CIFARTile Image Classification

Introduction

Image classification is one of the most interesting studies in computer vision. The main aim of this paper is to classify images among multiple classes using deep learning algorithms such as Convolutional neural networks (CNNs) and ResNet-50. For preprocessing, normalisation, scaling, one-hot encoding and data augmentation is conducted. Additionally, hyper-parameter tuning is implemented to find its best performance. At last, the performance is assessed by the process of loss and accuracy and confusion matrix.

Data Exploration

In this paper, the image dataset named CIFARTile presented by CVPR-NAS [1] is used. Each of CIFARTile image consists with four images from CIFAR10 [2] and its own label. The label is an integer value from 0 to 3 and related to the number of unique classes. To be specific, it is calculated by subtracting one from the total number of unique classes. For example, Fig. 1. includes two birds, one automobile and one airplane so the number of unique classes is three. Therefore, the label is 2 which is subtracted one from three.

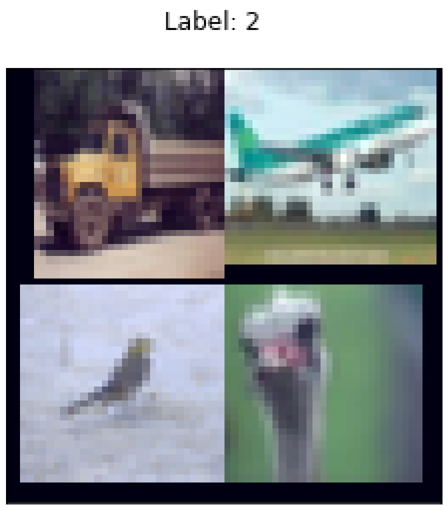


Fig. 1. Sample image from the CIFARTile dataset

Since the label has four classes – from 0 to 3, it is necessary to check the balance throughout all classes. An imbalanced dataset could cause overfitting so it is essential to set as balanced dataset. As Fig. 2. shown, all the dataset – training, validation and testing set – are equally distributed in each class.

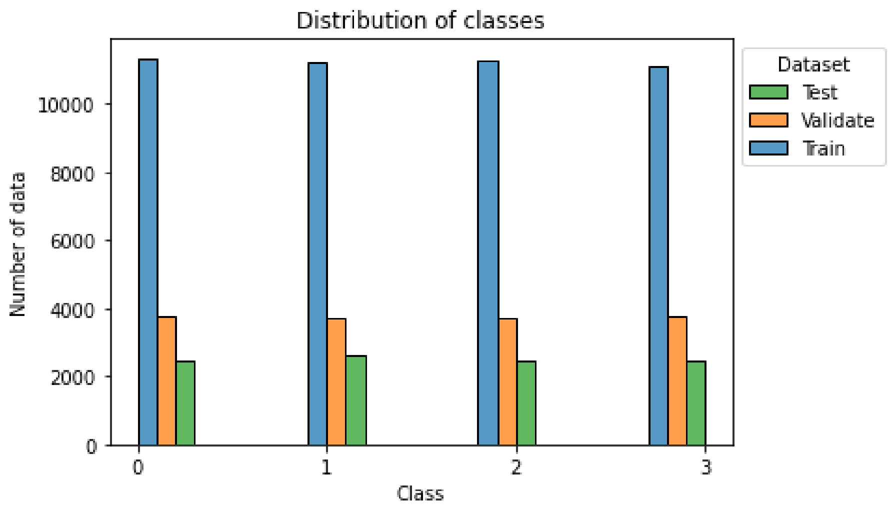


Fig. 2. Distribution of Training, Validation and Test set in each class

Methodology

Image Preprocessing

For deep learning approach, image data with pixels value from 0 to 1 can improve the performance of training model [3]. However, the dataset given is in range of -2 to 2 so normalisation has been conducted by

$$x_n = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{1}$$

In addition, while the deep learning model requires specific structure in the order of (height, width, channels), the given dataset is structured as (channels, height, width). Thus, the dataset's shape has been converted. However, using *numpy.shape()* method damages the original image even though it changes the shape successfully. Thus, *numpy.rollaxis()* method [4] is used for converting the dataset's shape with keeping the image safe. After that, one-hot encoding approach is executed on label to avoid resulting in poor performance.

