Deep learning for classifying/generating images of flowers

Jet Hughes

April 2024

1 Introduction

The report describes methods used to classify and generate images of flowers from the oxford102 flowers dataset [2] using TensorFlow's Keras library. The dataset contains 8K images of 102 types of flowers. The dataset can be loaded as "coarse-grained" with only ten classes (Shown in Figure 1) or "fine-grained" with all 102 classes (Shown in Figure 2). We first trained networks to fit the coarse dataset and later adapted them for the fine dataset. The dataset is imbalanced, as shown in Figures 3 and 4.

2 Task 1: Classification

2.1 Architecture

The input to the network is a 96 x 96 RGB image (3 channels). The only preprocessing is normalising the pixel values to a value between 0 and 1. The overall architecture is adapted from the VGG [4] networks. The image is passed through a series of blocks, each with three sets of 3x3 convolutions, followed by batch normalisation and 2x2 max pooling. Following the convolutions is a dropout layer with a rate of 0.1 and two fully connected dense layers, each with 128 neurons. The second dense layer is connected to the final output layer with 102 neurons (one for each class). L2 weight decay is applied to the first two dense layers with a weight decay beta 0.1. All hidden layers use ReLU activation; the final layers use softmax activation. All Convolutions use "same" padding so the shape is the same between consecutive convolutions. The architecture is shown in Table 1. The architecture for the fine and coarse models is the same.

The main advantage of the VGG-like approach is found in the consecutive 3x3 convolutions. Previous models use larger kernel sizes to capture complex features. Placing 3 3x3 convolutions in sequence means the third convolution can effectively "see" a 7x7 grid without needing a large 7x7 kernel. This allows it to learn more complex features at a comparatively lower computational cost.

Figure 1: A sample of images from the coarse dataset

Figure 2: A sample of images from the fine dataset

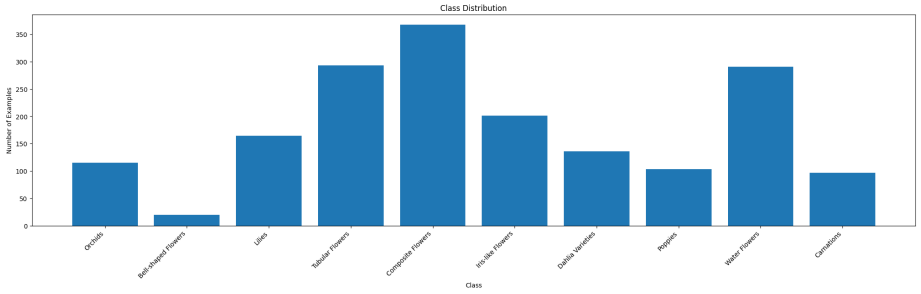

Figure 3: Bar chart showing distribution of the training data among the classes in the coarse dataset

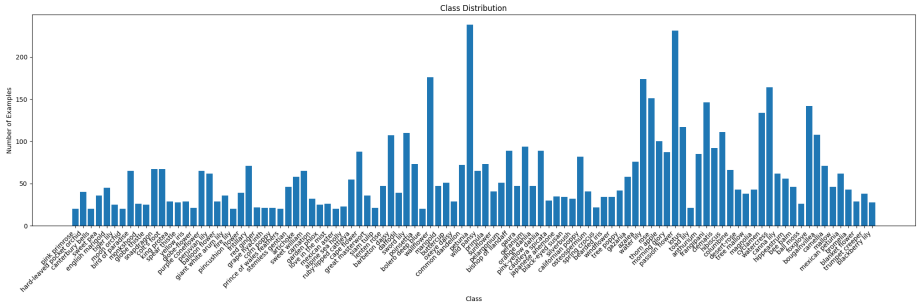

Figure 4: Bar chart showing distribution of the training data among the classes in the fine dataset

