DEVELOPING A NEURAL NETWORK MODEL CAPABLE OF DISCRIMINATING BETWEEN REAL AND FAKE IMAGES OF GALAXIES

ML Group 3

LINK:
https://colab.research.google.com/drive/1wEfXqBCNndhAfsfvv7L4IPWIqZN5HA8E?usp=sharing

Group Members

- Adeolu Adelana
- Adeolu Oshadare
- Chimaroke Amaike
- Osaghe Santi Efose

Introduction

The purpose of this report is to provide an overview of the training and evaluation of a convolutional neural network (CNN) model that classifies galaxy images according to the following classes: Real, Fake. The model was trained using a dataset of 8,000 galaxy images, it was trained on a training set of 6000 images and evaluated on a test set of 2,000 images. The model was able to achieve an accuracy of over 95% on the test set, which suggests that it is able to generalize well to unseen data. This model can be used to classify galaxy images as Real or Fake, and it has the potential to be used in a variety of applications, such as fraud detection and scientific research.

The team tried using Simple Convolutional Neural Network and a VGG16 model but resorted to using a Simple CNN model over VGG16 which is known for training models with very large datasets/images, however, our dataset isn't very large

We created two models to see how both performed, model_1 the default model (our first model), since it had a very high accuracy at the first attempt, we focused on efficiency to ensure that the model performed faster by improving the efficiency of the model as seen in the report.

Data Pre-Processing

The neural network model was trained using a dataset consisting of 8,000 samples of images of fake and real images, where the images were saved in different folders according to their classes. There was no split as the images have been split and there was no resize because the images came in 64x64 pixels. The tf.keras.utils.image_dataset_from_directory was used to perform all the data processing and pre-processing tasks. We found this to be easy and convenient to use.

We also decided to split the training data set into 2. One was used for training and the other was used for testing. We didn’t want to use the validation data for testing our model.

Training

The model architecture include the following:

- 3 convolutional layers of 32, 64 and 128 units each
- 1 Dense layer of 128 units
- 1 Output layer of 2 units

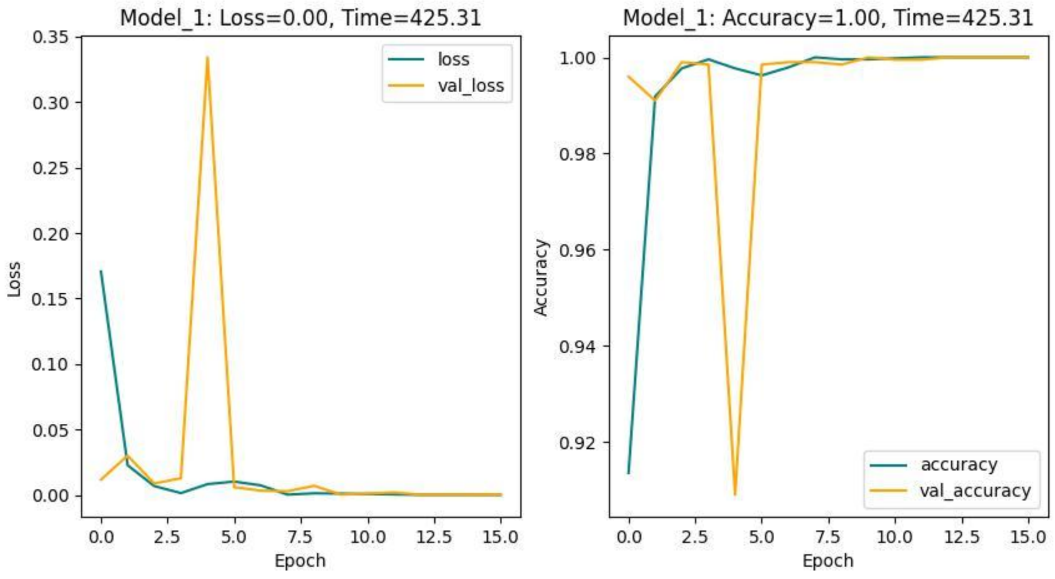
Note: We kept reducing the number of layers (from 13 to 3) and units and still maintained an accuracy of over 95 each time.