Most importantly, data augmentation is performed for training and validation dataset. The data augmentation method gives the variation for original image such as angle, brightness or colour changes. This technique generates more images so that the image dataset becomes bigger and the bigger dataset results in higher performance of deep learning model. Therefore, *image_data_generator()* method [5] is used with adopting shearing, channel shifting and flipping horizontally and vertically (Fig. 3.).

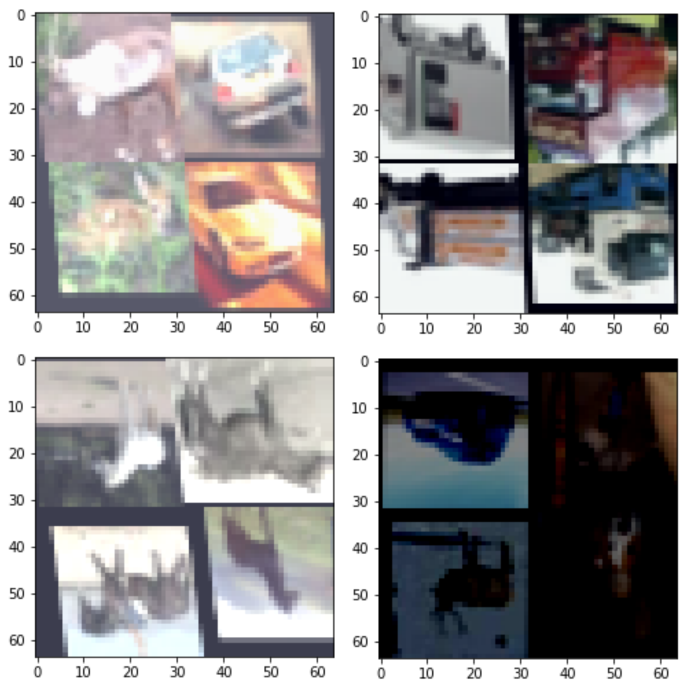


Fig. 3. Images generated by Augmentation

Architecture

Convolutional neural networks (CNNs) are well known as its high performance in detecting the important features automatically as long as training with large amount of training samples and powerful GPU [6]. Especially, CNN stands out on computer vision and image processing. Since the given dataset is digital image and requires classification depending on the features of each object in the image, it is expected to get high performance with CNN technique.

Furthermore, a transfer learning approach is also applied for better performance. ResNet-50 is Residual Network with 50 layers and is based on CNN but pre-trained model [7]. Thus, it is expected to have higher performance than CNN.

According to Ottoni *et al.* [8], hyper-parameters are important factors that influence the final capacity of models. Depending on the number of neurons, layers and the learning rate of optimizers, the model results in its highest capacity or causing poor performance with overfitting. Therefore, hyper-parameter tuning has conducted for both CNN and ResNet-50 by using Keras Tuner [9].

Moreover, the *EarlyStopping()* method [10] is added to prevent overfitting and to reduce the loss during training. The *patience* parameter is set as 2 for monitoring validation loss since the accuracy can be improved even after one epoch. However, as it is possible to get the best model after 2 epochs, the *ModelCheckpoint()* method is applied to save the best performance.

Results

Hyper-parameter tuning

The hyper-parameters selected are Number of Layers from 1 to 2, Number of Neurons from 32 to 512 increasing by 32 and Learning Rate of Adam optimizer with the value of 0.01, 0.001 and 0.0001. Among these combinations, three different specifications are chosen randomly for CNN and ResNet-50 as Table 1 and Table 2 shown.

Table I: Hyper-parameter Tuning Specification of CNN

	Number of Layers	Number of Neurons (First Layer)	Number of Neurons (Second Layer)	Learning Rate	Accuracy
Set A	2	288	32	0.0001	0.34
Set B	2	352	480	0.001	0.25
Set C	2	320	64	0.01	0.25

Table II: Hyper-parameter Tuning Specification of ResNet-50

	Number of Layers	Number of Neurons (First Layer)	Number of Neurons (Second Layer)	Learning Rate	Accuracy
Set A	1	320	-	0.0001	0.48
Set B	2	160	32	0.0001	0.47
Set C	1	128	-	0.01	0.25

According to Table 1, CNN obtained its highest accuracy with Set A. Especially, the accuracy score of Set A is 0.34 which is outstanding compared to Set B and C. Also, Table 2 figures that ResNet-50 got more than 0.47 with Set A and B, which is higher than CNN as expected previously.

