University of Oxford

Fourth year project

A Flower Classification Application

Author:

James Routley

Trinity College

Supervisors:

Professor Andrew Zisserman

Dr Yuning Chai

May 2015

Abstract

This project makes use of computer vision techniques to build a mobile application which performs image classification of flowers. The application can classify 102 flower species common to the United Kingdom, with a practical accuracy of 96.2%.

The classification pipeline consists of feature extraction and followed by classification. An image of a flower is passed to a convolutional neural network which outputs a vector that describes the features of the flower. The dot products of the feature vector and 102 one-vs-the-rest support vector machine classifiers are found, giving 102 prediction values, which give the likelihood of the flower being each of those 102 flower species. These prediction values are then sorted, producing a ranked list of the flower's most likely species.

This classification pipeline is run on a server and is accessible from an application which is running on a user's Android mobile device. The application allows the user to choose whether to take a photo of a flower or upload a previously captured flower image. This image is automatically uploaded to the classification pipeline, classified and a list of the six highest predicted species are returned to the device. Images of these six flowers are displayed in a scrollable list on the device. If one of the images is clicked, a new screen comes up, displaying more information about that flower.

Contents

Chapter 1

Introduction

1.1 Motivation

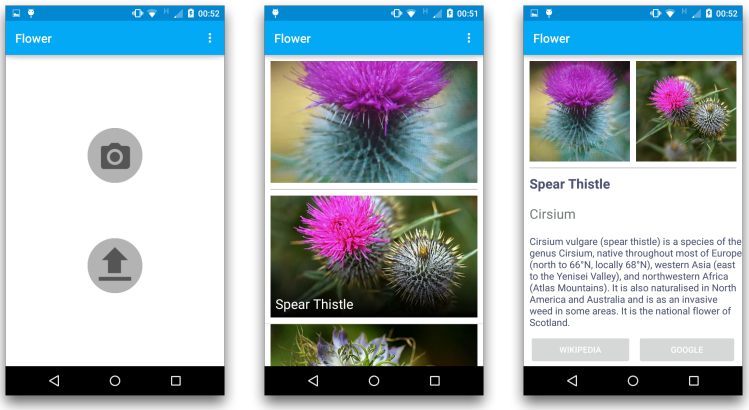
Classifying flower species is difficult for humans to do as it requires access to in-depth specialist knowledge; some flower species look similar, whilst other flowers look different despite being the same species, as shown in Figure 1.1. Intuitively, the flower species classification accuracy of a non-expert human would be low, and to learn to differentiate between flower species would require a large time commitment.

Figure 1.1: Top row: different species which look similar (l-r: artichoke, globe thistle, lotus, water lily). Bottom row: difference within a species; all are examples of the sword lily.

Currently, the standard method for classifying flowers is manual searching through lengthy field guides. Although tools like decision trees and digitalisation make the process easier, classification is still difficult for amateurs who may not possess the vocabulary to accurately describe the flower

in question.

This report presents an application which greatly simplifies this process, shown in Figure 1.2. From a single photo of a flower, computer vision and machine learning algorithms are used to classify that flower's species. The process is quick (in the order of seconds and has a practical accuracy of 96.2%. It requires the user to have no prior knowledge and is robust to variations in photo quality, light levels and background content. No comparable applications are currently available.



(a) (b) (c)

Figure 1.2: Screenshots of the screens seen by user. (a): The user chooses whether to take or upload a photo. (b): The photo is automatically uploaded, classified and the results of the classification are displayed. (c): If one of the images in (b) is clicked, a detailed view of that flower is shown. The flower's common name, genus, a snippet from information from Wikipedia and buttons linking to the flower's Wikipedia and Google Search pages are displayed.

For the user, the application is simple. A photo is taken, it is automatically uploaded and classified, and the six top classification predictions are presented to the user, who makes the final decision.

The classification itself is an automated process. After the photo is uploaded to a server, it is passed to a convolutional neural network which outputs a vector describing the features, such as shape and colour, of the photo. This feature vector is compared to a previously trained support vector machine model of 102 common flower species. A ranked list of the six most similar flowers from the 102 is returned to the application and presented in a clear and intuitive manner.

