Flower Classification Using Convolutional Neural Networks

V.T.M. Dinh, W.T.E. Huijbers, Y.Y. Ma, T.H.D. Nguyen, N. Preciado

Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Abstract

Despite the importance of flowers in many areas, flower identification is often a difficult task for humans due to many flower species sharing similar characteristics. Four CNN architectures were compared against each other and a baseline to determine their performance in classifying flower images: a randomly assembled architecture, GoogLeNet, ResNet50 and DenseNet121. The classification was performed on a dataset consisting of 9157 images of 20 different flower classes, split in a 0.7/0.15/0.15 ratio for training, validation and testing. Training of the models was terminated when the validation accuracy increased by less than 5% in 15 consecutive epochs or the limit of 120 epochs was reached. For the classification of flowers, DenseNet121 yielded the highest overall performance in accuracy (0.7) and macro-average AUC (0.892). However, ResNet50 and GoogLeNet performed better than DenseNet121 in the identification of specific classes.

1. Introduction

About one in every three bites taken from a fruit or vegetable comes from a flower that requires pollination [1]. The total economic value of these flowers was 153 billion euros in just 2009 [2]. Flower taxonomy, or the study of identifying, naming, and classifying flowers, helps identify pollinators beneficial to an area [3,4].

Identifying flowers can prove difficult for non-florists investigating different areas, due to the wide variety of species, which are often specific to a certain ecosystem. Therefore, identification support in the form of artificial intelligence (AI) can possibly prove useful for people unfamiliar with the local flower species. However, automatic flower classification remains a difficult task to solve, as flower species share similar characteristics, such as colour and shape. Additionally, the number of flowering plants known exceeds 250,000 and are classified into 350 families [5].

Compared to classical machine learning methods, deep learning methods such as the use of convolutional neural networks (CNNs) have proven to be more successful in solving the flower classification problem [5,6]. To decrease the training time of such networks, GPUs are often utilised as they improve the efficiency of training through the multi-core structure of the processor. As this project requires training models on image data, GPUs can assist in accelerating the training process [7]. In addition to improvements in hardware, the success of CNNs can largely be attributed to developments in network structures. An example of a promising network is the GoogLeNet (codenamed Inception V1) architecture, which won the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2014) in both the "image classification with provided data" and "object detection with external data" competitions [8]. The GoogLeNet network makes use of twelve times fewer parameters than the winning architecture proposed by Krizhevsky *et al*. only two years earlier, while its accuracy was found to be significantly higher [9,10].

Residual Network (ResNet) is an architecture that won ILSVRC classification competition in the consecutive year with an error of only 3.57% [8]. The architecture distinguished itself by implementing identity shortcuts, reducing vanishing gradients in deep neural networks [11]. Likewise, Densely Connected Convolutional Networks (Densenet) reduce vanishing gradients while encouraging feature propagation and reuse. However, concatenation is used instead of summation for connections helping layer progression [11].

In this study, CNN classifier models based on a random network, GoogLeNet, ResNet and DenseNet are used to identify different flower types in the publicly available Kaggle dataset "Petal to The Metal", which contains image data for 104 flower species native to North America[1]. Due to the scope of the study project and the limitation of hardware resources, only the 20 most frequently occurring flower classes are included in training and testing the models. The purpose of this study is to determine which network architecture is the most suitable for the classification of flower images.

[1] https://www.kaggle.com/c/tpu-getting-started/data

2. Methods

2.1. Raw Dataset

The 'Petals to the Metal - Flower Classification on TPU' dataset includes 104 different flower species and four subsets with different resolutions, namely 192x192, 224x224, 331x331 and 512x512 pixels. The images in the training and validation sets were labelled with their respective flower class and did not contain any duplicates, missing values or NaN values. The provided images in the test set, however, were unlabelled and therefore disregarded. Additionally, there was a high class imbalance between some of the classes, as can be seen in ***Figure A1*** and ***Table A1*** in the Appendix.

2.2. Data Preprocessing

For training purposes, the dataset of resolution 224x224 was used. The raw dataset was already divided into training, validation and testing sets of respectively 12753, 3712 and 7382 instances. A class imbalance was observed in the distribution of the data, resulting in a low number of training images for some flower classes. Since this could result in a low accuracy of the CNN during testing, it was decided to disregard classes for which the number of training images provided was deemed too low.

Instances belonging to these classes were removed from the training and validation subsets. As a result, the 20 most frequently appearing classes remained in the dataset. The remaining dataset was then split with a 0.7/0.15/0.15 ratio into training, validation and testing sets, respectively [6]. These numbers are optimised[2] based on experiments, as well as referred to general method of data splitting. As a result, the training, validation and test sets consisted of 6409, 1373 and 1375 images of flowers, respectively. The different classes and their frequency within the dataset of the resolution of 224x224 pixels are shown in ***Figure 1***.

To improve generalisation by the network, synthetic data was included through data augmentation on the training set by flipping images to the left or the right, and changing their saturation [12]. This process was executed randomly for both each image in the dataset, and each epoch in training.

2.3. Network Architecture

2.3.1. Random Network

In the initial experiments, a random network architecture was built and trained. This network was constructed arbitrarily, stacking random convolutional and dense layers on top of each other. All layers were followed by a ReLu activation, with the last layer being the only exception, having a Softmax activation for the classification task. The aim of this practice was to show that the performance of CNNs on the flower classification task will differ based on the architecture. The total number of network parameters was 226,036, all trainable parameters. In ***Table 1***, the architecture of the Random network is shown.