Layer Type	**Output Shape**	**Param #**
Rescaling	(None, 96, 96, 3)	0
Conv2D	(None, 96, 96, 32)	896
Conv2D	(None, 96, 96, 32)	9,248
BatchNormalization	(None, 96, 96, 32)	128
MaxPooling2D	(None, 48, 48, 32)	0
Conv2D	(None, 48, 48, 64)	18,496
Conv2D	(None, 48, 48, 64)	36,928
BatchNormalization	(None, 48, 48, 64)	256
MaxPooling2D	(None, 24, 24, 64)	0
Conv2D	(None, 24, 24, 128)	73,856
Conv2D	(None, 24, 24, 128)	147,584
Conv2D	(None, 24, 24, 128)	147,584
BatchNormalization	(None, 24, 24, 128)	512
MaxPooling2D	(None, 12, 12, 128)	0
Conv2D	(None, 12, 12, 256)	295,168
Conv2D	(None, 12, 12, 256)	590,080
Conv2D	(None, 12, 12, 256)	590,080
BatchNormalization	(None, 12, 12, 256)	1,024
MaxPooling2D	(None, 6, 6, 256)	0
Dropout	(None, 6, 6, 256)	0
Flatten	(None, 9216)	0
Dense	(None, 128)	1,179,776
Dense	(None, 128)	16,512
Dense	(None, 102)	13,158
Total params		3,121,286
Trainable params		3,120,326
Non-trainable params		960

Table 1: Classification Network Architecture

2.2 Training

The training uses the "Adam" optimiser with a batch size of 32. The loss function is sparse categorical cross-entropy. The learning rate was initially set to 0.01 and reduced by a factor of 10 after 100 epochs, then again after 50 epochs. The training was stopped after a further 20 epochs. Weights were initialised using the TensorFlow Glorot Uniform initialiser. Biases were initialised to zero. The data was augmented using the TensorFlow Image Data Generator. A random translation, horizontal flips and ZCA whitening with an epsilon value of 1e-6 were applied.

To reduce the effect of imbalanced data, I attempted to initialise the weights of the bias in the output layer to match the relative proportions of the classes. For example, if there is a dataset with three classes with three examples of dog, 1 example of cat and six examples of hamster. The bias of the corresponding

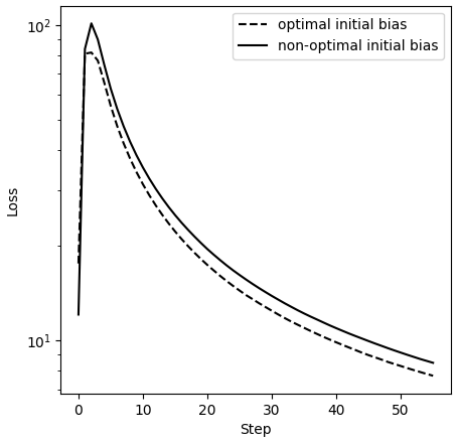

Figure 5: Loss over the first 50 steps of training. Optimal initial bias vs zeroes initial bias

output neurons would be set to 0.3, 0.1, and 0.6, respectively. This has been shown to quicken initial convergence [1]. However, As shown in figure 5, it did not appear significantly better than initialising biases in the output layer to zero, so I initialised the biases of the output layer to zero. The fine and coarse models were trained in the same way.

2.3 Results

The model achieved 81% balanced accuracy (macro average of recalls) on the coarse dataset and 83% balanced accuracy on the fine dataset.

2.4 Coarse Dataset

Results for the coarse dataset are shown in Figure 2. Accuracy is 0.85, and balanced accuracy is 0.81. The worst-performing class is bell-shaped flowers, with an f1 score of 0.33. This is expected as it is significantly under-represented in the dataset. Most other classes perform reasonably well (above 0.7 f1 score). Classes that are well-represented in the dataset perform well as expected. The Iris-like flowers have low precision but high recall, indicating they are consistently detected but are often confused with other flowers.

2.5 Fine Dataset

Since there are many more classes, the complete classification report on the fine dataset is included in Appendix A. The results are summarised in Table 3. Accuracy was 0.83, and balance accuracy was the same. This indicates the model does not significantly under-perform on underrepresented classes.

