NANYANG TECHNOLOGICAL UNIVERSITY

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SCHOOL OF COMPUTER SCIENCE AND

ENGINEERING

# Assignment for SC4001 Neural Networks and Deep Learning

AY2023-2024

**Group members:**

| Name | Matric No. |
|---|---|
| Chua Ming Ru | U2140945D |
| Min Khant Htoo | U2140545E |
| Alviento Adrian Nicolas Belleza | U2140615H |
| | |
| | |

# Content Page

## Introduction

For this project, we will be primarily looking at the Convolutional Neural Network (CNN) developed by Liu et al. for flower classification [1]. In their study, they trained and tested their model on a custom dataset, as well as the Oxford 102 Flower Dataset [2]. However, only 79 labels are used for their classification task.

Thus, in this project, we replicate the model as described by Liu et al. to create a baseline model for the classification of all 102 flower species in the Oxford 102 Flower Dataset. Following that, we will explore various techniques to improve the model. This includes the use of batch normalisation [3], depthwise & pointwise convolution, MixUp [4] and data augmentation. We then compare our final model with the starting model to show the improvements made.

## Network Architecture

The proposed deep CNN by Liu et al. consists of eight layers, subdivided into five convolutional layers and three fully-connected layers. The network takes as input a 100x100x3 image and passes it through a series of convolutional layers with varying kernel sizes and channels: 64 kernels of 5x5x3, 128 kernels of 5x5x64, 256 kernels of 3x3x128, 512 kernels of 3x3x256, and another 512 kernels of 3x3x512. Convolutional strides are uniformly set at 1 pixel, with padding adjustments according to kernel sizes. The first and second convolutional layers undergo a response-normalisation layer. Spatial max-pooling is done after each convolutional layer, except the fourth, using a 3x3 window and a stride of 2. After the convolutional stack are the three fully-connected layers of size 1024, 512 and 79 respectively.

A softmax function is applied to the output layer for classification for the 79 distinct classes. The ReLU non-linearity function is applied to the output of every convolutional and fully-connected layer. The network architecture can be seen below (Fig 1).

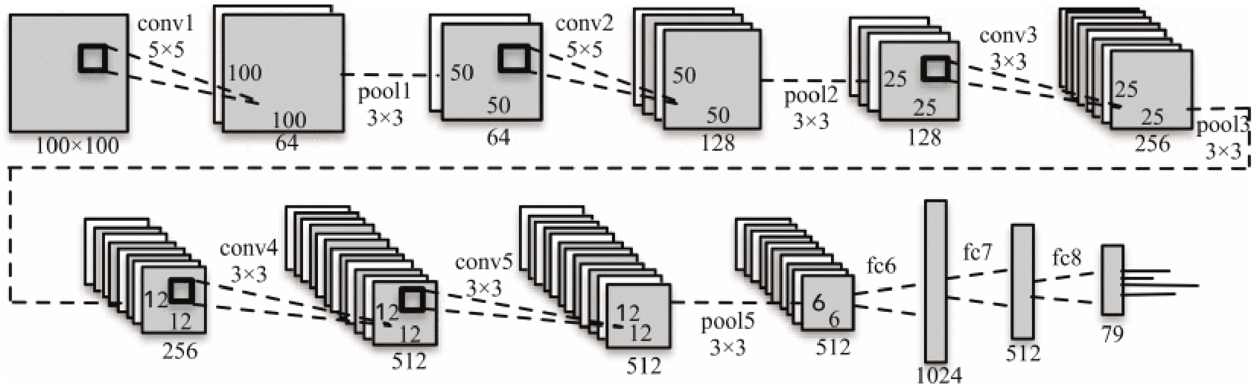

**Fig 1** [1]

To improve classification performance, the following techniques were used in the paper:

I. Local Response Normalisation

$$b_{x,y}^{i} = a_{x,y}^{i} / (k + \sum_{j=max(0,i-n/2)}^{min(N-1,\, i+n/2)} (a_{x,y}^{i})^{2})^{\beta}$$

- Here, $a_{x,y}^{i}$ denotes the activity of a neuron computed by applying kernel $i$ at position $(x,\, y)$, and then applying $ReLU$ non linearity to this activity. The response normalisation $b_{x,\,y}^{i}$ is as defined above.
- The sum is computed at the same position in the field of convolution kernels, $n$, and $N$ is the total number of convolution kernels in the layer.
- The constants are specified as follows: $k\ =\ 2\ ;\ n\ =\ 5\ ;\ \beta\ =\ 0.75$.

