MAY 20, 2020

Machine Learning- COIY065H7

COURSEWORK REPORT

DIANA JAGANJAC

STUDENT NAME: DIANA JAGANJAC
MSC DATA SCIENCE
Email: djagan01@mail.bbk.ac.uk

Academic Declaration

I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software.

1. Introduction

The dataset I have decided to use is a modified cats and dogs image dataset, where there are 9890 images in total. For training, 9790 images were used, and for testing 200 were used. The data was downloaded from Microsoft Kaggle Cats and Dogs Dataset (Microsoft, 2020) which originally contained 25,000 images in total. The original dataset had previously been used for a Kaggle challenge 'Dogs vs. Cats Redux: Kernels Edition' in 2017 (Kaggle, 2017) and I have reduced its size for the purposes of this report.

The aim of this report is to train a CNN model to identify whether an image is of a cat or a dog, and to test the accuracy of models when using different optimisers and learning rates. The optimisers tested in this report are Adam, SGD, RMSprop and a customised version of WAME. Learning rates are tested for the customised WAME models using learning rates of both 0.001 and 0.01.

Section 2 of this report focuses on the methodology and design of the models and WAME optimizer, **section 3** on the results from the experiments and **section 4** provides final conclusions. The results can be found in **Appendix 1** (in raw format) and instructions for accessing the code for the project in **Appendix 2**.

2. Methodology and Design

Building the CNN model

I decided to run the CNN model in Kaggle as this would allow for faster running of the various models by utilising the Kaggle notebooks GPU option. I used the Keras library, backed by TensorFlow to implement my CNN models in Kaggle.

I started by downloading the Microsoft Kaggle Cats and Dogs Dataset (Microsoft, 2020) and then manually reduced the size of the dataset by deleting approximately half of the images in each of the 'cats' and 'dogs' image folders. I also removed 100 images from each folder for testing. In this way, I could ensure that the trained model would not have seen the 'test' images, and also I was able to balance the data in this way to have an equal number of images of cats and of dogs. This is important as otherwise, the CNN model may have been trained to be more predisposed to shapes, curves, lines and pixels which are associated with one type of animal, such as dogs, compared to the other, such as cats.

Once my manual pre-processing of the data was complete, I uploaded the data to Kaggle in two different folders, one containing the images for training and one containing the images for testing. I then started to process the data using Keras, and mostly followed the very useful Keras programming tutorials on the pythonprogramming.net website (Harrison, 2018). I began by resizing the images from a size of 400 to 150. This allowed for my input photos to take up less memory when loading into the CNN, an important consideration when working with thousands of images in batches. Following this, I used a function from the previously mentioned Keras tutorials to create my training data, and then also shuffled the images to avoid bias and error when training my CNN model. This is important to do as without shuffling; the CNN model may learn the characteristics of images of dogs and as a result be biased towards these images compared to the images of cats. In this way, the error of classification could also be increased, therefore shuffling is an important step. Finally, I used NumPy to create a list of values corresponding to my training data and standardised this data to make values range between 0 and 1. This can improve the results of classification CNN's as the input values are normalised to make sure they are comparable with all of the other input images. Without this step, the classification error would be greater. Once the processing of my data using Keras was complete for the training data, I did the same for the testing dataset of 200 images.

After completing all of the pre-processing steps for my data, I moved on to building my CNN models. I used Keras to build my models and used a model I found online which uses 15 hidden layers, both the relu and sigmoid activation functions and compiles the model using the binary cross-entropy technique (Chollet, 2016). The use of the relu activation function is useful for my CNN model which utilises deep-learning as the relu activation function is less computationally complex and also tends to increase performance when compared to the sigmoid activation function (Krizhevsky, Sutskever and Hinton, 2012). Nevertheless, the use of the sigmoid activation function in the final layer is useful as the sigmoid activation function classifies data into a binary classification of either 0 or 1, which is ideal for this classification task. Furthermore, the use of binary cross-entropy loss function is appropriate for my model as the classification task is binary between two classes. i.e. images of either cats or dogs.

Once my data was processed, and my models were built, I decided to run a number of experiments each time using a different a different optimizer function. I decided to run experiments by using the following optimizers: Adam, SGD, RMSprop and a customised version of WAME. I also decided to experiment with the learning rate and tested a learning rate of both 0.001 and 0.01 for my customised WAME optimizer. I ran 20 epochs for each experiment and set a cross-validation testing rate of 10%, which meant that the first 10% of the input images would be held back from training and used to test and validate the accuracy of the model. Once this was complete, I also used the Keras 'evaluate' function to test the model on my testing dataset of a further 200 images which were previously unseen to the model. I ran each of the five experiments ten times, and then took the average of both the training validation accuracy, and testing accuracy which can be found in Figure. 1, Section. 3 of this report.