This application implements fine-grained classification, that is classification between categories which are visually and semantically similar. Although flowers are the chosen classification subject

of this project, the same techniques could successfully be applied to other fine-grained classification tasks.

1.2 Road map

This report consists of six chapters. In Chapter 2, the literature review examines previous work which is relevant, including work done by Oxford University on the flower classification problem and applications of a similar nature. This chapter also reviews the convolutional neural network algorithm which is used to extract features from the flower image.

Chapter 3 considers the flower classification. The two-step system of convolutional neural network based feature extraction and support vector machine classification is laid out. The experiments carried out to fine tune the classification system and the results obtained are also covered.

In Chapter 4, the client (application) and server architecture are examined. The process by which the user takes and uploads the photo, how the classification process is implemented and the communication between client and server are detailed.

Chapter 5 looks at application design and looks at the user interface and user experience. While chapter four covered the mechanics of how the application works, this chapter considers its look and feel.

Finally, Chapter 6 presents a summary of this project and discusses further work which can be carried out on this topic.

Chapter 2

Literature review

This chapter considers the previous work relevant to the project. It gives a description of the data used, looks at the work done at Oxford University on this particular problem, the history of the algorithms used, and other applications which solve similar problems.

2.1 Description of data

Figure 2.1: Example images from the 17 and 102 flower datasets

This project makes use of the Oxford 17 category flower dataset[1] and the Oxford 102 category flower dataset[2]. Example images from the datasets are shown in Figure 2.1. The 17 flower dataset

[1] http://www.robots.ox.ac.uk/ vgg/data/flowers/17

[2] http://www.robots.ox.ac.uk/ vgg/data/flowers/102

is split into 17 flower species, each containing 80 images. The 102 flower dataset is split into 102 flower species, each containing between 40 and 258 images.

The flower species in both sets are common to the United Kingdom, and the photos exhibit variations in pose and light levels. Both include species which exhibit great variation within the species and species which look similar to others in the dataset, shown in Figure 1.1.

The datasets are split into training, validation and test sets, and the categories themselves are numbered 1 to 17 and 1 to 102 respectively. A data file which relates these these category numbers to an appropriate genus, species and common name is used to present meaningful information to the user.

2.2 Flower Classification

This project builds on work done by the Oxford University Visual Geometry Group on the flower classification problem. Nilsback et al. [8] considered how combinations of features (local shape/texture, shape of the boundary, overall spatial distribution of petals, and colour) affect classification accuracy. After feature extraction has taken place, a support vector machine (SVM) classifier is used. When tested against the Oxford 102 flower dataset, a classification accuracy of 72.8% was obtained.

Chai [1] uses a flower classification method which involves segmenting each flower image into foreground (the flower itself) and background, before using Fisher vectors to extract flower features. An example of the image segmentation is shown in Figure 2.2. An SVM classifier was also used to achieve classification accuracy of 81.4% when tested on the Oxford 102 flower dataset.

Figure 2.2: Image segmentation from [1]

2.3 Convolutional Neural Networks

This project takes a different approach to Chai and Nilsback, making use of a convolutional neural network (CNN) for feature extraction. CNNs are a type of biologically-inspired neural network, based on the multilayer perceptron [6].

A CNN takes an input vector and transforms it to an output vector by passing it though several hidden layers each made up of a set of neurons. It can be thought of as a function f which transforms an input $\boldsymbol{x}$ into an output $\boldsymbol{y}$. The function f is in fact made up of a sequence of simpler functions $f_1, f_2, ..., f_N$ [9]. Each function $f(\boldsymbol{x}, \boldsymbol{w})$ has a set of parameters $\boldsymbol{w}$ which are determined during training.

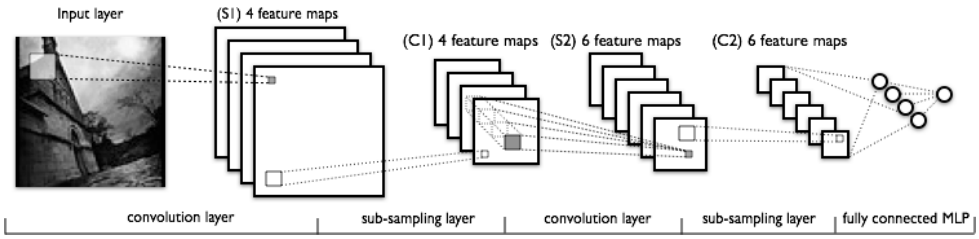


Figure 2.3: Diagram showing the layers in a convolutional neural network

2.4 Automatic visual classification applications

Automatic flower classification applications do not exist at the time of writing, but there are several applications which use computer vision techniques to classify subjects such as tree species and dog breeds.

