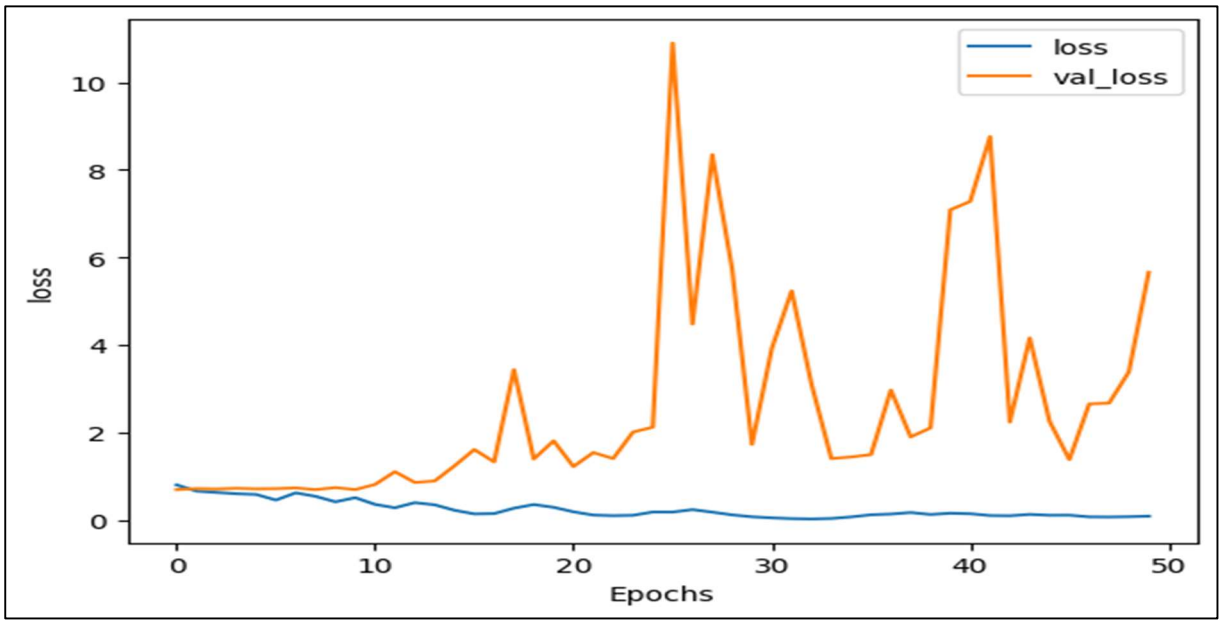
The model is configured with no pre-trained weights, an input shape for RGB images of size IMG_SIZE x IMG_SIZE, and two output classes. It's compiled with an Adam optimizer, a learning rate of 0.003, and Binary Cross Entropy loss function. The training, for 50 epochs, is performed on a dataset (X_train, y_train) with performance validation on a separate set (X_val, y_val).

Model Accuracy and validation accuracy



A trend towards overfitting was noted during our analysis of the MobileNet model after its successful completion of fifty epochs of training. Our findings disclosed that while training accuracy continued improving up until an outstanding score of 99.38% by epoch number thirty-two, validation accuracy failed to reflect this improvement pattern and peaked only at sixty-six point six-seven percent during epoch twenty-three before proceeding on an erratic path culminating in an abysmal low-end score of forty-nine point seven-five percent.