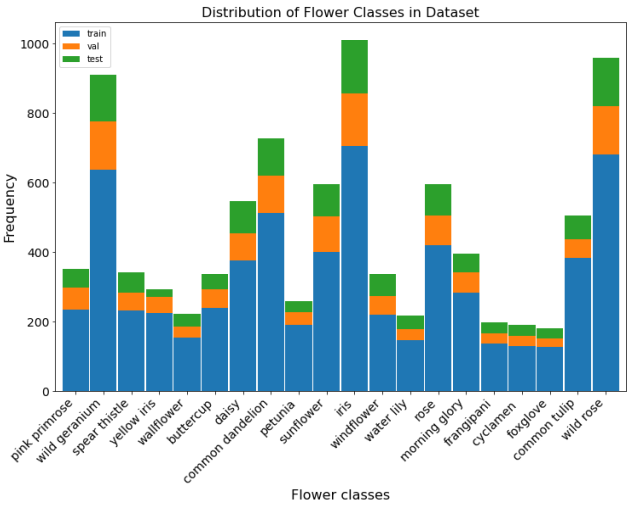


Figure 1. Distribution of flower classes in the dataset.

Table 1. Architecture of the Random network.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 16)	448
max_pooling2d (MaxPooling2D)	(None, 112, 112, 16)	0
conv2d_1 (Conv2D)	(None, 112, 112, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 32)	0
conv2d_2 (Conv2D)	(None, 56, 56, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 32)	0
conv2d_3 (Conv2D)	(None, 28, 28, 32)	9248
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 32)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 32)	200736
dense_1 (Dense)	(None, 32)	1056
dense_2 (Dense)	(None, 20)	660

[2]https://vitalflux.com/machine-learning-training-validation-test-data-set/

2.3.2. *GoogLeNet*

The GoogLeNet network is based on the idea of finding the optimal local construction, which can then be repeated spatially [9]. The network contains several "Inception modules", shown in ***Figure 2(b)***, stacked upon each other with occasional max-pooling layers with stride 2, which halves the resolution of the grid. The inception modules help with mitigating computation difficulties as the network gets deeper (i.e. more layers are appended). An intuitive interpretation of the inception layer is that "visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously". The inception layer resolved the computational requirements of the network, yet for deep neural networks, vanishing gradients also posed a problem; GoogleNet addresses this problem by adding two auxiliary output layers in the middle of the network to help gradients propagate to lower layers of the network.

Counting only layers with parameters, GoogLeNet is 22 layers deep, or 27 layers if pooling layers are also counted. The number of independent building blocks is roughly 100. In ***Figure 3,*** a schematic overview of the GoogLeNet network is shown [9].
The total number of network parameters was 7,063,892, while the number of trainable and non-trainable parameters (Batch Normalisation layers) was 6,982,420 and 81,472, respectively. The code for the construction of the network was inspired by *AI in Plain English.*[3]

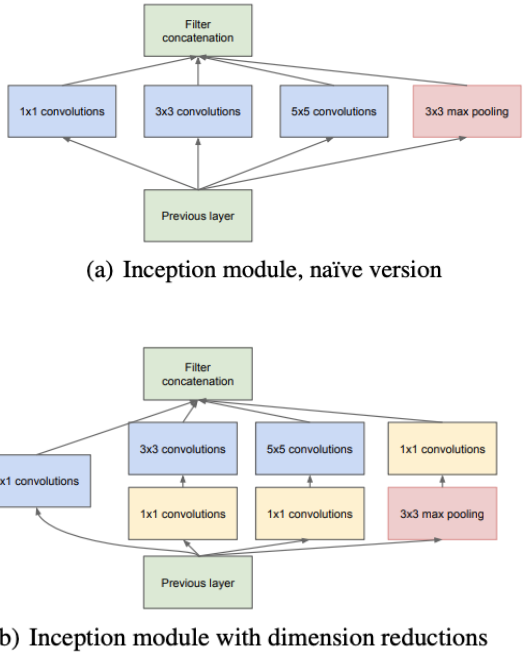
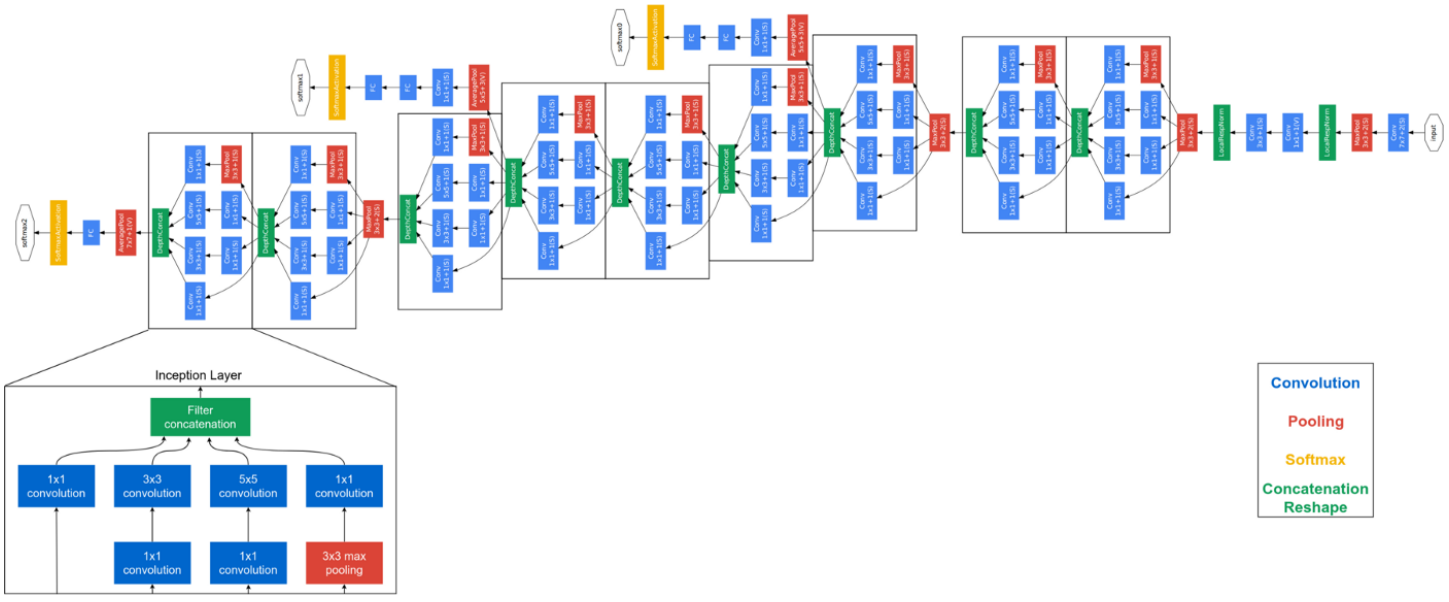