However, the hyper-parameter tuning was conducted only with 3 epochs so those accuracy are not the finals and it has a possibilities to improve with more epochs when fitting in the model with best parameters.

Testing

With the best parameters from hyper-parameter tuning, CNN ended up getting 0.41 accuracy and 1.20 after 15 epochs and ResNet-50 resulted in 0.61 accuracy and 0.93 loss (Fig. 4.). As Fig. 4. described, ResNet-50 shows not only higher accuracy but also lower loss than CNN. As predicted, ResNet-50 has higher performance than CNN since it is pretrained model.

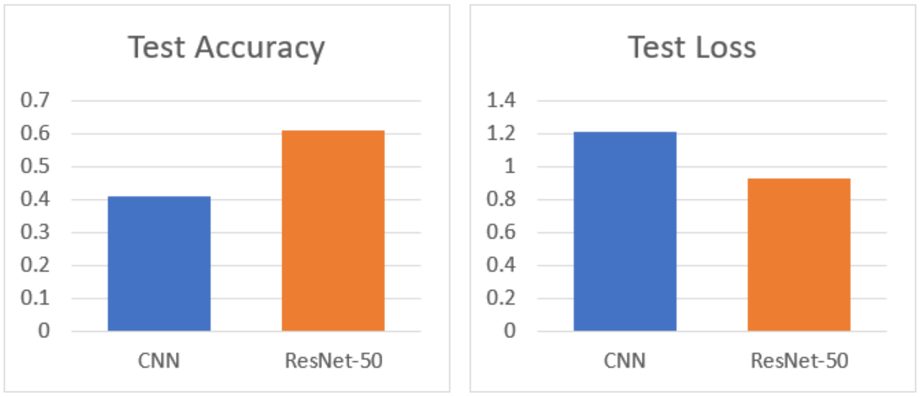


Fig. 4. Comparison of Test Accuracy (Left) & Test Loss (Right) between CNN and ResNet-50

Additionally, it is possible to check whether the results are overfitting. Fig. 5. Illustrates the validation loss and accuracy of ResNet-50. As it shown, overfitting starts at epoch 8. However, since the pipeline includes the *EarlyStopping()* and *ModelCheckpoint()* methods, the final model is saved before overfitting occurs so that it could result in its best performance.

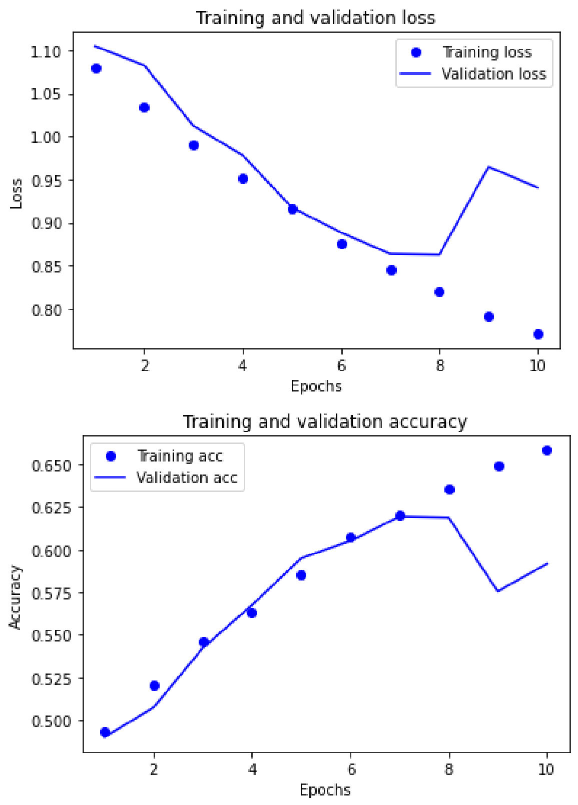


Fig. 5. Loss (Top) & Accuracy (Bottom) of Training and Validation set – ResNet50

As a result of testing, the confusion matrix is plotted for both CNN and ResNet-50. According to Fig. 6. ResNet-50 predicts almost more precisely than CNN. It is interesting that CNN predicts label 0 well, on the other hand, tends to predict label 1 and 2 as label 3.