Initialising WAME

I started initialising the WAME algorithm by initialising the necessary parameters for the algorithm, as mentioned in lines 1 and 2 of the algorithm. These included the variables 'alpha', eta+ which I named 'scale_up', /'scale_down', 'zeta_min', 'zeta_max' and the learning rate which I named 'lr'.

Following this, I moved on to deriving the new gradients, and step sizes for the WAME algorithm. I used python code for the Rprop to build this section (ntnu-ai-lab, 2020). This code was useful as it aided me in understanding how to implement the gradient comparison of the old and current gradients using the Keras switch function.

After using this code from Rprop, I moved on to looking at how the RMSprop algorithm is implemented in Keras (keras team, 2020). This was useful as it helped me to understand how the accumulators are implemented in RMSprop and allowed me to implement the code for the new 'Z' and theta values in lines 9 and 10 of the WAME algorithm. This ultimately allowed me to calculate the WAME optimizer values in lines 11 and 12 of the WAME algorithm.

3. Experiments, findings and discussion

Overall Results

After initialising the WAME algorithm, and developing a CNN architecture, five experiments were run a total of ten times each. The average accuracy of these results was calculated and are summarised in Table. 1. Table. 1 gives information for the testing accuracy run on the testing dataset, and the training validation accuracy which was run during the training stage. The full break-down of the results, including the results for the training accuracy before validation can be found in Appendix. 1.

Method	Testing Accuracy	Training Validation Accuracy
CNN + Adam	0.83	0.7743
CNN + SGD	0.715	0.6743
CNN + RMSprop	0.785	0.7293
CNN + WAME, lr = 0.001	0.835	0.8192
CNN + WAME, lr = 0.01	0.78	0.7712

Table. 1: Table showing the results from the five experiments run. The average accuracy for each experiment was calculated and is displayed. The optimizers Adam, SGD, RMSprop and WAME were tested, as well as the learning-rate (lr) for the WAME algorithm.

Optimiser and Learning-Rate Evaluation

The model accuracy and the model loss were evaluated graphically during the running of each experiment, and an example of this is shown in Figures 2 and 3. I decided to focus on these two metrics for the models during the training stage and the cross-evaluation stage of the experiments in order to evaluate the progress of each model over the 20 epochs that it ran. The code to create these line-graphs was found online (Machine Learning Mastery, 2016). This section of the report will focus on analysing the model training and cross-validation testing history found for each model in Figures 2 and 3.

Adam Optimizer

The experiment with the Adam optimizer shows that training accuracy and training loss followed a nice pattern, indicating that the model was learning during the training process rather than memorizing and overfitting. Nevertheless, the cross-validation test results show a different picture, in which the model accuracy stagnated around fourth epoch and did not improve significantly from there. The model loss also indicates that it worsened as the epoch's went on, this is most likely due to some overfitting of the model. Nevertheless, the Adam optimizer showed good results overall on the testing dataset (see Table. 1) despite the cross-validation training accuracy being not as high.

SGD Optimiser

The SGD optimizer indicated a nice learning rate as model accuracy continually increases, and loss decreases for the training model as seen in Figure. 3. The cross-validation testing results for this optimizer however paint a different picture, as there are huge fluctuations in both the accuracy and the loss graphs for this optimizer. Interestingly, the accuracy and loss fluctuations seem to match each other, which is something that cannot be said for the cross-validation results of the other optimizers. This perhaps indicates a very correlated relationship between some of the parameters in the model, and therefore could indicate the overfitting of the model. The SGD optimizer performed the worst of all the optimizers used in the experiments, and so this is most likely the case.

RMSprop

The learning rate of the RMSprop model showed similar results to that of the Adam model; the training accuracy increases steadily, and the training loss decreases steadily forming a nice curve. The cross-validation results however show that both the model accuracy and loss functions fluctuated repeatedly despite remaining within a fixed range of values. The model loss for the cross-validation results do also show a sharp increase in loss after 17.5 epochs. The RMSprop optimizer did not perform badly overall, however it was outperformed by the Adam optimizer and the WAME optimizer with a learning rate of 0.001.

WAME

The results for both of the WAME experiments are fairly similar when considering the training accuracy and training loss functions. Both follow similar patterns, and also form curves alike those formed by the Adam and RMSprop optimizers.