The iPhone application Leafsnap [5] uses computer vision to automatically identify tree species using photographs of leaves. The leaf shape is used as the principle feature, and shape is obtained by using colour-based segmentation to remove the background from the image of the leaf before taking multiscale curvature measures. For optimal use, the leaf must be placed on a white sheet of paper before the image is taken. The flowers are classified from this data by using nearest-neighbour classification. The application currently classifies the 185 tree species in the Northeastern United States.

Figure 2.4: Screenshots of the image capture and results screens from the Leafsnap application [5]

The application makes use of a client-server architecture, where the classification takes place on Leafsnap's servers. The classification time for a single photo is 5.4 seconds, not including upload time. It also implements a 'field guide' containing images and descriptions of the trees it classifies.

The group which produced Leafsnap, Columbia University, The University of Maryland and The Smithsonian Institution have released another automatic image classification application called Dogsnap, which classifies dog breeds.

Dogsnap uses greyscale SIFT descriptors and a colour histogram to extract features from the photograph and uses a one-vs-the-rest SVM for classification [7]. Dogsnap uses relative eye and nose position as the principle features as they are rigid in relation to one another.

2.5 Flower classification applications

Several basic flower classification application have been created which do not use image-based automatic classification. These fall into two categories; digital field guides and human powered classification.

The former provide an extensive database of plant species and allow the user to manually narrow

down their search based on distinguishing plant features[3,4], or by user location[5]. These are usually free and represent a significant improvement on paper field guides; they are more portable, easier to search and as most people carry phones they are usually to hand when needed.

The latter allows the user to take a photograph of the flower in question and upload it to a platform where it is manually classified by a person[6,7]. Applications of this nature differ in who is used to classify the flower. Some crowdsource information from the users of the application. This is usually free but the results are potentially unreliable, the application requires a large user base to work and classification time can take up to days. Others use a team of professionals which is more reliable and potentially faster but requires payment.

[3] https://play.google.com/store/apps/details?id=com.luontoportti

[4] https://itunes.apple.com/us/app/ipflanzen/id416983587

[5] https://play.google.com/store/apps/details?id=org.pottssoftware.agps21

[6] https://play.google.com/store/apps/details?id=org.plantnet

[7] https://play.google.com/store/apps/details?id=cz.thran.flowerchecker

Chapter 3

Classification

This chapter describes the classification pipeline and the experiments done on it. The classification pipeline takes an image as its input and outputs a classification result. The process is summarised in Figure 3.1. This description is mostly theoretical, details of its implementation are covered in Chapter 4. High classification accuracy is crucial for the quality of this application.

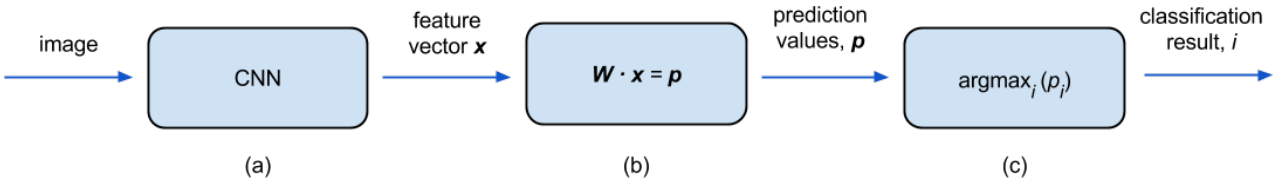


Figure 3.1: (a): Convolutional neural network inspired feature extraction. An image is supplied to the convolutional neural network which generates its feature vector x, which describes the content of the photograph. (b): Support vector machine image classification. The vector is compared to the previously found weight vectors W, producing a vector of prediction values, p. (c) The index i of the maximum value in p is found, which gives the classification result i.