Figure 2. Inception module.

2.3.3. *ResNet50*

Residual Networks (ResNets) differ from a plain layered network through the implementation of identity connections which reduce the vanishing gradient problem, often occurring in deep neural networks [13]. Identity connections add the input to a previous layer to the output of a subsequent layer, which prevents the weights from becoming too small. The identity mappings also solve the degradation problem common to deep neural networks, where increasing the network depth has an adverse effect on the model's performance.

Figure 3. Schematic overview of GoogleNet network



[3]https://ai.plainenglish.io/googlenet-inceptionv1-with-tensorflow-9e7f3a161e87

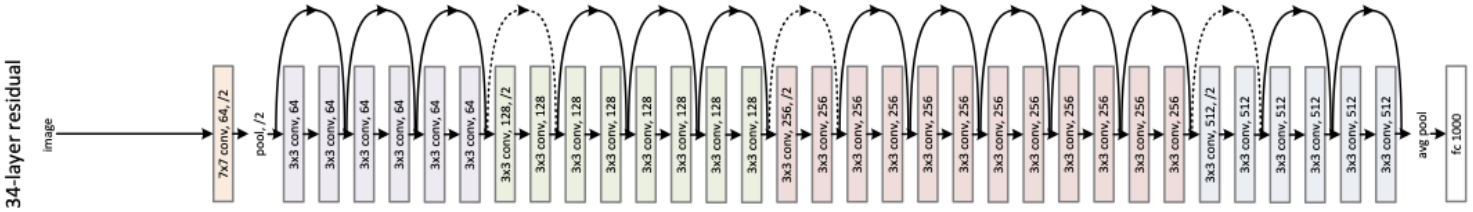


Figure 4. Schematic overview of the ResNet network.

Intuitively, when it is optimal to preserve the output of the previous layer, the network will drive the weights of the "skipped" layer to zero. This behaviour ensures that the deep neural network would only be as bad as their shallower counterparts. However, these identity connections can only be made when the input and output are of the same dimensions. Therefore, dimensions are either matched through zero-padding to retain dimensions during convolutions, or through 1x1 convolutions with non-one strides to reduce dimensionality. In ***Figure 4,*** a schematic overview of the ResNet architecture is shown. The total number of network parameters was 24,243,092, while the number of trainable and non-trainable parameters (Batch Normalisation layers) was 24,189,972 and 53,120, respectively. The code for the construction of the network was inspired by *Towards Data Science.*[4]

2.3.4. DenseNet

The Dense Convolutional Network (DenseNet) architecture connects each layer to every other layer in a feed-forward fashion [11]. Each layer uses the feature-maps of preceding layers as inputs and feeds its own feature-maps as inputs to subsequent layers, shown schematically in ***Figure 5***. In contrast to ResNets, features are combined by concatenation, instead of through summation before they are passed into a layer. Compared to traditional CNNs with L layers and L connections, a DenseNet network with L layers has $L(L+1)/2$ connections. Additionally, these connections ensure that information from lower layers are preserved throughout the subsequent layers, reducing the number of parameters compared to traditional CNNs, as relearning redundant feature-maps is not necessary. Another advantage is the improved gradient flow throughout the network, resulting in an easily trainable model. The dense connectivity pattern was also reported to reduce overfitting on tasks in which training sets are small.

The total number of network parameters was 7,063,892, while the number of trainable and non-trainable parameters (Batch Normalisation layers) was 6,982,420 and 81,472, respectively. The code for the construction of the network was inspired by *Towards Data Science.*[5]

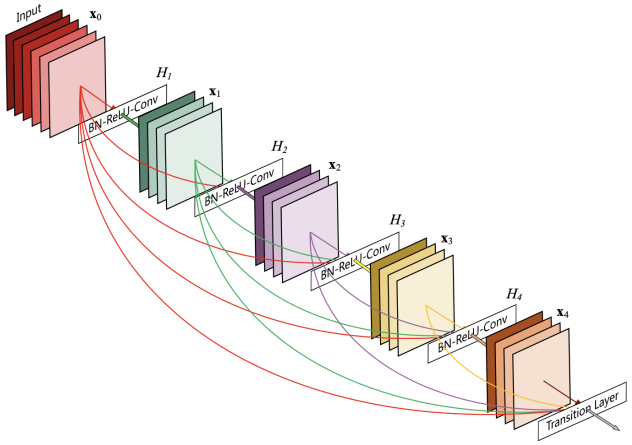


Figure 5. Schematic overview of a 5-layer DenseNet block with a growth rate of k = 4.