    - This normalisation was applied after the ReLU function of the first two convolutional layers.

II. Overlapping Pooling
    - A max pooling layer is applied at each of the convolutional layers with overlapping strides.
    - This is achieved by setting step length, $s = 2$ and pooling unit, $z \times z = 3 \times 3$
III. Dropout regularisation
    - A dropout rate of 0.5 is used for every fully-connected layer.

Finally, the dataset undergoes a specific preprocessing procedure. The maximum square patch from the resulting image generated by combining the saliency map and luminance map from the original image was cropped out. This focuses the image on the specific area of interest, in this case, the flower. This crop was then resized to 100x100. The mean RGB value computed on the training set is then subtracted from each pixel, converting back the images to three channels with normalised RGB values.

## Baseline model

As a baseline model, we replicated the architecture used by Liu et al. as closely as possible. However, some differences had to be made. Firstly, the output layer in our baseline model is changed to 102 neurons in order to classify the 102 flower species in the Oxford dataset. Figure 2 shows the distribution of the Oxford flower 102 dataset if we use the train, validate and test dataset split by Liu et al.. We noted that while the test and validate dataset is class-balanced, the dataset uses a 1020:1020:6149 train:validate:test split, which is not the standard 80:10:10 split. This is a bug with the dataset. As such, we chose to swap the test and train dataset around to get the distribution closer to the standard 80:10:10.

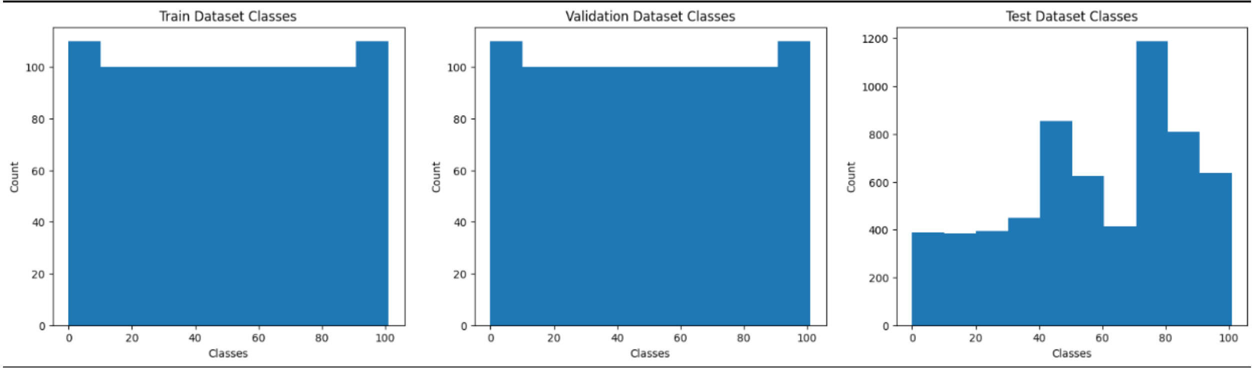

**Fig 2**

Next, we followed the data preprocessing steps found in Liu et al. paper to the best of our knowledge. However, when compared with a simple resizing and normalisation, the latter yielded better validation loss. As such, we chose the latter data preprocessing method as a baseline.

Since some parameters were not explicitly mentioned in the paper, we specified them as follows:

I. Batch size: 128
II. Optimiser: Adam with initial learning rate of 0.001
III. Loss function: Cross Entropy Loss
IV. Epochs: 100 with early stopping patience set at 15 epochs

Using the specified model and parameters above, we trained our baseline model. It achieves a validation loss and a validation accuracy of 2.8758 and 47.16% respectively. We specifically compare our implementation with other CNNs performing the same function (Table 1), specifically in the GoogLeNet and AlexNet implementation by Gurnani et al. on the same dataset [5]. Thus, our baseline model performs considerably worse with over 20% reduction in both top 1 and top 5 accuracies on the test set.

| Model | Accuracy (Top 1) | Accuracy (Top 5) |
|---|---|---|
| AlexNet | 43.39% | 68.68% |
| GoogLeNet | 47.15% | 69.17% |
| Baseline Model | 18.9% | 45.9% |

**Table 1**

## Improving Results

To improve the results of the baseline model, we focus more on improving the training process and the dataset instead of the model architecture, since we do not have access to state-of-the-art computing resources.