3.1 Convolutional neural network (CNN) based feature extraction

The pipeline makes use of the first six layers of the fast convolutional neural network CNN-F introduced in Chatfield et al. [2]. This algorithm takes a photo of size 244x244 pixels as its input and produces a vector x which describes the features of the photo. The feature vector is 4096 dimen-

sional and is l_2 normalised before being passed to the support vector machine (SVM), as SVMs work best with l_2 normalised vectors.

The CNN is trained on the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) dataset [4]. Details of the training can be found in Chatfield et al. [2]. The feature extraction process itself is treated as a black box in this project.

3.2 Support vector machine (SVM) image classification

3.2.1 How SVMs are used in the classification pipeline

The classification pipeline uses the open source support vector machine (SVM) library LibLinear [3] for quick, large scale classification. One-vs-the-rest classification is used. N SVM models are trained for N flower categories, producing $\boldsymbol{w_1}, \boldsymbol{w_2}, ..., \boldsymbol{w_N}$. These weight vectors are stacked into a weight matrix $\boldsymbol{W}$:

$$\boldsymbol{W} = \begin{pmatrix} \boldsymbol{w_1'} \\ \boldsymbol{w_2'} \\ \vdots \\ \boldsymbol{w_N'} \end{pmatrix}$$

During classification, the scalar product of the unseen flower's feature vector, $\boldsymbol{x}$ and each of the weight vectors is calculated, to produce a vector of prediction values, $p_1, p_2 ... p_n$:

$$\boldsymbol{W} \cdot \boldsymbol{x} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_N \end{pmatrix}$$

The value of p_i indicates how similar the unseen flower's features were to those of category i. The higher the product, the more similar they are. A ranked list of categorisation possibilities is calculated by ordering the prediction values by size.

3.2.2 Training and testing SVMs

Before use, the SVM models $\boldsymbol{w_1}, \boldsymbol{w_2}, ..., \boldsymbol{w_N}$ which make up $\boldsymbol{W}$ must be trained. They are trained using a set of images which are known to be of certain categories. The models are then tested, using a set of images also known to be of certain categories, to check they are sufficiently accurate.

Training

The SVM models are trained using LibLinear's training function. To train a model to recognise a flower category i, all photos in the training and validation sets are passed to the training function, with the photos of category i labeled as in the category, and all other flowers labeled as not in the category. The flowers are labeled as in or not in the category by passing a 1 or -1 respectively to the training function, alongside that flower's feature vector.

The training function produces a weight vector of dimension 4096, which describes the characteristics of the flowers in that category.

The C parameter of the SVM must be adjusted during training. The C parameter allows for a solution with a larger margin, in return for the violation of some constraints. This can improve classification accuracy. To find the optimal C parameter, SVM models are trained using the training set images before being tested against the validation set images. The change in classification accuracy is recorded as the C parameter is changed. Figure 3.2 shows how the accuracy changes as the C parameter is changed. Based on this graph, a C parameter of 10^3 was chosen for the the 17 flower dataset with no test or training set augmentation (explained in Section 3.4).

Testing

Testing allows the quality of each SVM model to be checked. Testing uses the test subset of the flower datasets. Each flower in the test set is run against each of the SVM models generated during training. The dot product of the flower's feature vector and each of the model's weight vectors is found. The greater the result of the dot product, the greater the correlation between the tested flower and that category. Thus the category which produces the greatest dot product result is predicted