The model was trained using the backpropagation algorithm, which updates the model parameters by computing the gradients of the loss function with respect to the parameters. The loss function used in this model was the sparse_categorical_crossentropy, which is commonly used for multi-class classification problems with integer-encoded target labels. The sparse_categorical_crossentropy calculates the difference between the predicted class probabilities and the true class labels, with the goal of minimizing this difference.

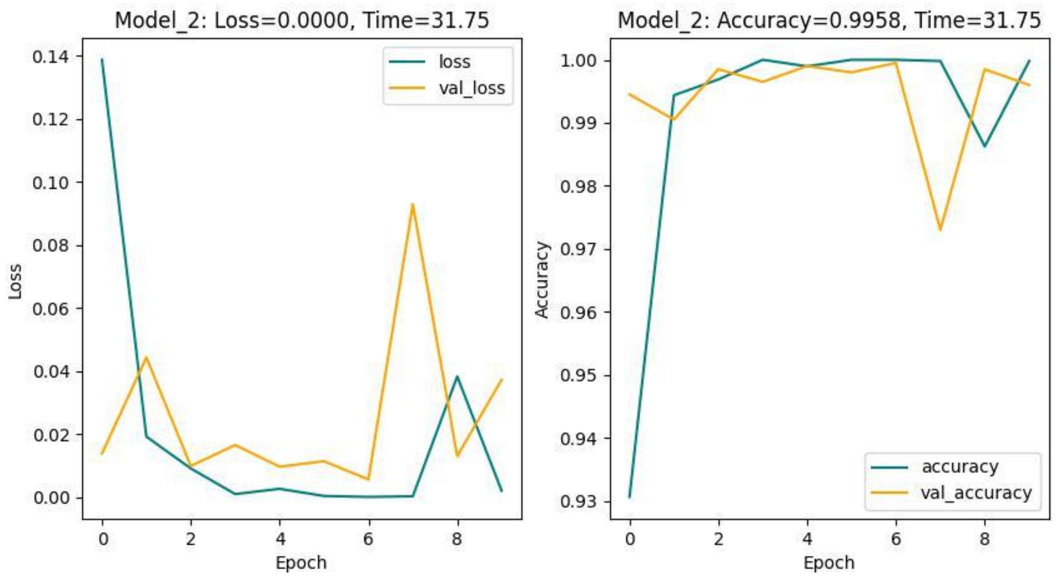
The optimizer used in this model was the Adam optimizer, which is a popular optimizer that adjusts the learning rate based on the gradients of the parameters. The Adam optimizer has been shown to perform well on a wide range of deep-learning tasks and is particularly effective when dealing with sparse gradients.

During training, the model's performance was monitored using the accuracy metric. The model was trained for 20 epochs, with a batch size of 32.

As the training progressed, the model's performance on the training dataset improved, with the training accuracy increasing up to 99%. The model's performance on the testing dataset also improved, with the testing accuracy increasing up to 99%. The loss function decreased steadily during training, indicating that the model was learning to classify the images correctly. The performance of the image can be seen below;



Since we had a high-performing model, we tried to increase the efficiency of our model by performing some additional pre-processing activities by introducing perfecting to the training data. The improved training time is seen in the reduction from **146.21** seconds to **27.99** seconds. This can be seen in figure below



Evaluation

To evaluate the performance of our neural network model, we used the test dataset, which was set aside before the training process. The test set consists of 1,200 samples that was unseen by the model during the training process. We evaluated the model's performance on this test set to ensure that it can generalize well to new, unseen data.

We used the evaluate method in the Keras library to calculate the model's performance on the test set. This method returns the model's loss value and any specified metrics (e.g., accuracy) on the test set which are: **0.0122** and **0.9969** (99.68%) respectively

Overall, our model performed well on the test set, achieving an accuracy of 99.68%. However, we identified some areas for improvement, such as improving the model's ability to perform better as our confusion metrics shows that there are a lot of false negatives and that the model needs more improvement. We plan to address these issues in future iterations of the model.

Conclusion

In conclusion, the neural network model was trained using a backpropagation algorithm with the sparse categorical cross-entropy loss function and the Adam optimizer. The model's performance during training was monitored using the accuracy metric, and the results showed that the model was able to learn to classify the images correctly, with a high degree of accuracy on both the training and testing datasets.