### Batch Normalisation

Batch normalisation is the process of normalising the input batch data to zero mean and unit variance. It can be applied before the start of every layer to make more accurate predictions while also making the model converge faster, thereby reducing training time, as shown in the original paper [3]. It is speculated that batch normalisation reduces *internal covariate shift*.

The formula for batch normalisation is as follows:

$$\chi_{i,j} = (x_{i,j} - \mu_j)/\sqrt{\sigma_j^2 + \epsilon}$$

where $\mu_j$ and $\sigma_j$ are the mean and variance of the input channel $j$ respectively.

As we can see in the figure below (Fig 3), the training time decreases by 2 seconds per epoch when using batch normalisation, and the early stopping takes longer to activate than the base model. This totals up to about 80 seconds of time saved from epoch 0 to 40. Moreover, the validation loss is in general smaller with batch normalisation than without. However, there tends to be spikes in the validation loss in the batch normalisation model, which would suggest that some hyperparameter tuning for batch normalisation is needed.

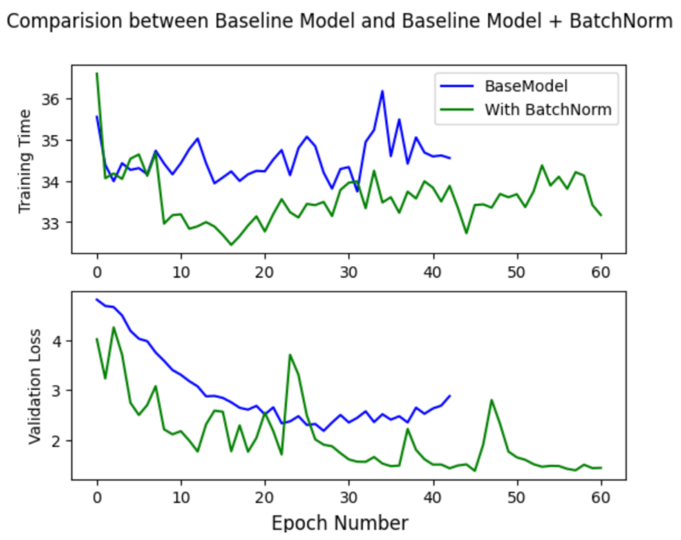

**Fig 3**

We have identified the momentum to be the most important hyperparameter for batch normalisation since it determines:

I. How fast the population mean and variance of the weights are determined at the start of training.
II. The stability of the normalised weights as time goes on.

This is important as it allows the model to ignore noise when training the model. We determined $momentum = 0.05$ as the optimal after running a few trials as shown below (Fig 4).