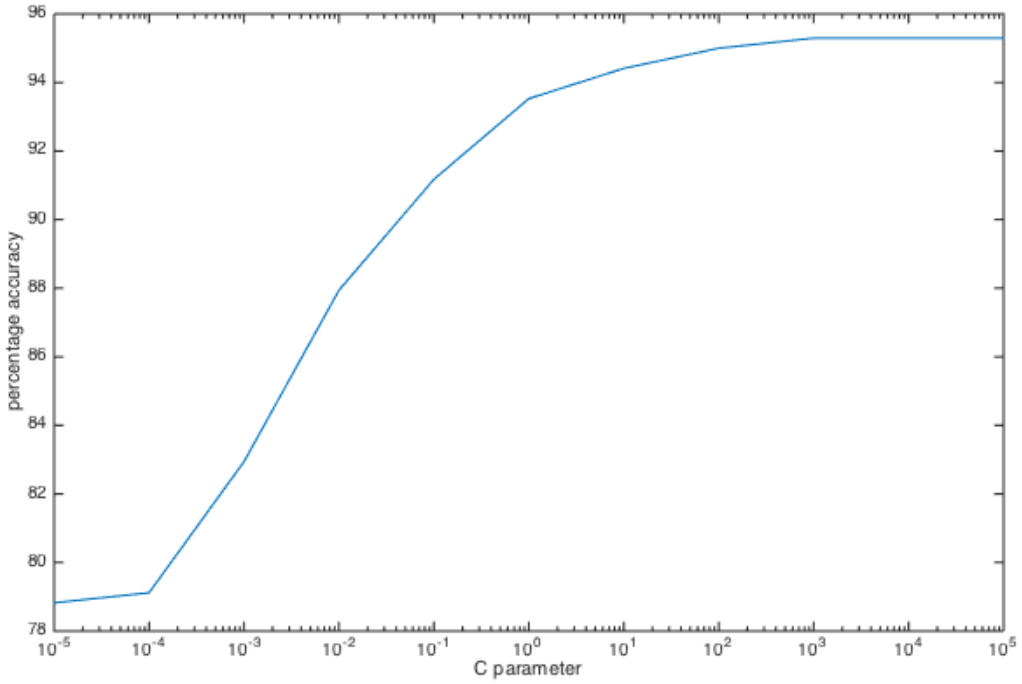


Figure 3.2: Graph showing how accuracy changes as the C parameter is changed for the 17 flower dataset with no test or training set augmentation. Image augmentation is explained in Section 3.4

to be the category of the flower.

By comparing the actual categories of the flowers in the test set to the predicted categories, an overall percentage accuracy can be calculated.

3.3 Experiments

This section describes the metrics used, experiments run and results obtained whilst testing the quality of the classification pipeline.

Accuracy

To calculate an overall accuracy, the accuracy of each model must first be calculated. The overall accuracy is then defined as the mean of the model accuracies:

$$\text{Overall accuracy} = \frac{\text{Sum of model accuracies}}{\text{Number of flower categories}}$$

Model accuracy is defined as:

$$\text{Model accuracy} = \frac{\text{Number of correctly classified flowers}}{\text{Number of flowers classified}}$$

Rank accuracy

Rank accuracy measures the accuracy when considering the top K predicted categories, as demonstrated in Figure 3.3.

$$\text{Rank accuracy} = \frac{\text{Sum of model accuracies when considering the top } K\text{results}}{\text{Number of flower categories}}$$

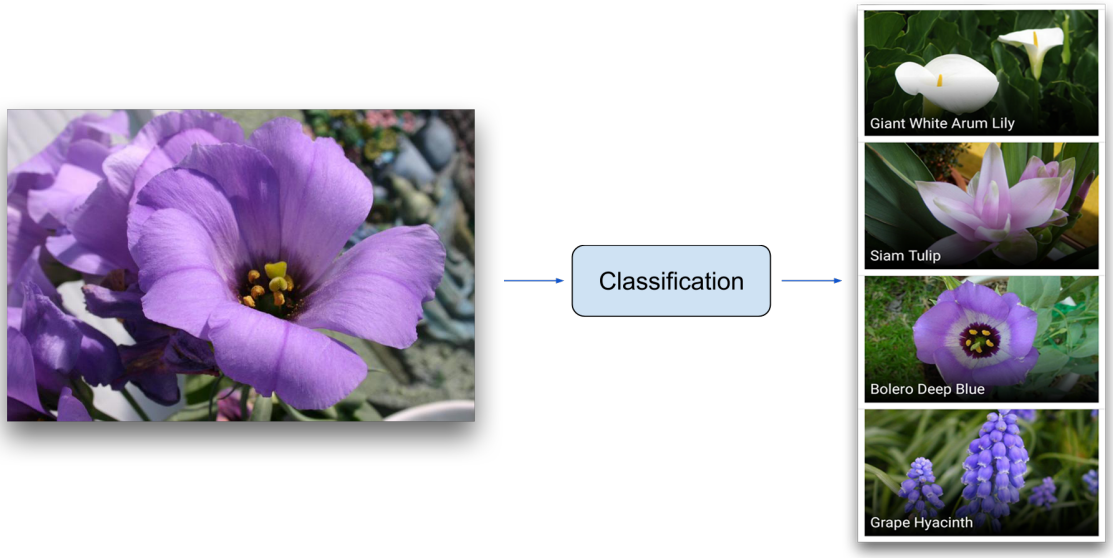


Figure 3.3: Diagram demonstrating rank accuracy. Left: A bolero deep blue flower is being classified. Right: The ranked classification predictions. The accuracy is 0% when considering the first rank ($K = 1$), 0% when considering the first two ranks ($K = 2$), and 100% when considering $K = 3$ and $K = 4$.