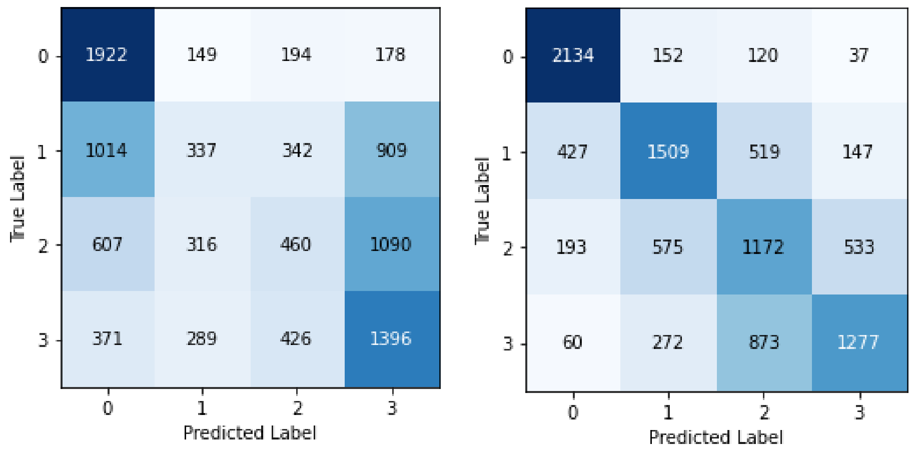


Fig. 6. Confusion Matrix of CNN (Left) and ResNet-50 (Right)

Discussion

During the trial of experiments, there are few keys found out to improve the model's performance. Firstly, hyper-parameter tuning brought a huge difference especially with ResNet-50. Before hyper-parameter tuning, ResNet-50 tends to predict only one class but it has been solved and reached to its highest accuracy by hyper-parameter tuning with finding the best number of layers, neurons and learning rate. Also, data augmentation solved a problem with overfitting from CNN. It is proved that the more data gives higher accuracy in deep learning.

As a future work, it is suggested that trying with more variation of hyper-parameters and more epochs. Since the limitation of GPU specification, only three different hyper-parameters have been conducted. Also, even though ResNet-50 stopped at epochs 8 – since it only has 2 *patience* parameters – it could have improved after more epochs with larger *patience* value. In addition, it is suggested to try with more deep learning model and pre-trained model. Again, more powerful GPU can run more models with hyper-parameter tuning.

References

[1] CVPR-NAS-Datasets, "Github," 2021. [Online]. Available: https://github.com/RobGeada/cvpr-nas-datasets. [Accessed 5 Dec 2022].

[2] A. Krizhevsky, V. Nair and G. Hinton, "CIFAR-10 (Canadian Institute for Advanced Research)," [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html. [Accessed 5 Dec 2022].

[3] K.-M. Koo and E.-Y. Cha, "Image recognition performance enhancements using image normalization," *Human-centric Computing and Information Sciences,* vol. 7, no. 33, 2017.

[4] C. R. Harris, K. J. Millman, S. J. van der Walt and et al, "Array Programming with NumPy," *Nature,* vol. 585, no. 7825, pp. 357-362, 2020.

[5] M. Abadi, A. Agarwal, P. Barham , E. Brevdo, Z. Chen, C. Citro and et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. [Online]. Available: https://www.tensorflow.org/. [Accessed 2 Jan 2023].

[6] D. Han, Q. Liu and W. Fan, "A new image classification method using CNN transfer learning and web data augmentation," *Expert Systems with Applications,* vol. 95, pp. 43-56, 2018.

[7] A. Shabbir, N. Ali, J. Ahmed, B. Zafar, A. Rasheed, M. Sajid, A. Ahmed and S. H. Dar, "Satellite and Scene Image Classification Based on Transfer Learning and Fine Tuning of ResNet50," *Mathematical Problems in Engineering,* vol. 2021, p. 18 , 2021.