The WAME model with a learning rate of 0.001 had fluctuations in its cross-validation testing accuracy, as well as its cross-validation testing loss. After the thirteenth epoch, there seemed to be a greater increase in model loss, whilst it stabilised again at the fifteenth epoch, only to jump again at the sixteenth. The cross-validation model loss seems to indicate that the best loss was around the ninth epoch, perhaps indicating that 20 epoch were not necessary for this model. Of the four optimisers used, the WAME optimiser with a learning rate of 0.001 provided the best results for the testing dataset, with a testing accuracy of 0.835.

The WAME model with a learning rate of 0.01 followed a similar training pattern as the model with a learning rate of 0.001. The cross-validation testing accuracy results seemed to fluctuate quite heavily, reaching a peak at around the 13th epoch. The cross-validation model loss too had significant fluctuations, with an exponential increase in loss occurring at around epoch 17.5. Again, the model loss seems to be lowest at around the 10th epoch, indicating that this model may not have required more than 10 epochs to generate viable results. This model had a lower testing accuracy when compared to the WAME model with a learning rate of 0.001, with a testing accuracy of 0.78.

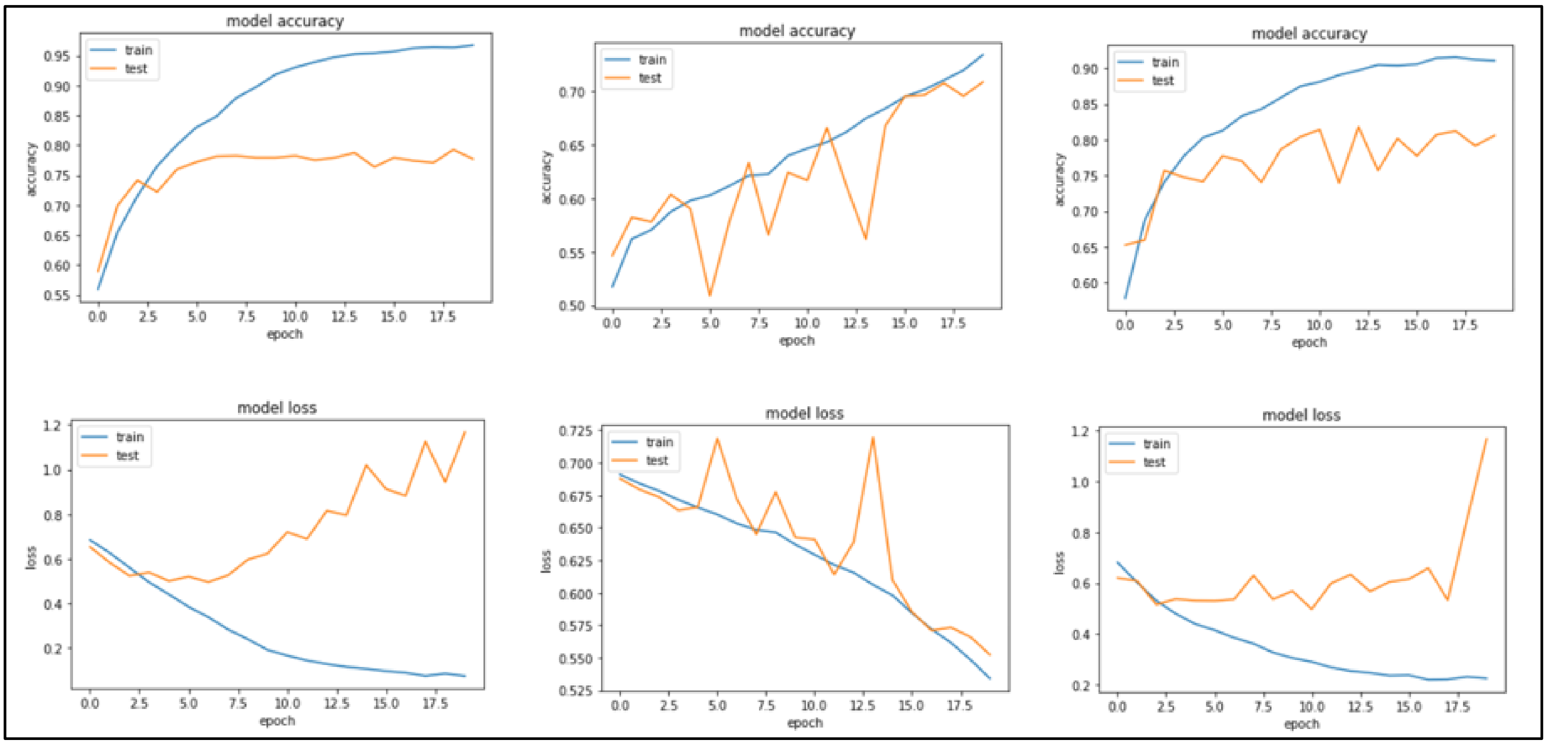

Table. 2: tables showing the results of the training and cross-validation testing accuracy and loss from three out of five experiments using different optimizers for a CNN classification task. The optimizers used were Adam (left), SGD (middle) and RMSprop(right).

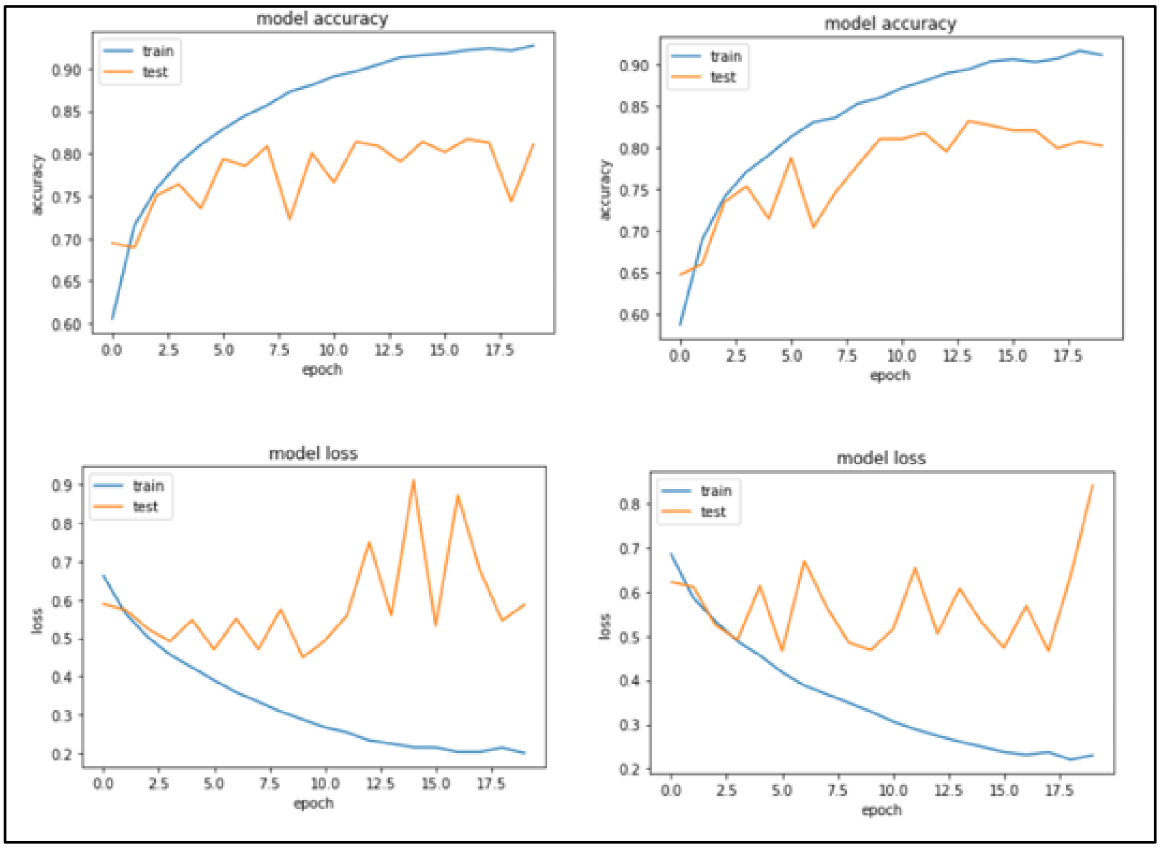

Table. 3: tables showing the results of the training and cross-validation testing accuracy and loss from two out of five experiments using different optimizers for a CNN classification task. The optimizers used were WAME, lr = 0.001 (left) and WAME, lr = 0.01 (right).

Conclusions

This report discussed the aims, methodology, design and results of the five experiments carried out to analyse the accuracy of a CNN architecture in a binary classification technique when different optimizers and learning rates were used. The results indicated that each model was prone to overfitting to some degree, and in the case of the WAME optimizer, which achieved the best results, a learning rate of 0.001 and 9 to 10 epochs produced the best results when run on the testing dataset. Further improvements to the model could be made, such as experimenting with different dropout rates in the model's dropout layers to reduce overfitting. Experimenting with the number of neurons and hidden layers could also lead to better results.