3.3.1 Results

Table 3.1 shows the percentage accuracy for the 17 and 102 flower datasets when using training set augmentation. Table 3.2 shows the percentage rank accuracy for the 17 and 102 flower

	17 flower dataset	**102 flower dataset**
Baseline	89.3 $\pm$ 0.96%	85.7 $\pm$ 0.92%
Mirroring	89.1 $\pm$ 0.93%	85.2 $\pm$ 0.30%
Sampling	89.1 $\pm$ 1.06%	84.3 $\pm$ 0.13%
Mirroring and Sampling	89.7 $\pm$ 1.21 %	84.7 $\pm$ 0.43%

Table 3.1: Mean classification accuracy and standard deviation for the Oxford 17 and 102 flower datasets using training set augmentation (Section 3.4.1), and not using test set augmentation (Section 3.4.2)

Number of ranks considered	**17 flower dataset**	**102 flower dataset**
1	89.7%	84.7%
4	98.6%	94.7%
6	99.1%	96.2%
8	99.6%	97.1%
10	99.8%	97.6%

Table 3.2: Classification accuracies obtained when considering different numbers of ranks for the 17 and 102 flower datasets, using training set augmentation

datasets when using training set augmentation. To improve the reliability of the accuracy results, each experiment is run five times, using a randomly generated training set of 20 images from each category. The test set is made up of all the remaining images.

3.3.2 Analysis of results

Accuracy

The highest accuracies obtained for the 17 flower and 102 flower datasets are 89.7% and 85.7% respectively. A lower accuracy for the 102 flower dataset is to be expected as there are more classification possibilities. The accuracy for the 102 flower set is 4 percentage points higher than those obtained in Chai [1] and 13 percentage points higher than those obtained in Nilsback et Al. [8]. The standard deviations of the results are low, indicating that the accuracy figures are reliable. These accuracies are obtained using models trained on a subset of the total images available. The final models used for the application are trained on all the images, and should have higher accuracy rates.

Mirroring and sampling increase the size of the training set and can improve classification accuracy, as seen in [2]. However, this was not seen in these results.

Rank accuracy

The rank accuracy graph shows that classification accuracy quickly increases as more ranks are considered (Table 3.2). While the accuracy is a useful metric for understanding the quality of the models, rank accuracy gives a better insight into the accuracy experienced by the user as they are presented with the top K classification results by the application. The value of K is a compromise between accuracy and complexity; the more results presented to the user, the greater the likelihood of the correct result being in the list but at some point the extra information becomes confusing.

A value of $K = 6$ was settled on as it gives a high accuracy of 96.2% for the 102 flower dataset and six is a very manageable number of options for a user to be presented with. Figure 3.5 shows that considering further ranks yields a greatly diminished increase in accuracy. Offering the user a smaller number of options has the added benefit of reducing the amount of data needed by the application per classification, speeding up the classification.

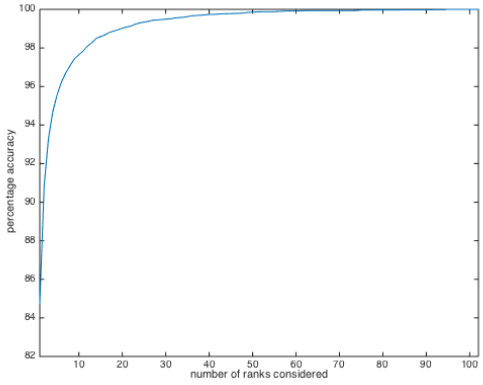


Figure 3.4: Rank accuracy for 102 flower dataset, using training set augmentation