2.3.5. Training

The number of epochs during which training was performed was variable for each model, as it was dependent on the accuracy of the validation set. Termination of training occurred either when the validation accuracy stopped increasing by more than 5% for 15 consecutive epochs, or when 120 epochs had passed. Then, the model chose the weights which yielded the highest validation accuracy as its parameters.

3. Results

3.1. Accuracy

The majority class classifier, which assigns all instances the most prevalent class, was used as the Baseline to compare the other models to. In ***Table 2***, the accuracy values with 95% confidence intervals (CI) are tabulated for the Random, GoogLeNet, ResNet50 and DenseNet121 networks and compared to the baseline.

[4]https://towardsdatascience.com/understand-and-implement-resnet-50-with-tensorflow-2-0-1190b9b52691

[5]https://towardsdatascience.com/creating-densenet-121-with-tensorflow-edbc08a956d8

From ***Figure 6,*** it can be observed that the bar graph shows no overlap between the Confidence Interval of each model's accuracy, implying significant differences between most of the models' performance with DenseNet121's accuracy of 0.700 being the highest and all models are better than the Baseline. No conclusions can be derived for the performance of the Random and ResNet50 networks.

Table 2. Accuracy with 95% CI and macro-average AUC score.

Network	Accuracy	95% CI MOE	ROC AUC Score	Epochs
Baseline	0.112	0.017	0.5	-
Random	0.492	0.026	0.694	13
GoogLeNet	0.587	0.026	0.750	48
ResNet50	0.501	0.026	0.713	60
DenseNet121	0.700	0.024	0.829	32

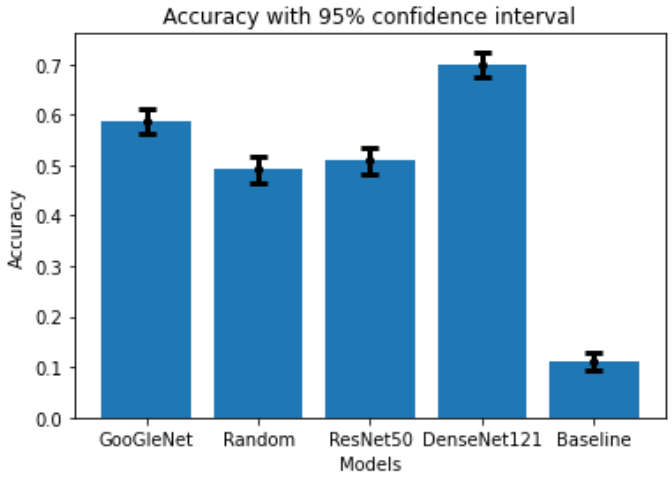


Figure 6. Accuracy with 95% CI.

The loss and accuracy plots of each network is shown in ***Figure A2-A5*** in the Appendix.

3.2. ROC Curves

The macro-average ROC AUC scores are tabulated for each network in ***Table 2***. The One - All AUC scores for each class are tabulated in ***Table 3,*** while the One - All ROC curves are shown in ***Figure 7***. It can be observed that DenseNet121 outperforms all other models on the entire dataset. Yet, this is not the case for individual classes. For the classification of specific classes, the performance of other models is better in some cases when measured in AUC: ResNet50 for identifying 'Daisy' (0.83 > 0.81), and GoogLeNet for recognizing 'Wild geranium' (0.84 > 0.82), 'Buttercup' (0.88 > 0.79), 'Sunflower' (0.89 > 0.87) and 'Iris' (0.90 > 0.89).

The lowest AUC measured for each model was the 'Common tulip' for DenseNet121 (0.71), the 'Cyclamen' for GoogLeNet (0.56) and 'Frangipani' for ResNet50 (0.61) and the Random model (0.51).

Table 3. One - All AUC scores.

One - All AUC scores				
	Random	GoogLeNet	ResNet50	DenseNet121
Pink primrose	0.57	0.77	0.69	0.81
Wild geranium	0.71	0.84	0.76	0.82
Spear thistle	0.72	0.80	0.68	0.89
Yellow iris	0.64	0.71	0.69	0.85
Wallflower	0.75	0.79	0.77	0.83
Buttercup	0.65	0.88	0.71	0.79
Daisy	0.74	0.78	0.83	0.81
Common dandelion	0.80	0.76	0.76	0.88
Petunia	0.64	0.63	0.62	0.78
Sunflower	0.78	0.89	0.76	0.87
Iris	0.84	0.90	0.79	0.89
Windflower	0.69	0.70	0.66	0.81
Waterlily	0.65	0.63	0.63	0.92
Rose	0.76	0.69	0.78	0.87
Morning glory	0.71	0.75	0.63	0.81
Frangipani	0.51	0.70	0.61	0.82
Cyclamen	0.60	0.56	0.70	0.75
Foxglove	0.65	0.71	0.75	0.79
Common tulip	0.67	0.72	0.69	0.73
Wild rose	0.78	0.79	0.76	0.87