Model Loss and validation Loss



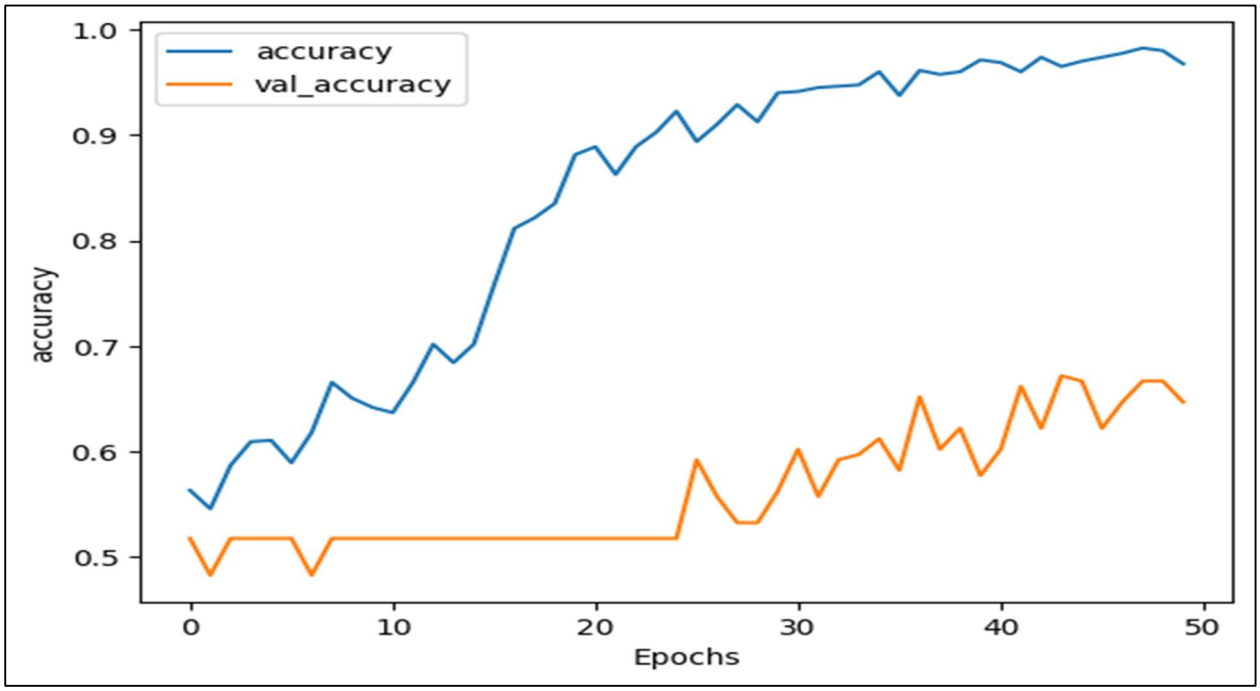
In a similar vein, the loss incurred by the training set keeps decreasing throughout the process and comes to as low as 0.0232 by epoch 33. However, the behavior of validation loss is not parallel to that of its counterpart. It varies greatly and shows a general upward trend, attaining its peak at epoch 26 with an amount of 10.8927 before eventually ending at 5.6587 during epoch 50.

The observed outcomes indicate that although we have succeeded in effectively teaching our model with available training data inputs; limitations reveal themselves when applied towards unfamiliar information presented within validation sets. Optimizing performance can be achieved by exploring alternate approaches such as regularizing techniques (e.g., dropout or L1/L2 regularization), implementing diverse forms of data augmentation methods, and considering using a less complex architecture design for models. Hyperparameters, such as the learning rate, should also be assessed for fine-tuning to facilitate the model's training process.

EfficientNetB0 Model:

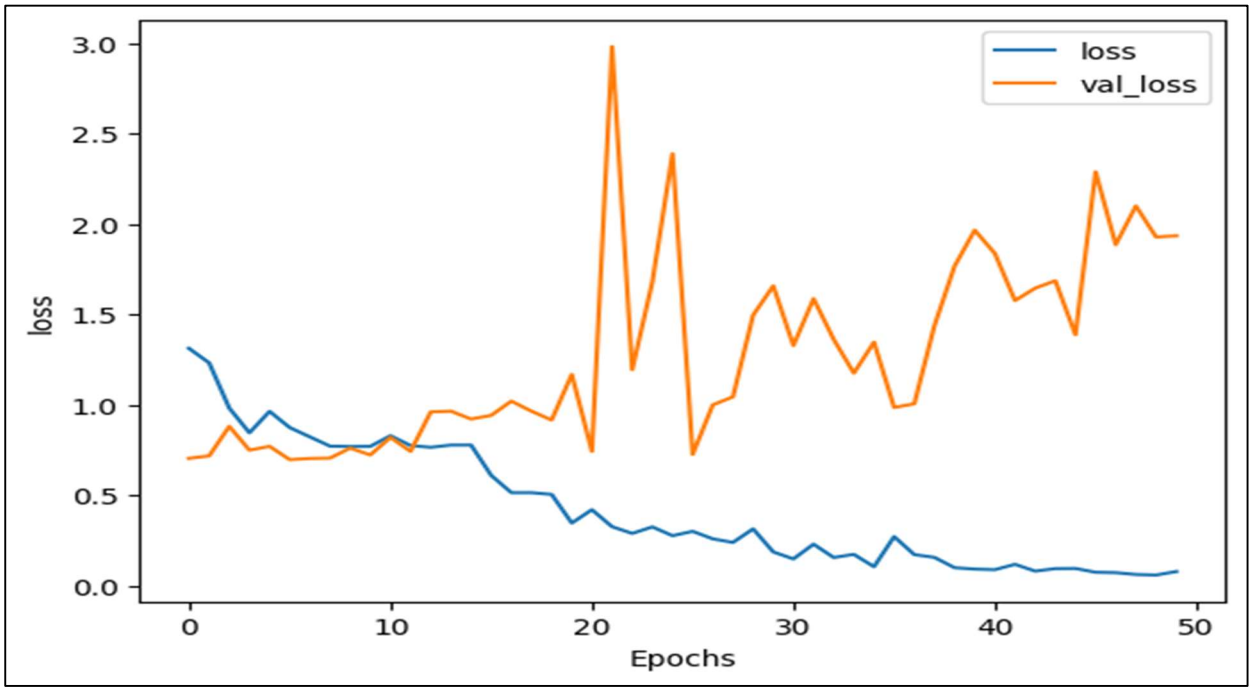
we employ the EfficientNetB0 model that lacks pre-trained weights. An Adam optimizer, set at a learning rate of 0.0003, is compiled alongside Binary Cross Entropy as our loss function. Afterward, the model undergoes training using the training data for a total of 50 epochs. To ensure accurate measurements and estimations of our model's performance, validation data is used for evaluation purposes.

Model Accuracy and validation accuracy



During its first epoch of training, this particular model demonstrated an initial accuracy rate of roughly 56.3%. However, over time and by its fiftieth epoch, it significantly improved to a successful rate of around 96.75% - evidence that it learned and adapted to patterns within the given data set effectively. Meanwhile, one important measure of performance is evaluating how well these predictions matched actual labels in said data set – represented by loss values- which generally decreased throughout its training progress; indicating progress towards greater accuracy.

Model Loss and validation Loss



For instance, we observed that our model's validation accuracy started at 51.74% during its initial phase and varied during training until settling at 64.68% by epoch 50. However, this observation implies that there are chances for overfitting since similar fluctuations suggest some bias towards prior knowledge gained through past experiences with familiar data patterns rather than genuinely unknown ones encountered herein. Additionally, it also indicates a lack of consistency when exposed to different datatypes due to fluctuations resulting from such biases. Furthermore, validation loss fluctuations indicate that our machine learning model is not generalizing well to new data. This situation occurs when the model focuses solely on its current dataset while ignoring outliers and noise present in such datasets that could negatively impact its quality and performance when exposed later to different sets of input data.

Part – C: Optimization Phase

As part of our pursuit of enhanced model performance, hyperparameter tuning was executed. Hyperparameters control various settings within models and need setting prior to actual learning.

Specifically, we sought optimization for four hyperparameters: learning rate, number of dense layer units, dropout rate and optimizer.

Our optimal solution entailed identifying a suitable 0.001-learning rate for our optimizer since it regulates how change occurs within models at each weight update based on estimated error values; selecting incorrect rates causes issues like long training times or suboptimal outputs or unstable processes resulting from overly-high rates_+._