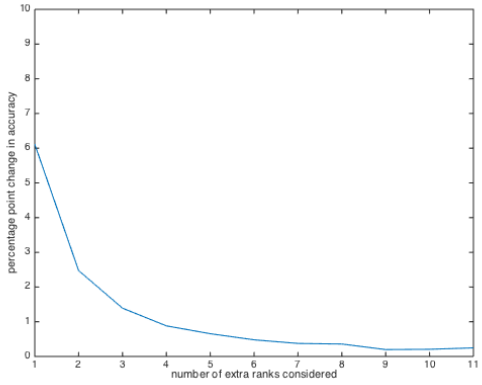


Figure 3.5: Percentage point increase in accuracy when considering more ranks

Test set augmentation

	17 flower dataset	**102 flower dataset**
Baseline	89.1 $\pm$ 0.12%	85.7 $\pm$ 0.92%
Mirroring	88.9 $\pm$ 0.12%	85.7 $\pm$ 0.10%
Sampling	90.1 $\pm$ 1.23%	86.7 $\pm$ 0.40%
Mirroring and Sampling	89.7 $\pm$ 0.66 %	85.3 $\pm$ 0.44%

Table 3.3: Mean classification accuracy and standard deviation for the Oxford 17 and 102 flower datasets using test set augmentation (Section 3.4.2) , and not using training set augmentation (Section 3.4.1)

Test set augmentation, explained in Section 3.4.2, gave the results shown in Table 3.3. Although a slight improvement is seen, test augmentation requires the CNN based feature extraction to be run multiple times. As the feature extraction is the most time costly part of classification, test image augmentation increases classification time by up to 3.6 times, as shown in Table 3.4. For this reason, test image augmentation is not implemented in the classification pipeline.

	Classification time / seconds
Baseline	0.97
Mirroring	1.03
Sampling	2.98
Mirroring and Sampling	3.64

Table 3.4: Classification time when using test image augmentation. These results were obtained by timing the Matlab classification script from within Matlab. Actual classification times will be longer due to additional server computation, image upload and classification information download time.

Confusion matrices

Confusion matrices give a more detailed view of the classification. It presents the classification results as a series of rows. Each row represents all of the images in the test dataset for that category. The numbers in the columns represent the proportion of those images which were classified as that category. For example, for the 17 flower dataset confusion matrix, shown in Figure 3.8, 45% of the images of category 1 in the training set were correctly classified. 3% were incorrectly classified as category 6, 15% as 9 and 38% for 14.

The 17 flower confusion matrix (Figure 3.8) shows that most of the individual classification models perform well, with all but four showing accuracies greater than 70%. However, it can be seen that categories 1, 8 and 15 perform much worse than expected, with accuracies between 40% and 45%. All three of these categories have significant numbers of images (between 38% and 53%) misclassified to category 14, and some images from category 1 are misclassified to category 8.

One can see that by looking at images of the flowers in question, shown in Figure 3.6, that they all look similar, with yellow petaled flowers on mostly green backgrounds. This similarity explains the confusion, which is mostly corrected for when considering rank accuracy.

Category 6, the sword lily shows large colour variation within the species, shown in Figure 3.7. However, the high accuracy of 85% indicates that the these problems are effectively handled by the classification pipeline.

Figure 3.6: Example images from categories 1, 8, 14 and 15. The flowers are similar and there is misclassification between classes.

Figure 3.7: Example images from category 6. Despite being of the same species, the flowers in the dataset differ visually

3.4 Training and test set augmentation

This section covers the techniques used to augment the training and test datasets. This is done as in some cases it has been shown to increase accuracy [2].

3.4.1 Training set augmentation

Taking mirrors and samples of the images in the training set (Figures 3.10 and 3.11) allows the training set to be expanded. The objective of this is to get the most out of the training set images. Both techniques generate new images from which new feature vectors can be calculated. These can be used to improve the quality of the SVM model.

Only the vertical mirror can be used, as shown in Figure 3.10, as flowers only exhibit vertical symmetry.