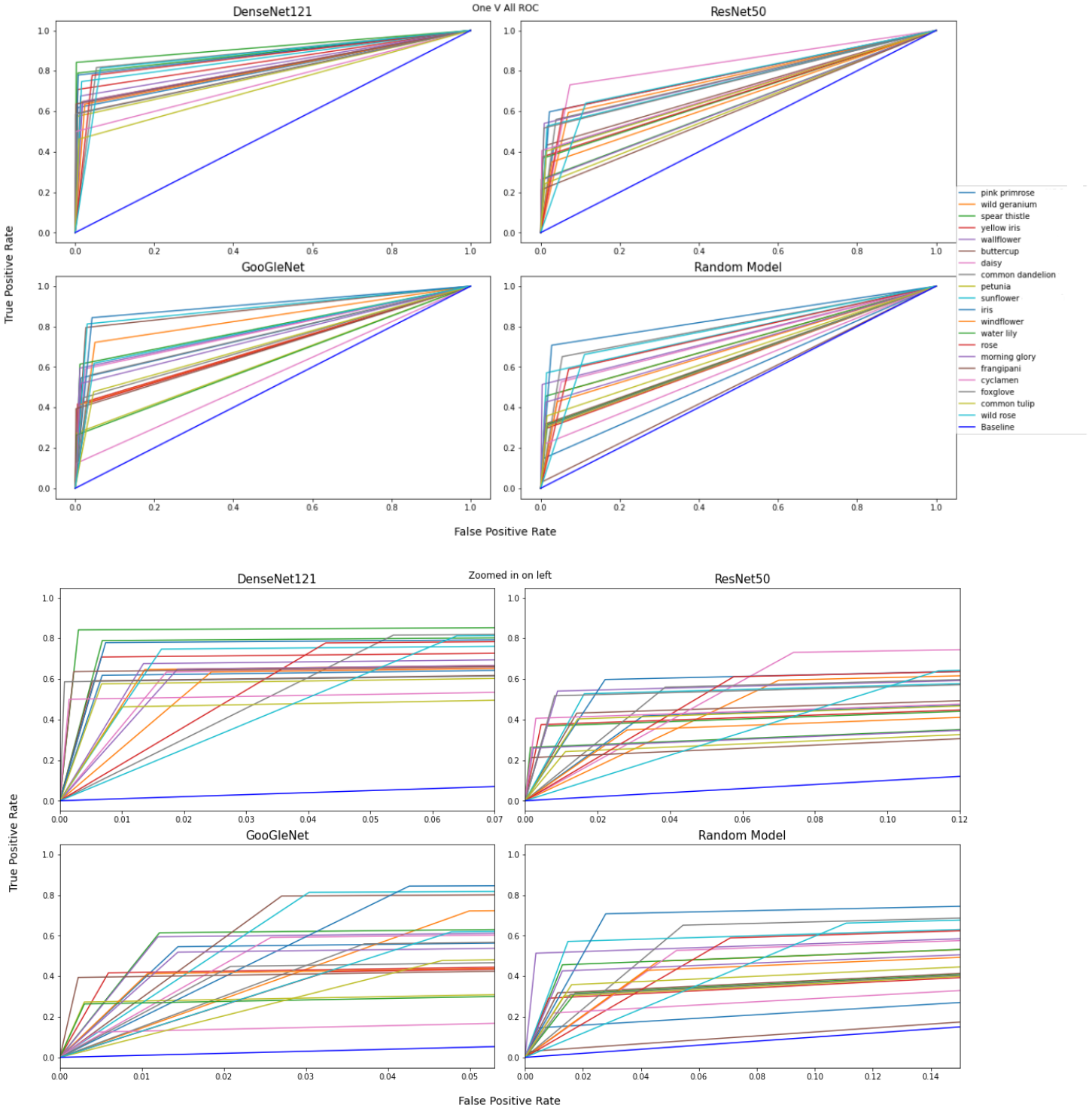


Figure 7. One - All ROC curves for each network.

In ***Figure 9*** and ***Figure 10***, the confusion matrices of the Random, GoogLeNet, ResNet50 and DenseNet121 networks and the baseline are shown. Here, the y-axis represents the true class, while the x-axis represents the predicted class.

The confusion matrix of the overall best-performing model, DenseNet121, further affirms the fact that the model does not suffer from class imbalance: it does not predict the most prominent classes for other classes. This behaviour is relatively consistent across other models as well. A reasonable explanation for this would be that there is sufficient training data for each class, especially the three most frequently occurring classes. We previously stated that ResNet50 should be used for identifying 'Daisy', yet it can be observed in the confusion matrix that even though ResNet50 is well-suited for the task, the false positive rate for the aforementioned class is still relatively high.

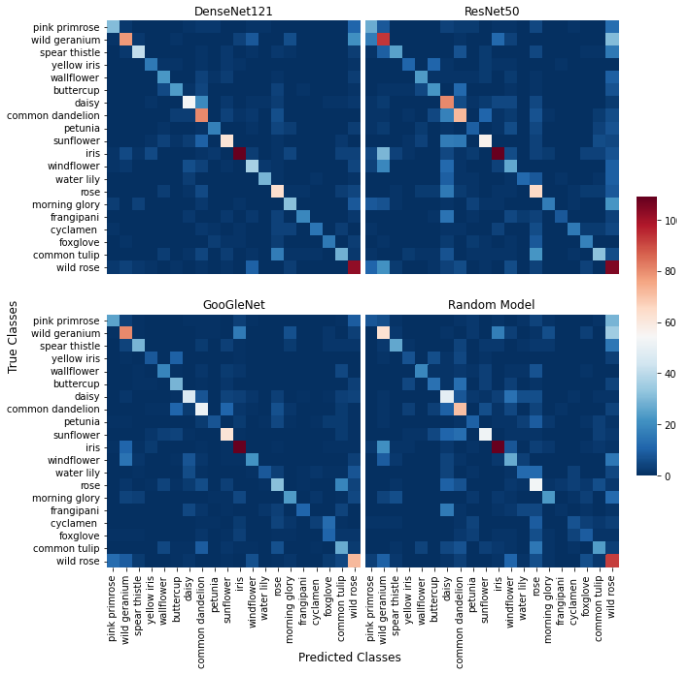


Figure 9. Confusion matrices of Random, GoogLeNet, ResNet50 and DenseNet121 networks.