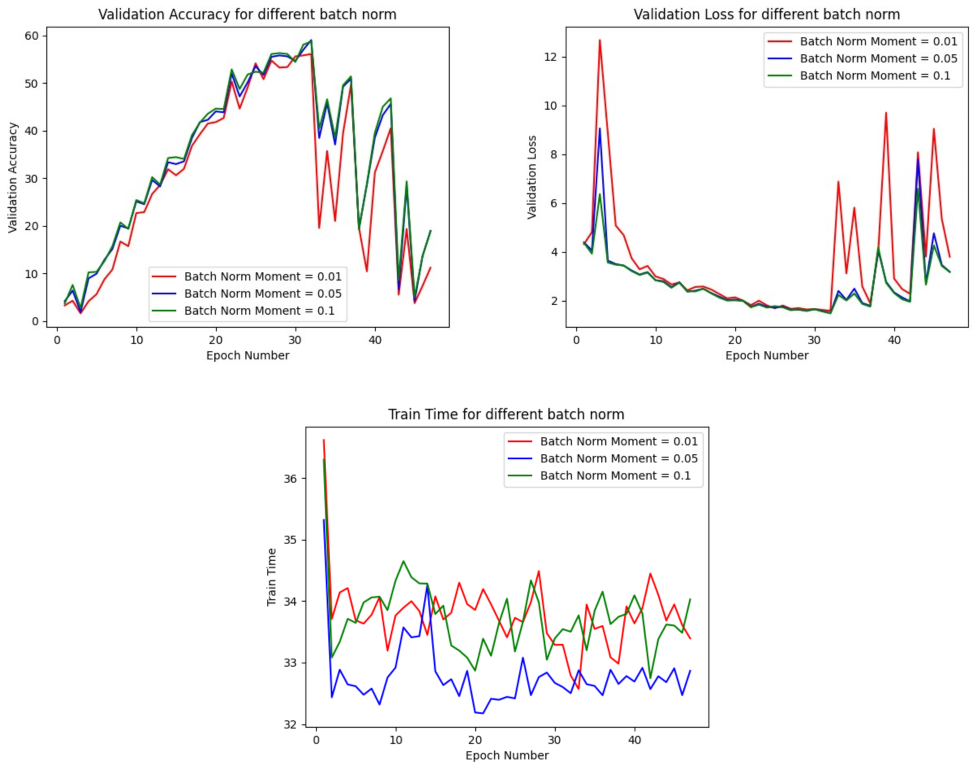

**Fig 4**

We utilise batch normalisation for the first layer, with $momentum = None$ instead of $momentum = 0.05$. This is because normalisation is applied before data augmentation and thus normalisation fails to take into account the effects of data augmentation, particularly those alterations that affect pixel colour values. By utilising batch normalisation with $momentum = None$ for the first layer, we take the overall lifetime average and variance instead of a moving average, and thus the effects of any data augmentation techniques will be applied.

We also removed the initial preprocessing step of normalising the dataset, since batch normalisation does the same thing, thus saving about 30 seconds in preprocessing time. For large-scale data, which takes long preprocessing time, or for live data, which we cannot preprocess since we have incomplete information, this approach might be more useful.

## Depthwise + Pointwise vs Standard Convolution

In the lectures, we learned that substituting standard convolutions with depthwise followed by pointwise convolution layers can potentially reduce training time and model complexity. However, contrary to expectations, the model's speed did not show the expected reduction (Fig 5), which was approximated to be around 1/9th of the original time. We theorise that this is due to:

I. A bottleneck in the data transformation pipeline
II. A result of the increased number of convolutional layers within the model, offsetting the expected efficiency gains.

As such we choose to not include depthwise + pointwise convolution into the final model.

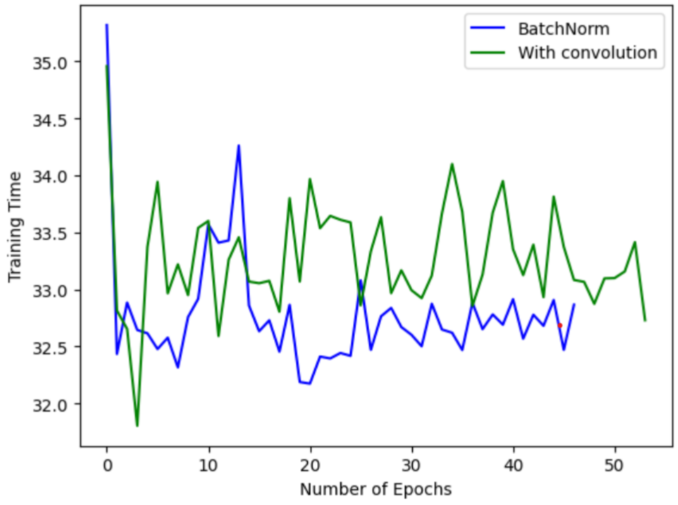

**Fig 5**

## One-Cycle Policy

The one-cycle policy is a method for adjusting the learning rate during training. It starts with a low learning rate that rises with each batch until it hits a peak. Following that, it is reduced with each batch until it returns to the low starting point, at which point the training concludes. The increase in learning rate at the start allows the model to explore a wide range of solutions while the decrease enables the model to converge on the most optimal solution that it has found. As stated in the original paper [6], one-cycle policy can lead to faster model convergence and better model generalisation.
We try a different approach to determine the maximum and minimum learning rate for one-cycle policy. Instead of training a model for each learning rate, we vary the learning rate with each rate and tie the training loss to the batch's learning rate. This allows us to determine the maximum learning rate with only one training, which helps save time at the expense of correctness.

The graph indicates that the optimal maximum learning rate is 0.001 (Figure 6a). A common approach is to set the minimum learning rate at one-tenth of the maximum, which in this case would be 0.0001. We then fix the training to stop at 30 epochs or when the early stopper condition activates (patience = 10).