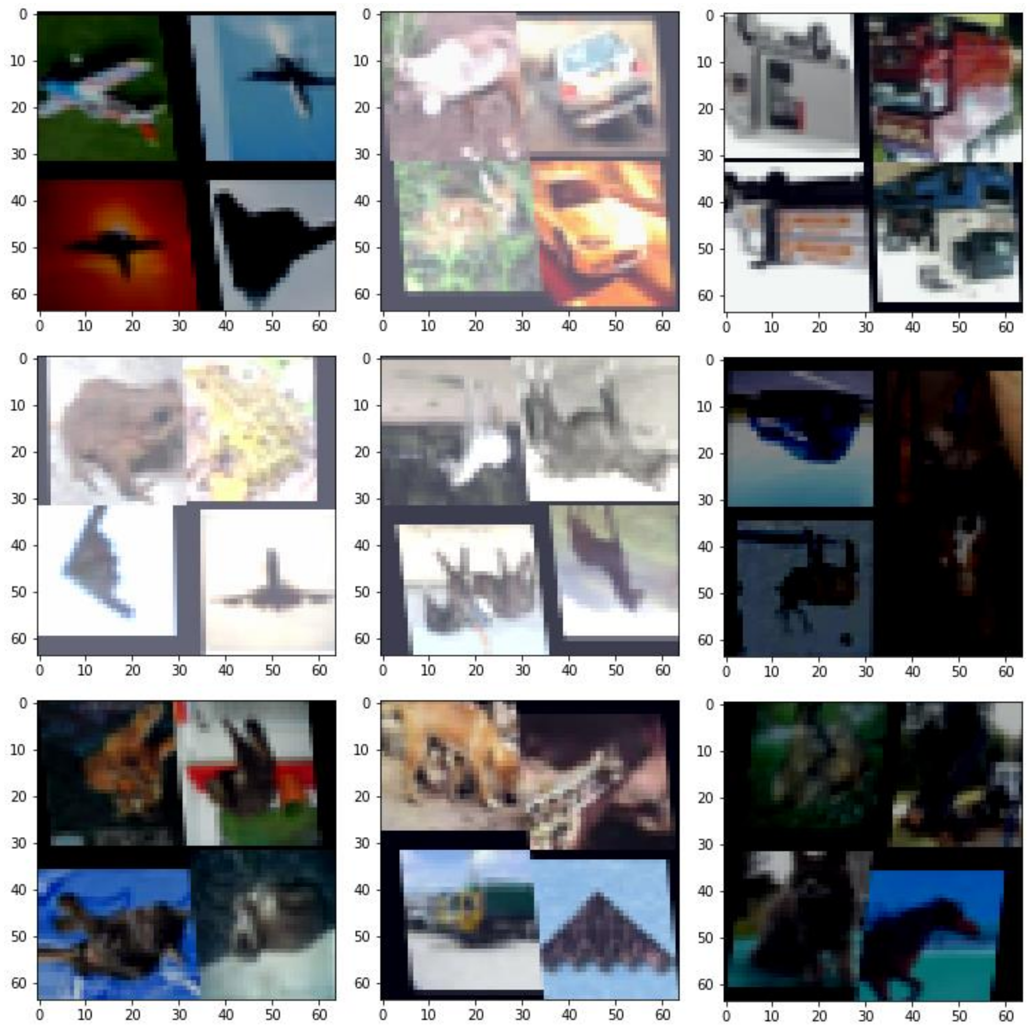
[8] A. Ottoni, . M. Novo and D. Costa, "Hyperparameter tuning of convolutional neural networks for building construction image classification," *Vis Comput ,* 2022.

[9] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi and others, *KerasTuner,* https://github.com/keras-team/keras-tuner, 2019.

[10] "EarlyStopping," Keras, 2022. [Online]. Available: https://keras.io/api/callbacks/early_stopping/. [Accessed 5 Jan 2023].