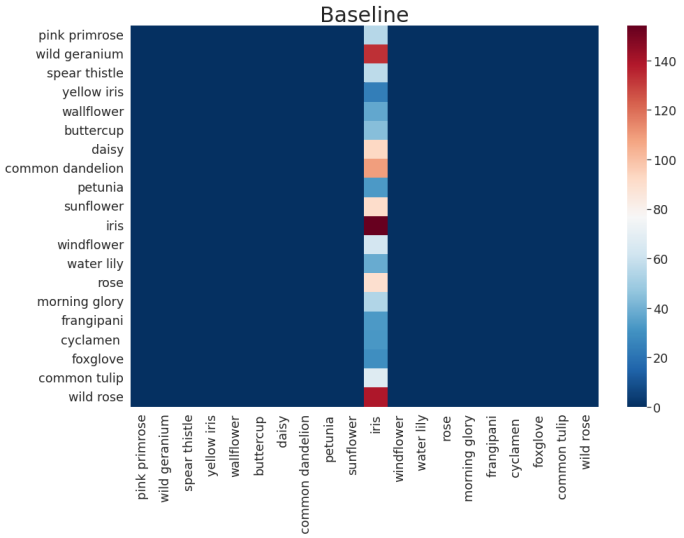


Figure 10. Confusion matrix of baseline.

4. Discussion

While all models performed significantly better than the baseline, only DenseNet121 stood out in its accuracy and AUC performance. An explanation for this performance could be the way in which features are processed in DenseNet121. Each layer gets its input from all previous layers, which enables the subsequent layers to be able to process both abstract and detailed information simultaneously. This can be thought of as looking at a picture from both a close distance and from far away at the same time, which could be verified by extracting kernels from the network and visualising them, which will elucidate which kernel from which layer extracts what information.

The random model and ResNet50 had a similar performance, which was surprising as ResNet50 is a well-known model. This difference could be caused by many factors, such as the use of a different dataset, batch size, optimiser and learning rate. Additionally, the random model and ResNet50 both performed the worst on the class ‘Frangipani’. This raises the question of whether the implementation of ResNet50 was performed correctly.

Despite some classes being better recognised than others overall, there was no class on which the AUC was the lowest or highest across all models. This can be explained by a low variance within the classes, meaning that all images within a single class contained similar features.

The data preprocessing technique grayscaling eliminates colour features and places more focus on shapes and textures in the images data and can reduce training time [14]. In a trial-run using the random network, the model was tested on grayscale and the original RGB image data. However, it was observed that a lower accuracy was obtained due to the missing colour features when grayscale data was used. It appeared that flower classification requires both colour and texture features to obtain a good model performance [15]. Therefore, RGB data was selected for further experiments. As the use of RGB data still results in a longer training time of the model, it may be worthwhile to investigate whether different grayscaling methods have different effects on the accuracy.

Due to the limitation of hardware resources and the complexity of data, it took a significant amount of time to train models with default settings. This was especially true of DenseNet121, requiring around 20 minutes per epoch and 36 hours in total. The solutions found were using GPUs, splitting and saving the dataset in TFRecord format, and adding a ‘stop_function’ in the training processes. As GPUs consist of more processing cores than CPUs, utilising the former would tremendously reduce training time, since deep neural networks can take advantage of parallelisation (during the forward pass of each batch). Additionally, splitted data was saved in TFRecord format to allow for reproducibility, as well as avoiding repeated dataset shuffles during training, which further reduced training time. Finally, with a ‘stop_function', our models would terminate at a

suitable epoch, which not only avoids wasting time by training without performance increase, but also avoids overfitting.

5. Conclusion

Overall, DenseNet121 exhibited the highest performance on the processed Kaggle dataset. Other models, namely ResNet50 and GoogLeNet can perform better than DenseNet121 when it comes to identifying certain classes. None of the models suffered from class imbalance, all performing better than the baseline in this specific classification task.

6. References

[1] A.-M. Klein et al., "Importance of pollinators in changing landscapes for world crops," Proceedings of the Royal Society B: Biological Sciences, vol. 274, no. 1608, pp. 303–313, Feb. 2007, doi: 10.1098/rspb.2006.3721.

[2] N. Gallai, J.-M. Salles, J. Settele, and B. E. Vaissière, "Economic valuation of the vulnerability of world agriculture confronted with pollinator decline," Ecological Economics, vol. 68, no. 3, pp. 810–821, Jan. 2009, doi: 10.1016/j.ecolecon.2008.06.014.

[3] A. M. Streich and K. A. Todd, "Classification and Naming of Plants," Institute of Agriculture and Natural Resources at the University of Nebraska–Lincoln, Jan. 2014. [Online]. Available: https://alec.unl.edu/documents/cde/2017/natural-resources/classification-and-naming-of-plants.pdf

[4] A. E. Martins, M. G. G. Camargo, and L. P. C. Morellato, "Flowering Phenology and the Influence of Seasonality in Flower Conspicuousness for Bees," Frontiers in Plant Science, vol. 11, Feb. 2021, doi: 10.3389/fpls.2020.594538.

[5] H. Hiary, H. Saadeh, M. Saadeh, and M. Yaqub, "Flower classification using deep convolutional neural networks," IET Computer Vision, vol. 12, no. 6, pp. 855–862, Sep. 2018, doi: 10.1049/iet-cvi.2017.0155.

[6] J. J. Faraway, "Does data splitting improve prediction?," Statistics and Computing, vol. 26, no. 1–2, pp. 49–60, Oct. 2014, doi: 10.1007/s11222-014-9522-9.

[7] S. I. Baykal, D. Bulut, and O. K. Sahingoz, Comparing deep learning performance on BigData by using CPUs and GPUs. Istanbul, Turkey: IEE, 2018, pp. 1–6. doi: 10.1109/EBBT.2018.8391429.