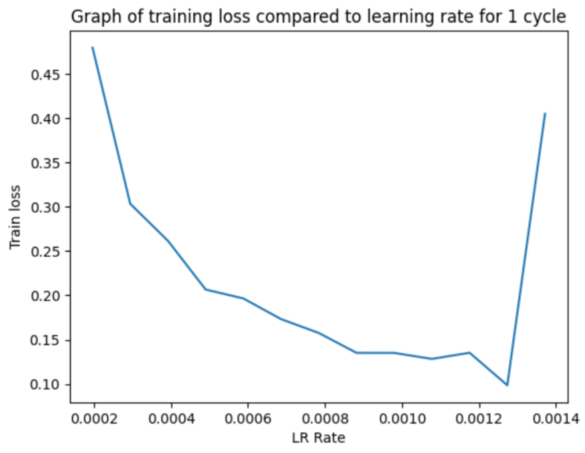

Fig 6a

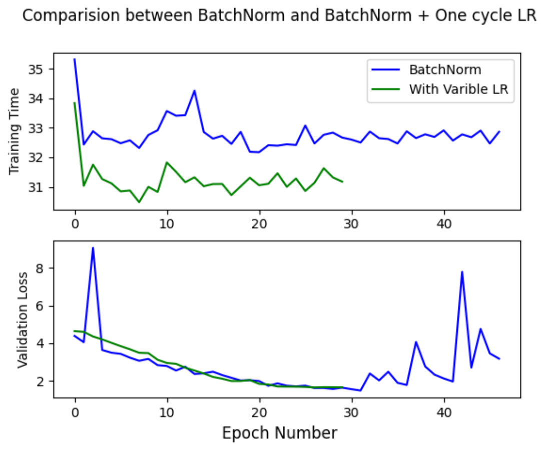

Fig 6b

One-cycle policy achieves around roughly the same validation loss at an earlier epoch. Additionally, the training time per epoch also decreases (Fig 6b). Overall, the one-cycle policy improves our model since it efficiently searches for the optimal solution and converges on it and hence we include it into the final model.

## Data Augmentation

### Random MixUp and CutMix

MixUp and CutMix are innovative data augmentation strategies that serve as regularisation methods to enhance model generalisation, rather than merely expanding the dataset's size and variability. These techniques manipulate the input data in distinct manners:

I. **MixUp** creates a new composite image by performing a weighted average of the pixel values from two different images, blending their classes in proportion to the mixing (Fig 7a).
II. **CutMix** adopts a different approach by cutting a section of one image and pasting it onto another, combining the context of one image with the local features of another (Fig 7b).

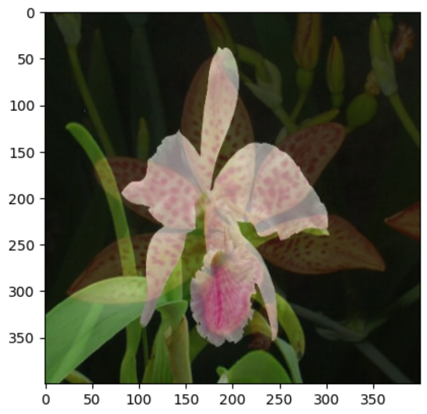

Fig 7a

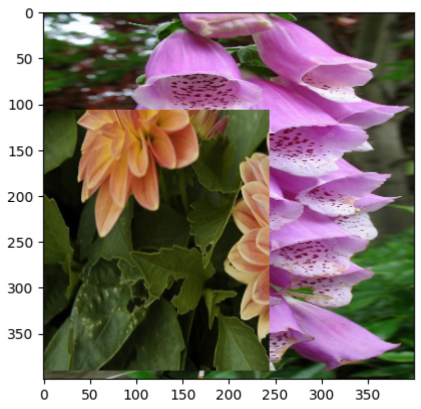

Fig 7b

We utilise both MixUp and CutMix for our model. For MixUp and CutMix, we use *alpha=0.2* and *alpha=1.0* to be the optimal, after some experimentation. We randomly choose MixUp or CutMix with *probability=0.5* and apply the chosen augmentation to a batch. Initially, we found that the dropout rate of 0.5 worsened the impact of MixUp and CutMix. Changing the dropout rate to 0.05 helped improve this. This suggests that too much regularisation can harm model learning.