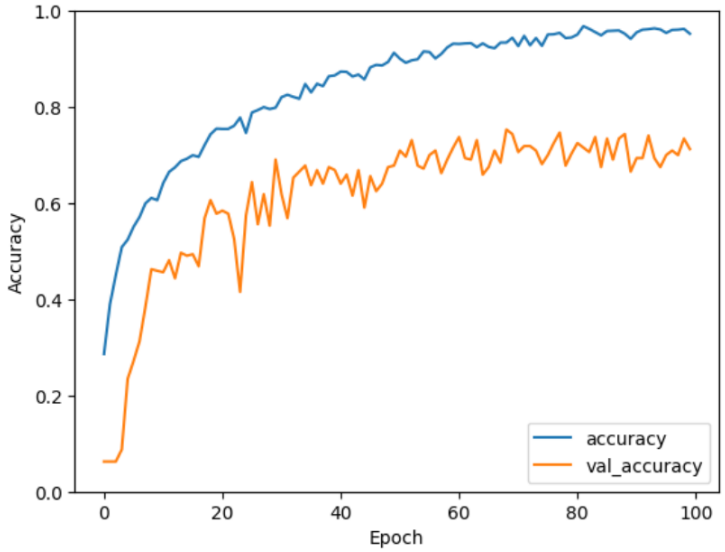

Figure 6: The first 100 epochs of training on the coarse dataset. Learning rate is 0.001

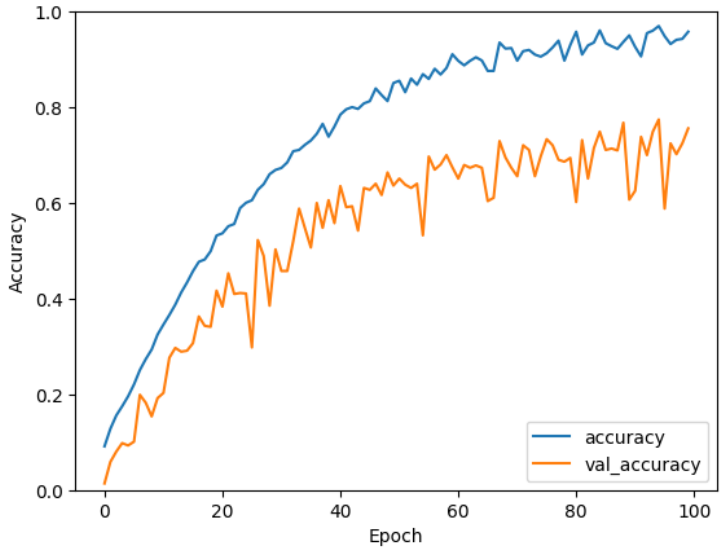

Figure 7: The first 100 epochs of training on the fine dataset. Learning rate is 0.001

Class	Precision	Recall	F1-score	Support
Orchids	0.96	0.87	0.91	30
Bell-shaped Flowers	1.00	0.20	0.33	10
Lilies	0.80	0.66	0.73	50
Tubular Flowers	0.77	0.90	0.83	40
Composite Flowers	0.94	0.97	0.96	70
Iris-like Flowers	0.64	0.90	0.75	40
Dahlia Varieties	0.95	1.00	0.98	20
Poppies	0.94	0.85	0.89	20
Water Flowers	1.00	1.00	1.00	20
Carnations	0.94	0.75	0.83	20
Accuracy				0.85
Macro avg	0.90	0.81	0.82	320
Weighted avg	0.87	0.85	0.85	320

Table 2: Classification Report on the coarse dataset

Class	Precision	Recall	F1-score	Support
Accuracy			0.83	1020
Macro avg	0.85	0.83	0.83	1020
Weighted avg	0.85	0.83	0.83	1020

Table 3: Summary of classification Report on the fine dataset

The model learns very steadily over the first 100 epochs. After decreasing the learning rate, the model jumps in accuracy and stabilises. The training and validation accuracy stay satisfactorily close and don't diverge significantly. This indicates the model is not excessively overfitting.

3 Task 2a: Autoencoder

3.1 Architecture

The input to the network is a 96x96 RGB image. The only preprocessing is normalising the pixel values to a value between 0 and 1. The overall architecture is adapted from U-Net [3]. Similarly to U-Net, the network consists of two parts: a contracting part and an expanding part. The contracting part consists of 3 blocks, each with two sets of 3x3 ReLU convolutions followed by 2x2 max pooling. After each block, the number of channels is doubled. The expanding part also consists of four blocks. However, the max-pooling layers are replaced with "up-convolutions", and the output of each up-convolution is concatenated with the feature map from the corresponding contracting block. A 1x1 convolution with three channels follows the final expanding block to create a colour image output.