[8] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge," International Journal of Computer Vision, vol. 115, no. 3, pp. 211–252, Apr. 2015, doi: 10.1007/s11263-015-0816-y.

[9] C. Szegedy et al., "Going deeper with convolutions," Sep. 2014. [Online]. Available: https://arxiv.org/pdf/1409.4842.pdf

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Communications of the ACM, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[11] G. Huang, Z. Liu, L. Van Der Maaten, and K. Weinberger, "Densely Connected Convolutional Networks," Jan. 2018. [Online]. Available: https://arxiv.org/pdf/1608.06993.pdf

[12] R. Poojary, R. Raina, and A. Kumar Mondal, "Effect of data-augmentation on fine-tuned CNN model performance," IAES International Journal of Artificial Intelligence (IJ-AI), vol. 10, no. 1, p. 84, Mar. 2021, doi: 10.11591/ijai.v10.i1.pp84-92.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image [6]N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and Evaluation of the First Tensor Processing Unit," IEEE Micro, vol. 38, no. 3, pp. 10–19, May 2018, doi: 10.1109/mm.2018.032271057. "Recognition," Dec. 2015. [Online]. Available: https://arxiv.org/pdf/1512.03385.pdf

[14] Y. Kim and T. S. Yun, "How to classify sand types: A deep learning approach," Engineering Geology, vol. 288, p. 106142, Jul. 2021, doi: 10.1016/j.enggeo.2021.106142.

[15] R. H. Shaparia, N. M. Patel, and Z. H. Shah, "Flower Classification using Texture and Color Features," Kalpa Publications in Computing, vol. 2, pp. 113–118, Feb. 2017, [Online]. Available: https://easychair.org/publications/open/jKTm#:~:text=Texture%20feature%20and%20color%20feature

Appendix

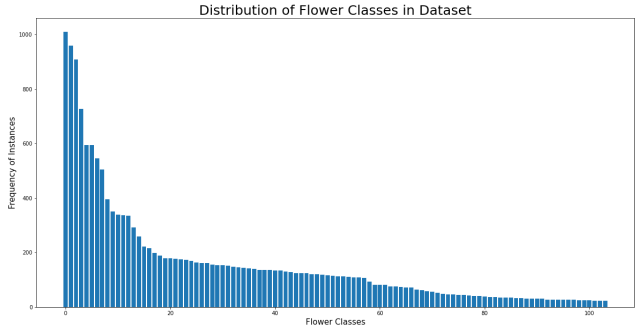


Figure A1. Sorted distribution of flower classes in the original dataset (104 classes).

Table A1. Distribution of flower classes in the dataset.

Flower Class	Training (Total: 6409)	Validation (Total: 1373)	Testing (Total: 1375)
pink primrose	234	62	55
wild geranium	635	140	133
spear thistle	232	51	57
yellow iris	223	45	24
wallflower	152	33	37
buttercup	239	54	44
daisy	374	78	93
common dandelion	512	106	109
petunia	189	37	33
sunflower	399	104	91
iris	704	152	154
windflower	219	54	63
water lily	145	33	38
rose	418	86	90
morning glory	282	59	54
frangipani	135	30	33
cyclamen	129	28	32
foxglove	126	25	29
common tulip	382	55	67
wild rose	680	140	139

Table A2. GoogLeNet architecture.

Type	Patch size/stride	Output Size	Depth	#1 x 1	# 3 x 3 Reduce	# 3 x 3	# 5 x 5 Reduce	#5 x 5	Pool Proj	Params	Ops
convolution	7 x 7 / 2	112 x 112 x 64	1							2.7K	34M
max pool	3 x 3 / 2	56 x 56 x 64	0								
convolution	3 x 3 / 1	56 x 56 x 192	2		64	192				112K	360M
max pool	3 x 3 / 2	28 x 28 x 192	0								
inception (3a)		28 x 28 x 256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28 x 28 x 480	2	128	128	192	32	96	64	380K	304M
max pool	3 x 3 / 2	14 x 14 x 480	0								
inception (4a)		14 x 14 x 512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14 x 14 x 512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14 x 14 x 512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14 x 14 x 528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14 x 14 x 832	2	256	160	320	32	128	128	840K	170M
max pool	3 x 3 / 2	7 x 7 x 832	0								
inception (5a)		7 x 7 x 832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7 x 7 x 1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7 x 7 / 1	1 x 1 x 1024	0								
dropout (40%)		1 x 1 x 1024	0								
linear		1 x 1 x 1000	1							1000K	1M
softmax		1 x 1 x 1000	0								

Table A3. ResNet architecture. Building blocks are shown in brackets, with the number of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

Layer Name	Output Size	ResNet50
conv1	112 x 112	7 x7, stride 2
conv2_x	56 x 56	3 x 3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times3$
conv3_x	28 x 28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times4$
conv4_x	14 x 14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times6$
conv5_x	7 x 7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times3$
	1 x 1	average pool, 1000-d fc, softmax
FLOPs		3.8×10^9

Figure A2. Loss and accuracy of the Random network.