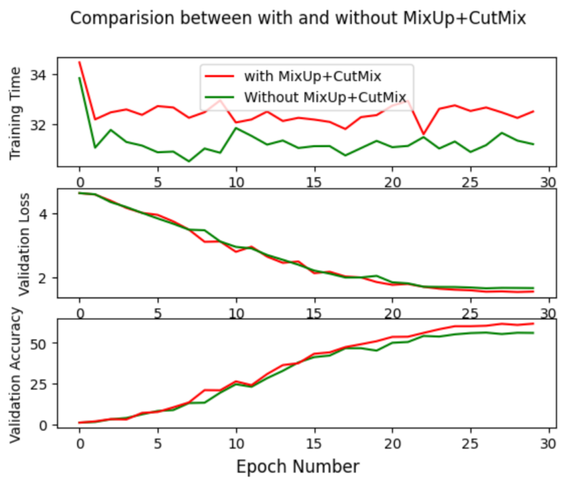

**Fig 8**

The training time takes 2 seconds longer per epoch and there is a minor improvement in validation loss and accuracy (Fig 8). As such, we choose to include MixUp + CutMix into the model.

## Copy Padding

When resizing the images, the aspect ratio can change if the ratio of the original image and the target size does not match (Fig 9). This could mean that some essential information about the image is lost in our current data augmentation pipeline.

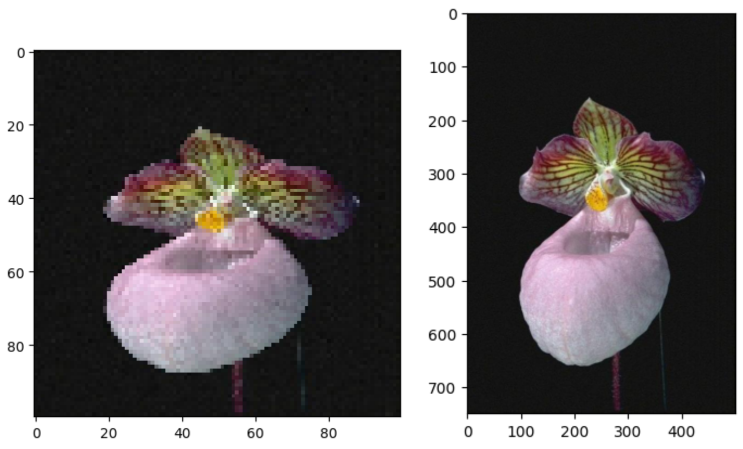

**Fig 9**

Therefore, we created a custom data augmentation method called “copy padding” to help reduce this loss in information (Fig 10). Given an image of size *(x, y)*, we want to pad the image to *(z, z)* where *z = max(x, y)*. This is preferable to the approach of cutting the image, since less information is lost. The regions will be padded with a cutout of the same image at the same region. This is better than filling up the target region with random noise, since we repeat the same regional information, which leads to lesser information loss.

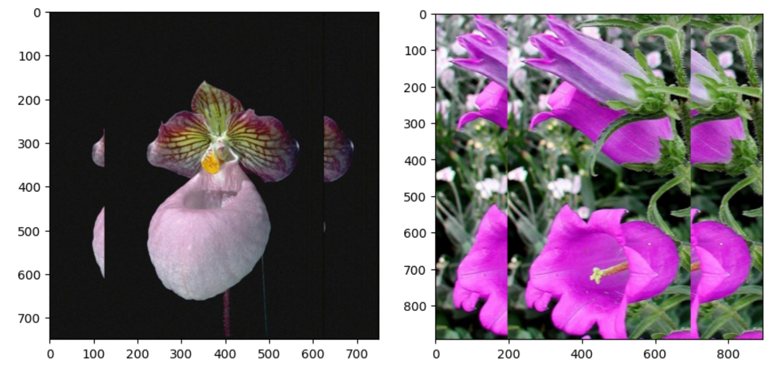
**Fig 10**

However, after using it with our dataset, there is very little increase in the accuracy. We theorise that this could be due to:

I. The ratio difference between the original images and the target size does not differ by that much
II. The convolutional layers, especially max pooling combined with local response normalisation, are insensitive to aspect ratio changes.

The training time per epoch increases by 10 seconds per epoch with very little upside. Therefore, we decided not to use copy-padding for our data augmentation.

## Standard Data Augmentation

We found Random Horizontal Flip, Random Color Jitter, Random Rotation and Random Resize Crop to generally improve the model by a significant amount.

The most impactful transformation is random colour jitter, and this might be because a majority of the flower images are taken in different weather conditions. Changing the brightness and contrast randomly for each batch might help the model learn about the different lighting conditions for images.

We found Random Rotation transformation to be the least impactful (loss/ accuracy) and most harmful to training time per epoch (doubles the training time). This might be because most flower images are rotation invariant. Moreover, matrix rotation is costly, especially when the image size is large.