The main difference from U-Net is that the filters in the first block are

reduced from 64 to 32 channels, and the number of blocks is reduced from 4 to 3. The resulting network contains $\tilde{4}.5$ million parameters (reduced from $\tilde{3}4$ million). The architecture is shown in Table 4

Layer (type)	**Output Shape**	**Param #**	**Connected to**
InputLayer	(None, 96, 96, 3)	0	-
conv2d	(None, 96, 96, 32)	896	input_layer
conv2d_1	(None, 96, 96, 32)	9,248	conv2d
max_pooling2d	(None, 48, 48, 32)	0	conv2d_1
conv2d_2	(None, 48, 48, 64)	18,496	max_pooling2d
conv2d_3	(None, 48, 48, 64)	36,928	conv2d_2
max_pooling2d_1	(None, 24, 24, 64)	0	conv2d_3
conv2d_4	(None, 24, 24, 128)	73,856	max_pooling2d_1
conv2d_5	(None, 24, 24, 128)	147,584	conv2d_4
max_pooling2d_2	(None, 12, 12, 128)	0	conv2d_5
conv2d_6	(None, 12, 12, 512)	590,336	max_pooling2d_2
conv2d_7	(None, 12, 12, 512)	2,359,808	conv2d_6
conv2d_transpose	(None, 24, 24, 128)	589,952	conv2d_7
concatenate	(None, 24, 24, 256)	0	conv2d_transpose conv2d_5
conv2d_8	(None, 24, 24, 128)	295,040	concatenate
conv2d_9	(None, 24, 24, 128)	147,584	conv2d_8
conv2d_transpose_1	(None, 48, 48, 64)	73,792	conv2d_9
concatenate_1	(None, 48, 48, 192)	0	conv2d_transpose_1 conv2d_3
conv2d_10	(None, 48, 48, 64)	73,792	concatenate_1
conv2d_11	(None, 48, 48, 64)	36,928	conv2d_10
conv2d_transpose_2	(None, 96, 96, 32)	18,464	conv2d_11
concatenate_2	(None, 96, 96, 64)	0	conv2d_transpose_2 conv2d_1
conv2d_12	(None, 96, 96, 32)	18,464	concatenate_2
conv2d_13	(None, 96, 96, 32)	9,248	conv2d_12
conv2d_14	(None, 96, 96, 3)	99	conv2d_13
Total params		**4,500,515**	
Trainable params		**4,500,515**	
Non-trainable params		**0**	

Table 4: U-Net-Like Autoencoder Architecture

3.2 Training

The network was first trained as an autoencoder using the Adam optimiser and mean squared error loss for ten epochs.

3.3 Results

The U-Net-Like model performed exceptionally well as an autoencoder. However, it isn't truly an autoencoder, as the decoding half of the models takes input (skip connections) from the encoding half.

It achieved a mean per-pixel error of 0.01 with a standard deviation of 0.01.

Figure 8: Four examples of images reconstructed using the U-Net-based autoencoder and their original images.

4 Task 2b: Diffusion

The diffusion model has the same architecture as the autoencoder but is trained on a different dataset. It was trained to remove noise from images iteratively. For each image of the flower dataset, many versions of the image were created with increasing levels of noise added. Each image and label pair in the new dataset was an image and the slightly less noisy image that came before it. Training the model using this dataset encourages it to reduce noise in the input image iteratively to end up with a de-noised "flower".

4.1 Dataset

The dataset was created by incrementally adding uniform noise to the previous image. The dataset consists of 10000 images, i.e., 20 noise increment pairs of 500 flowers from the fine dataset. The size of the dataset is limited by available RAM. The noise increment factor was 0.15, i.e., the first noisy image is $clean\_image + 0.15 \times uniform\_noise\_image$. The nth noisy image is given by the formula:

$$noisy\_image_i = noisy\_image_{i-1} + noise\_increment \times uniform\_noise\_image$$

4.2 Training

The model was trained for ten epochs using the Adam optimiser and mean squared error loss. The data was ordered, and the batch size was equal to the number of noise increments (i.e., 20). This means that each batch contains all versions of one image.

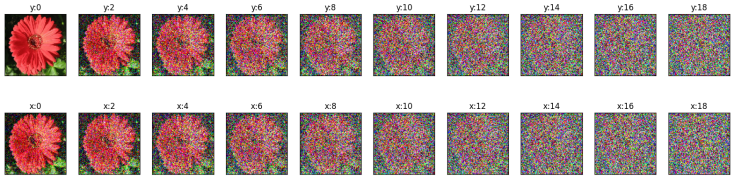

Figure 9: Example of incremental noise processing applied to one image. The top row shows the target(cleaner) image. The bottom row shows the input (noisy) image.