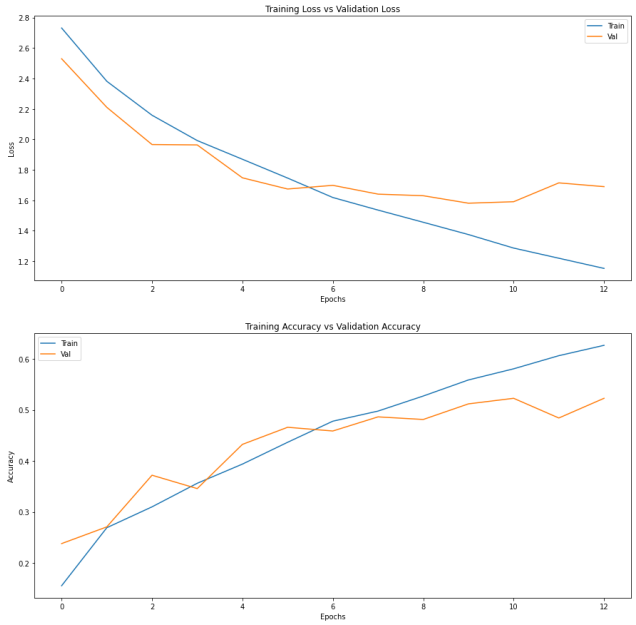


Table A4. DenseNet architecture. The growth rate for the networks is k = 32. Note that each "conv" layer in the table corresponds to the sequence BN-ReLU-Conv.

Layers	Output Size	DenseNet121
convolution	112 x 112	7 x 7 conv, stride 2
pooling	56 x 56	3 x 3 max pool, stride 2
dense block (1)	56 x 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
transition layer (1)	56 x 56	1 x 1 conv
	28 x 28	2 x 2 average pool, stride 2
dense block (2)	28 x 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
transition layer (2)	28 x 28	1 x 1 conv
	14 x 14	2 x 2 average pool, stride 2
dense block (3)	14 x 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
transition layer (3)	14 x 14	1 x 1 conv
	7 x 7	2 x 2 average pool, stride 2
dense block (4)	7 x 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$
classification layer	1 x 1	7 x 7 global average pool
		1000D fully-connected, softmax

Figure A3. Loss and accuracy of GoogLeNet.

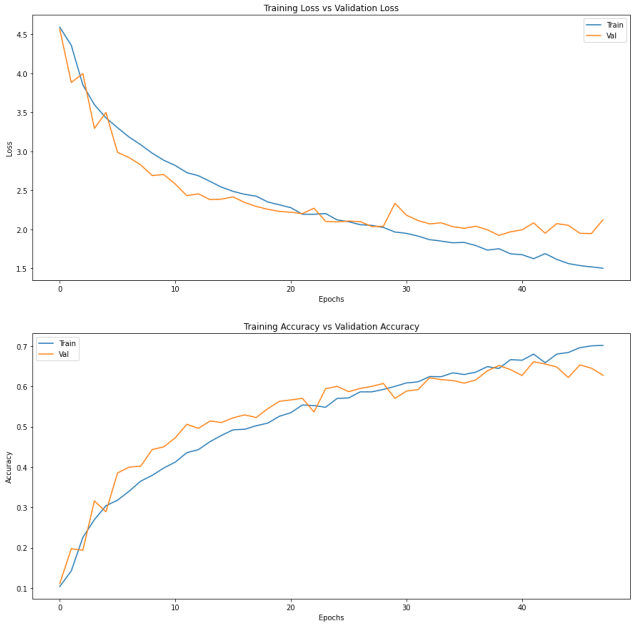


Figure A4. Loss and accuracy of ResNet50.

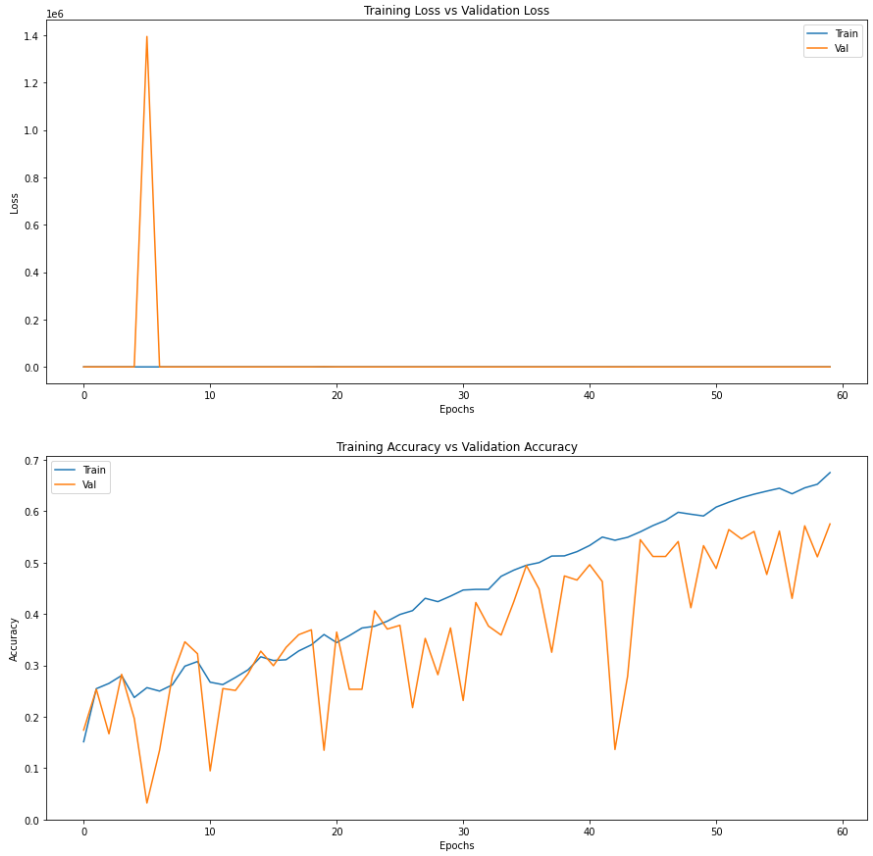


Figure A5. Loss and accuracy of DenseNet121.