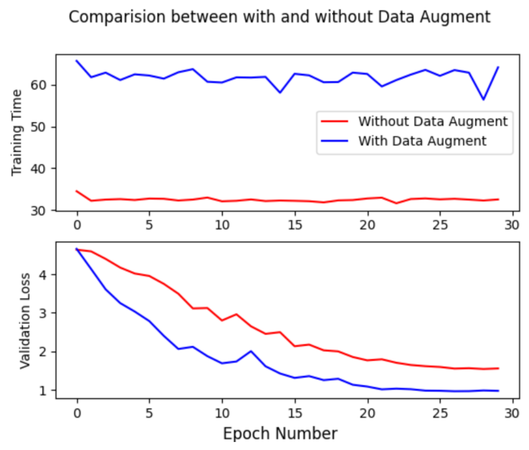

**Fig 11**

We apply the standard data augmentation method and this yields a much better validation loss curve However, the data augmentation time is costly, and doubles the training time per epoch (Fig 11). Ultimately, we decided to include standard data augmentation into the final model.

## Evaluation of Final Model

Combining the various techniques and the optimal hyperparameters we found, we train our final model. The following table summarises the improvements of our final model over the baseline model, as well as the AlexNet and GoogLeNet models by Gurnani et al [5].

| | Accuracy (Top 1) | Accuracy (Top 5) | F1 |
|---|---|---|---|
| AlexNet | 43.39% | 68.68% | - |
| GoogLeNet | 47.15% | 69.17% | - |
| Baseline Model | 18.9% | 45.9% | 0.176 |
| Final Model | 76.4% | 92.1% | 0.754 |

**Table 2**

Figure 12 compares the training and validation loss for the baseline and final model.

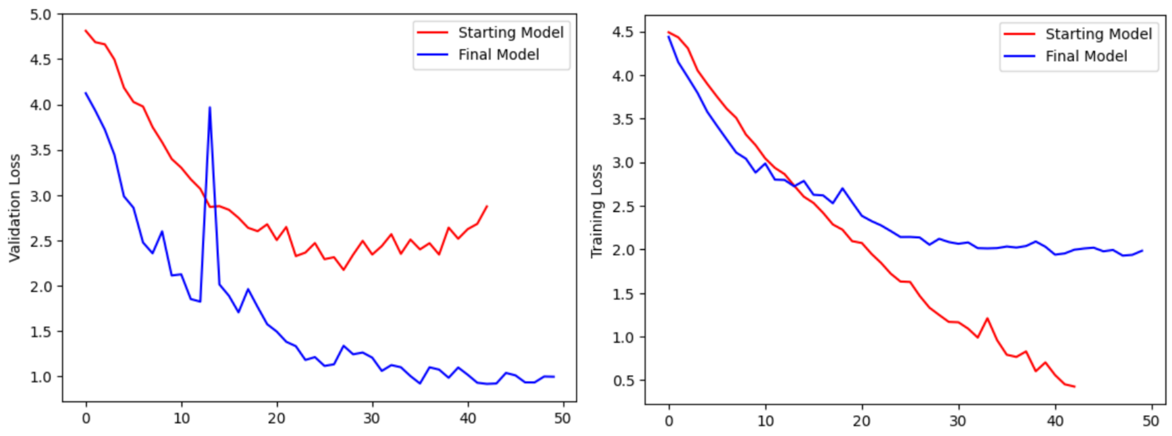

**Fig 12**

The validation loss decreased and training loss increased for the final model compared to the training model. Evidently, the regularisation techniques and optimisations done to the baseline model were effective in improving the model significantly, specifically reducing the overfitting problem of the baseline model. Thus, the overall Accuracy and F1 score of the final model on the test set were significantly improved.

## Discussion

To better understand why the final model performs better, we implemented Gradient-weighted Class Activation Mapping (Grad-CAM) [7]. This algorithm identifies which parts of an image influences the model's classification decision by calculating the gradients of the class score with respect to the last activation layer, pooling these gradients, and then using them to weigh the activation maps to reveal the most influential features. The corresponding gradients and activation maps are then converted into a heat map which is overlaid on top of the original image, to indicate the regions that influence the predictions the most. In Figure 13, we have a sample of the Grad-CAM map for both the baseline and final models when predicting from the test set.