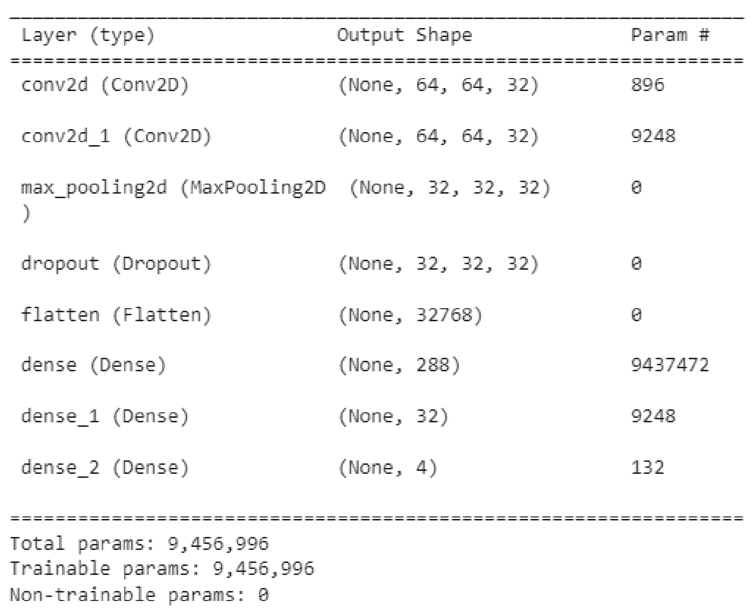
Appendices

Appendix A – 9 Images Generated by Data Augmentation



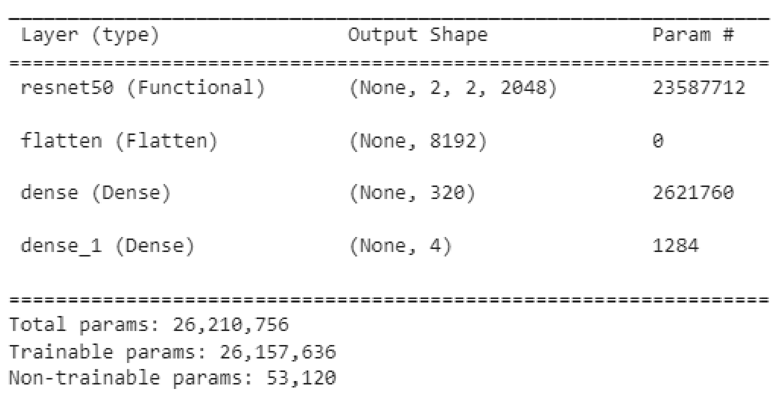
Appendix B – Final Model Architecture of CNN (Top) and ResNet-50 (Bottom)

CNN



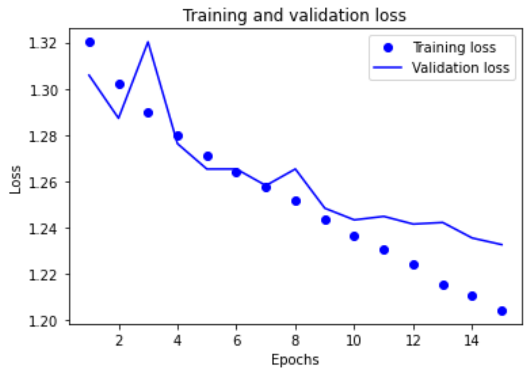
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        896       
                                                                 
 conv2d_1 (Conv2D)           (None, 64, 64, 32)        9248      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 32, 32, 32)        0         
                                                                 
 flatten (Flatten)           (None, 32768)             0         
                                                                 
 dense (Dense)               (None, 288)               9437472   
                                                                 
 dense_1 (Dense)             (None, 32)                9248      
                                                                 
 dense_2 (Dense)             (None, 4)                 132       
                                                                 
=================================================================
Total params: 9,456,996
Trainable params: 9,456,996
Non-trainable params: 0
```

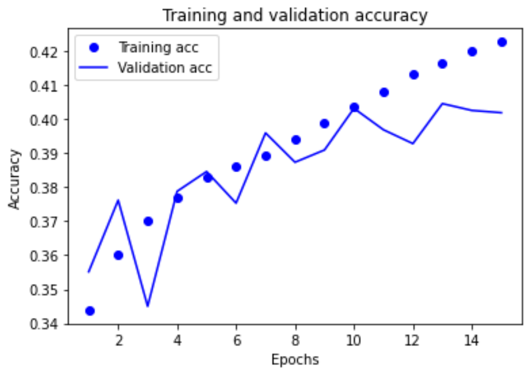
ResNet-50



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 resnet50 (Functional)       (None, 2, 2, 2048)        23587712  
                                                                 
 flatten (Flatten)           (None, 8192)              0         
                                                                 
 dense (Dense)               (None, 320)               2621760   
                                                                 
 dense_1 (Dense)             (None, 4)                 1284      
                                                                 
=================================================================
Total params: 26,210,756
Trainable params: 26,157,636
Non-trainable params: 53,120
```

Appendix C - Loss (Top) & Accuracy (Bottom) of Training and Validation set (CNN)


Training and validation loss
Training loss
Validation loss
Loss
Epochs


Training and validation accuracy
Training acc
Validation acc
Accuracy
Epochs