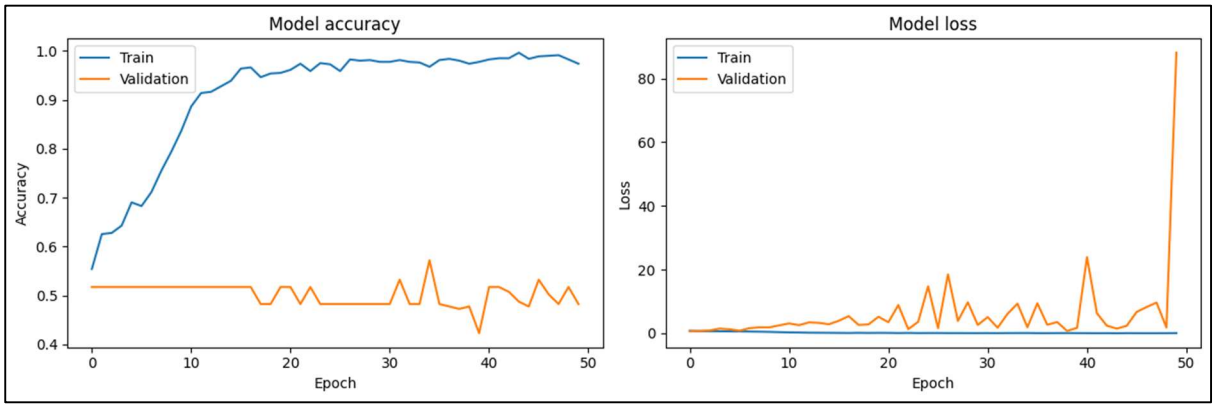
Similarly, we found that having 320 units generated ideal output space dimensionality within the dense layer. By increasing the number of neurons in a network, it may be possible for it to learn more complex representations; however, this comes with increased computational complexity and a higher risk of overfitting.

```
The optimal learning rate for the optimizer is 0.001
The optimal number of units in the dense layer is 320
The optimal dropout rate is 0.2
The optimal optimizer is adam
```

According to the study, 0.2 was deemed as the optimal dropout rate for reducing overfitting through regularization techniques such as dropout in neural networks. During training sessions, randomly selected neurons are "dropped out," meaning their contribution towards activating downstream neurons is temporarily removed during forward passes; therefore, no weight updates apply on backward passes involving those neurons. Additionally, "adam" was established as an ideal optimizer capable of making significant differences between achieving good results within minutes or dragging it for hours or even days.

Upon completion of hyperparameter tuning, our model went through a rigorous training process spanning over fifty different epochs wherein significant fluctuations were noted between various performance indicators used to evaluate success rates such as training vs testing error/mistakes made while processing data samples collected as part of this project effort.

Model accuracy and loss



The results indicate that while there were constant improvements in training performance metrics - such as accuracy scores- main issues persisted with regards to overfitting vis-a-vis our calibration data utilization practices which leads us down unexpected paths for achieving desired outcomes without properly accounting for factors likely responsible for poor results.

We noticed no significant improvement in terms of validity measures (accuracy) which remained around .5 on average for most epoques but remain increasingly concerned by consistent upward trends seen within testing losses throughout these same periods; by endpoint (50/50), final testing accuracy dropped from .5174 to .4826 while testing loss surged to 88.1037 - clearly highlighting overfitting's negative impact on data analysis and processing.

Nonetheless, our model displayed impressive consistency regarding its interpretation of training data set(s) - showing remarkable increases in accuracy scores throughout this exercise (maxing out at 0.9963 during epoch 44) - giving us the confidence that it is heading in the right direction and has a great potential in future research and applications alike. A continuous decrease in loss through each epoch is observed during training which suggests that learning and amelioration of performance occurs overtime on our model as it responds to input. On the other hand, comparison between trends from validation versus training sets projects a situation where there is high variance - meaning overfitting of our model only to suit its known dataset rather than adapting well when introduced to new ones.

Conclusion

Despite multiple optimization attempts, Model 3 is still plagued by overfitting issues. During training phases, the algorithm yields high accuracy rates of around 99%, while validation accuracy remains low and fluctuates between 48-57%. This pattern is indicative of a failure in adapting well on unseen data. Furthermore, as epochs progress validation loss increases steeply exposing many conceivable struggles in applying learning from training data while addressing new data queries.

Ultimately, we need other optimization strategies for Model 3's performance improvement. If your machine learning model runs into overfitting problems don't worry - there are effective approaches that can help you solve it! To start with consider implementing further methods of regularization like using dropout or L1/L2 regularization. Gathering more information through data augmentation or increasing your dataset size might be beneficial too. And if nothing else works out for you - try using a simpler model architecture! This technique prevents the possibility of your model relying too heavily on memorized training material instead of actually understanding real world examples better.