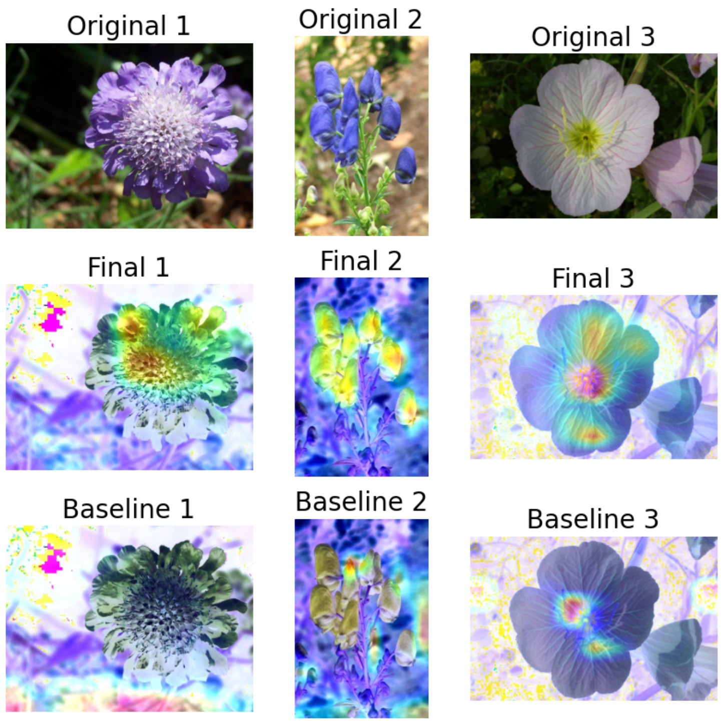

**Fig 13**

As evident in the heatmaps, it is clear to see why the final model performs significantly better than the baseline model. It demonstrates that the final model considers a more extensive array of relevant features relative to the baseline. This is indicated by the broader and more pronounced heatmap area along the relevant features of the flowers, underscoring these regions as significant in influencing the models' predictions.

Overall, we found Batch Normalisation, One Cycle Policy and standard data augmentation techniques to be the most impactful in terms of improving training time and/or model accuracy. Batch Normalisation and One Cycle Policy are methods which can generally be applied to all types of neural networks, while standard data augmentation methods depend on the dataset itself. Moreover, regularisation techniques such as MixUp, CutMix, and dropout layers in the fully-connected layer helps the model generalise better and achieve better results. However, too much regularisation can be harmful to model learning, especially when paired with a high learning rate (One Cycle Policy's peak). Lastly, we learnt that the aspect ratio difference between the original and the target image might not matter as much as we initially thought, and this might be due to max pooling combined with local response normalisation, in the original model.

## Conclusion

In conclusion, this project successfully improved the CNN model implemented by Liu et al. on the Oxford 102 Flower Dataset through a series of regularisation techniques and optimisation. The final model notably outperformed the baseline model by over threefold in both accuracy and F1 scores. Moreover, Grad-CAM visualisations confirmed that the final model effectively focuses on more relevant features for classification. Thus, it is clear that meticulous regularisation and optimisation can lead to remarkable enhancements in model performance.

## References

[1] Y. Liu, F. Tang, D. Zhou, Y. Meng, and W. Dong, “Flower classification via convolutional neural network,” in *2016 IEEE International Conference on Functional-Structural Plant Growth Modeling, Simulation, Visualization and Applications (FSPMA)*, Qingdao, China: IEEE, Nov. 2016, pp. 110–116. doi: 10.1109/FSPMA.2016.7818296.

[2] M.-E. Nilsback and A. Zisserman, “Automated Flower Classification over a Large Number of Classes,” in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, Dec. 2008, pp. 722–729. doi: 10.1109/ICVGIP.2008.47.

[3] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” arXiv, Mar. 02, 2015. doi: 10.48550/arXiv.1502.03167.

[4] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, “mixup: Beyond Empirical Risk Minimization.” arXiv, Apr. 27, 2018. doi: 10.48550/arXiv.1710.09412.

[5] A. Gurnani, V. Mavani, V. Gajjar, and Y. Khandhediya, “Flower Categorization using Deep Convolutional Neural Networks.” arXiv, Dec. 08, 2017. doi: 10.48550/arXiv.1708.03763.

[6] L. N. Smith and N. Topin, “Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates.” arXiv, May 17, 2018. Accessed: Nov. 10, 2023. [Online]. Available: http://arxiv.org/abs/1708.07120

[7] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization.” Dec. 02, 2019. doi: 10.1007/s11263-019-01228-7.