4.3 Results

The generated "flowers" do not look like flowers. Figure 10 shows the best images. These images have a green background with a textured yellow blob. An imaginative individual may convince themselves they see a blurry image of a yellow flower in a green field.

The images in Figure 10 show the model gradually morphing noise into an image. After the 19th iteration, the images become very high contrast and converge to a single colour of maximum brightness, then to nearly pure black or white images. I would expect the number of interactions to generate a good image that matches the number of increments in the training dataset. However, I think the best images occur close to the 14th iteration.

Using a larger dataset will improve the quality of the results. I was limited by available RAM. It may also help to reduce contrast in the input images. To create a more interesting output, it may help to use structured noise as a starting point for the generative model.

References

[1] A. Molas. How to initialize your bias., June 2023.

[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*, pages 722–729, Bhubaneswar, India, Dec. 2008. IEEE.

[3] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015. arXiv:1505.04597 [cs].

[4] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, Apr. 2015. arXiv:1409.1556 [cs].

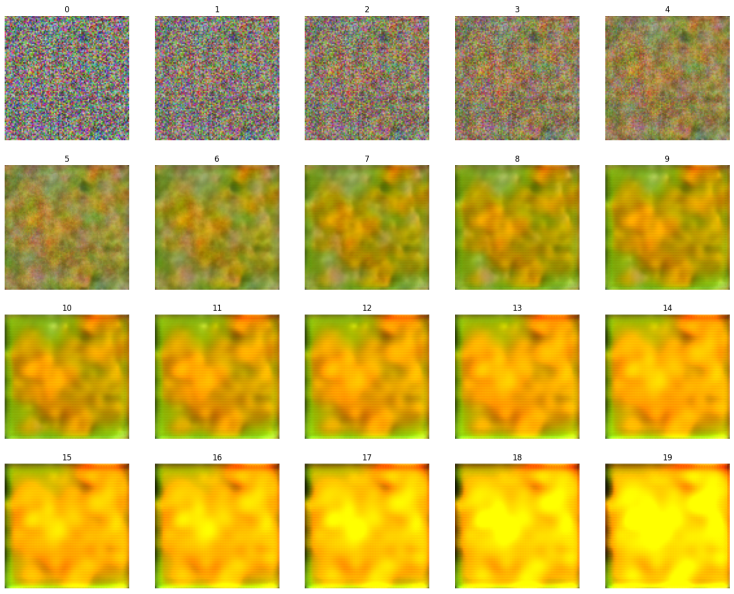

Figure 10: Images of the first 20 iterations of generating an image of a flower from noise

Appendix A Classification Report on Fine Dataset

Table 5: Combined Classification Report

Class	Precision	Recall	F1-score	Support
Pink Primrose	0.67	0.60	0.63	10
Hard-leaved Pocket Orchid	0.91	1.00	0.95	10
Canterbury Bells	0.40	0.20	0.27	10
Sweet Pea	0.62	0.50	0.56	10
English Marigold	0.89	0.80	0.84	10
Tiger Lily	0.90	0.90	0.90	10
Moon Orchid	0.89	0.80	0.84	10
Bird of Paradise	1.00	0.80	0.89	10
Monkshood	0.75	0.90	0.82	10
Globe Thistle	1.00	0.80	0.89	10
Snapdragon	0.64	0.70	0.67	10
Colt's Foot	0.83	1.00	0.91	10
King Protea	1.00	0.80	0.89	10
Spear Thistle	1.00	1.00	1.00	10
Yellow Iris	0.91	1.00	0.95	10
Globe-flower	0.86	0.60	0.71	10