References

Chollet, F. (2016) ‘Building powerful image classification models using very little data’, *The Keras Blog*, pp. 1–13. Available at: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html (Accessed: 20 May 2020).

Harrison (2018) *Python Programming Tutorials*, *2018-08*. Available at: https://pythonprogramming.net/loading-custom-data-deep-learning-python-tensorflow-keras/ (Accessed: 20 May 2020).

Kaggle (2017) *Dogs vs. Cats Redux: Kernels Edition*. Available at: https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/overview (Accessed: 20 May 2020).

keras team (2020) *keras/optimizers.py at master · keras-team/keras*. Available at: https://github.com/keras-team/keras/blob/master/keras/optimizers.py (Accessed: 20 May 2020).

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012) *ImageNet Classification with Deep Convolutional Neural Networks*. Available at: http://code.google.com/p/cuda-convnet/ (Accessed: 20 May 2020).

Machine Learning Mastery (2016) ‘Display Deep Learning Model Training History in Keras Access Model Training History in Keras’, pp. 1–10. Available at: https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/ (Accessed: 20 May 2020).

Microsoft (2020) *Download Kaggle Cats and Dogs Dataset from Official Microsoft Download Center*. Available at: https://www.microsoft.com/en-us/download/details.aspx?id=54765 (Accessed: 20 May 2020).

ntnu-ai-lab (2020) *RProp/rprop.py at master · ntnu-ai-lab/RProp*. Available at: https://github.com/ntnu-ai-lab/RProp/blob/master/rprop.py (Accessed: 20 May 2020).

Appendix 1- Results (raw data)

The tables below give all the results for the experiments that were run, the first table gives the results of the testing accuracy on the testing dataset, the second table the results of the cross-validation accuracy on the training dataset, and table three the training accuracy on the training dataset.

Method	CNN + Adam	CNN + SGD	CNN + RMSProp	CNN + WAME, lr = 0.001	CNN + WAME, lr = 0.01
Run 1	0.9	0.68	0.75	0.86	0.82
Run 2	0.81	0.82	0.87	0.86	0.79
Run 3	0.76	0.72	0.68	0.76	0.83
Run 4	0.87	0.76	0.78	0.93	0.83
Run 5	0.8	0.82	0.77	0.87	0.74
Run 6	0.69	0.76	0.75	0.8	0.77
Run 7	0.9	0.72	0.77	0.75	0.84
Run 8	0.81	0.76	0.88	0.79	0.86
Run 9	0.68	0.7	0.73	0.89	0.76
Run 10	0.85	0.74	0.8	0.84	0.74

Method	CNN + Adam	CNN + SGD	CNN + RMSProp	CNN + WAME, lr = 0.001	CNN + WAME, lr = 0.01
Run 1	0.76	0.67	0.75	0.84	0.69
Run 2	0.77	0.65	0.79	0.76	0.74
Run 3	0.81	0.7	0.74	0.86	0.63
Run 4	0.8	0.76	0.78	0.87	0.72
Run 5	0.82	0.62	0.73	0.79	0.76
Run 6	0.75	0.66	0.75	0.71	0.72
Run 7	0.73	0.67	0.76	0.85	0.82
Run 8	0.79	0.59	0.76	0.77	0.78
Run 9	0.65	0.66	0.83	0.77	0.79
Run 10	0.73	0.69	0.73	0.66	0.74

Method	CNN + Adam	CNN + SGD	CNN + RMSProp	CNN + WAME, lr = 0.001	CNN + WAME, lr = 0.01
Run 1	0.76	0.76	0.83	0.87	0.82
Run 2	0.82	0.62	0.95	0.96	0.8
Run 3	0.86	0.66	0.92	0.9	0.88
Run 4	0.78	0.67	0.71	0.94	0.83
Run 5	0.86	0.74	0.78	0.96	0.91
Run 6	0.86	0.78	0.91	0.87	0.82
Run 7	0.85	0.73	0.77	0.88	0.8
Run 8	0.95	0.75	0.75	0.95	0.8
Run 9	0.81	0.76	0.84	0.96	0.89
Run 10	0.73	0.61	0.93	0.97	0.85

Appendix 2- Code

All code is found in the uploaded GPU_ML.ipynb file and can be run by opening the file in a Jupyter notebook.