Continued on next page

Table 5 – continued from previous page

Class	Precision	Recall	F1-score	Support
Purple Coneflower	1.00	1.00	1.00	10
Peruvian Lily	0.78	0.70	0.74	10
Balloon Flower	0.89	0.80	0.84	10
Giant White Arum Lily	0.67	0.80	0.73	10
Fire Lily	1.00	1.00	1.00	10
Pincushion Flower	0.89	0.80	0.84	10
Fritillary	0.78	0.70	0.74	10
Red Ginger	1.00	0.90	0.95	10
Grape Hyacinth	1.00	0.90	0.95	10
Corn Poppy	1.00	0.70	0.82	10
Prince of Wales Feathers	0.83	1.00	0.91	10
Stemless Gentian	1.00	0.70	0.82	10
Artichoke	0.71	1.00	0.83	10
Sweet William	0.75	0.90	0.82	10
Carnation	0.86	0.60	0.71	10
Garden Phlox	0.60	0.30	0.40	10
Love in the Mist	0.89	0.80	0.84	10
Mexican Aster	1.00	0.80	0.89	10
Alpine Sea Holly	0.80	0.80	0.80	10
Ruby-lipped Cattleya	0.89	0.80	0.84	10
Cape Flower	0.83	1.00	0.91	10
Great Masterwort	0.70	0.70	0.70	10
Siam Tulip	0.75	0.60	0.67	10
Lenten Rose	0.77	1.00	0.87	10
Barbeton Daisy	0.89	0.80	0.84	10
Daffodil	0.78	0.70	0.74	10
Sword Lily	0.42	0.80	0.55	10
Poinsettia	0.82	0.90	0.86	10
Bolero Deep Blue	0.86	0.60	0.71	10
Wallflower	0.90	0.90	0.90	10
Marigold	0.83	1.00	0.91	10
Buttercup	0.80	0.80	0.80	10
Oxeye Daisy	1.00	0.90	0.95	10
Common Dandelion	0.89	0.80	0.84	10
Petunia	0.53	0.90	0.67	10
Wild Pansy	0.90	0.90	0.90	10
Primula	1.00	1.00	1.00	10
Sunflower	1.00	0.90	0.95	10
Pelargonium	0.90	0.90	0.90	10
Bishop of Llandaff	1.00	0.90	0.95	10
Gaura	1.00	0.70	0.82	10
Geranium	0.91	1.00	0.95	10
Orange Dahlia	1.00	1.00	1.00	10

Continued on next page

Table 5 – continued from previous page

Class	Precision	Recall	F1-score	Support
Pink-yellow Dahlia	1.00	1.00	1.00	10
Cautleya Spicata	1.00	1.00	1.00	10
Japanese Anemone	0.90	0.90	0.90	10
Black-eyed Susan	1.00	1.00	1.00	10
Silverbush	0.91	1.00	0.95	10
Californian Poppy	1.00	0.90	0.95	10
Osteospermum	1.00	1.00	1.00	10
Spring Crocus	1.00	0.80	0.89	10
Bearded Iris	0.70	0.70	0.70	10
Windflower	0.91	1.00	0.95	10
Tree Poppy	0.83	1.00	0.91	10
Gazania	1.00	1.00	1.00	10
Azalea	0.73	0.80	0.76	10
Water Lily	1.00	1.00	1.00	10
Rose	0.75	0.90	0.82	10
Thorn Apple	0.71	1.00	0.83	10
Morning Glory	1.00	0.90	0.95	10
Passion Flower	0.69	0.90	0.78	10
Lotus	0.89	0.80	0.84	10
Toad Lily	0.88	0.70	0.78	10
Anthurium	0.67	1.00	0.80	10
Frangipani	0.91	1.00	0.95	10
Clematis	0.69	0.90	0.78	10
Hibiscus	0.82	0.90	0.86	10
Columbine	0.62	0.50	0.56	10
Desert-rose	1.00	0.80	0.89	10
Tree Mallow	0.78	0.70	0.74	10
Magnolia	0.90	0.90	0.90	10
Cyclamen	0.73	0.80	0.76	10
Watercress	0.77	1.00	0.87	10
Canna Lily	0.57	0.80	0.67	10
Hippeastrum	0.89	0.80	0.84	10
Bee Balm	0.91	1.00	0.95	10
Ball Moss	0.67	0.60	0.63	10
Foxglove	0.60	0.90	0.72	10
Bougainvillea	0.55	0.60	0.57	10
Camellia	0.89	0.80	0.84	10
Mallow	0.75	0.30	0.43	10
Mexican Petunia	0.89	0.80	0.84	10
Bromelia	0.89	0.80	0.84	10
Blanket Flower	1.00	1.00	1.00	10
Trumpet Creeper	0.78	0.70	0.74	10
Blackberry Lily	1.00	1.00	1.00	10

Continued on next page

Table 5 – continued from previous page

Class	Precision	Recall	F1-score	Support
Accuracy			0.83	1020
Macro avg	0.85	0.83	0.83	1020
Weighted avg	0.85	0.83	0.83	1020