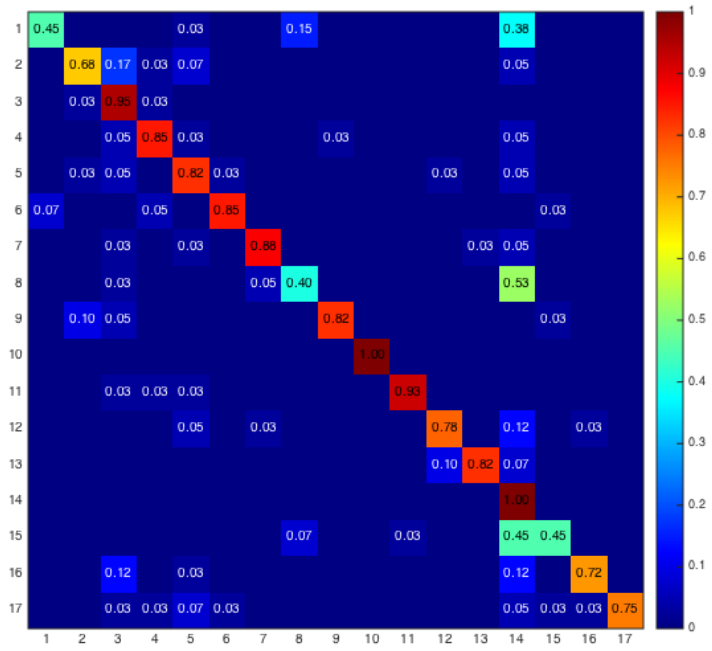


Figure 3.8: Confusion matrix for the 17 flower dataset, obtained using mirroring and sampling

3.4.2 Test set augmentation

For the same reasons as laid out above, mirroring and sampling the test image can improve classification accuracy.

For each image, several image vectors $\boldsymbol{x_i}$ are now being produced. Each one of these is run against the weight matrix to produce several vectors of prediction values $\boldsymbol{p_i}$. To classify the flower, these vectors of prediction values must be combined. This is done by finding the mean of each corresponding value.

$$\boldsymbol{x} = \frac{1}{n}\sum_{n=0}^{n} \boldsymbol{x}_i$$

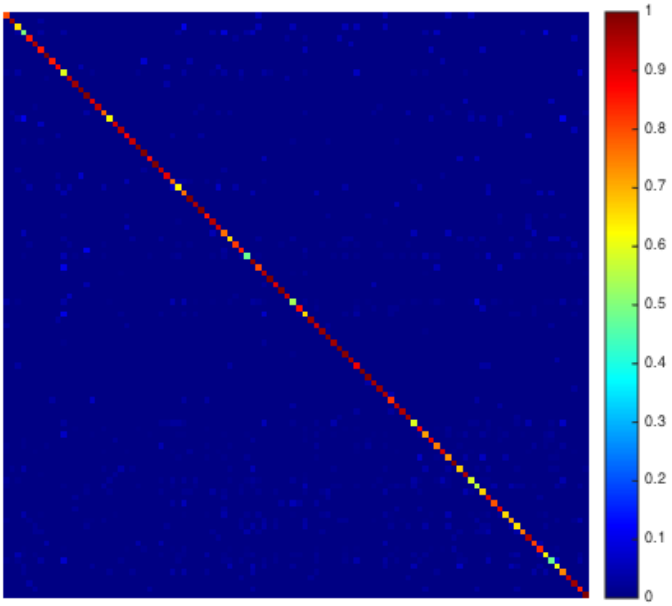


Figure 3.9: Confusion matrix for the 102 flower dataset, obtained using mirroring and sampling

Figure 3.10: An example flower shown with its vertical mirror

Figure 3.11: An example flower shown with the images sampled from it

Chapter 4

Client and server architecture

This chapter describes the implementation of the client and the server. It is concerned only with technical details, the look and feel of the application is discussed in the next chapter. The client is an application running on a user's Android phone. The application allows the user to send photo to the server over the internet. The server is a constantly running computer which runs the classification algorithms described in Section 3. Upon receiving the photo, the server performs the classification and then returns the results to the application, where they are displayed. The server client architecture is shown in Figure 4.1.

Figure 4.1: Diagram showing the information flow between client and server

4.1 Client architecture

The client is an Android application made up of three activities[1]. Each activity is an object which allows the user to perform a specific task, and has a screen with which the user can interact with it. The functions of the activities are shown in Figure 4.2. Screenshots of the activities are shown

[1] http://developer.android.com/reference/android/app/Activity.html

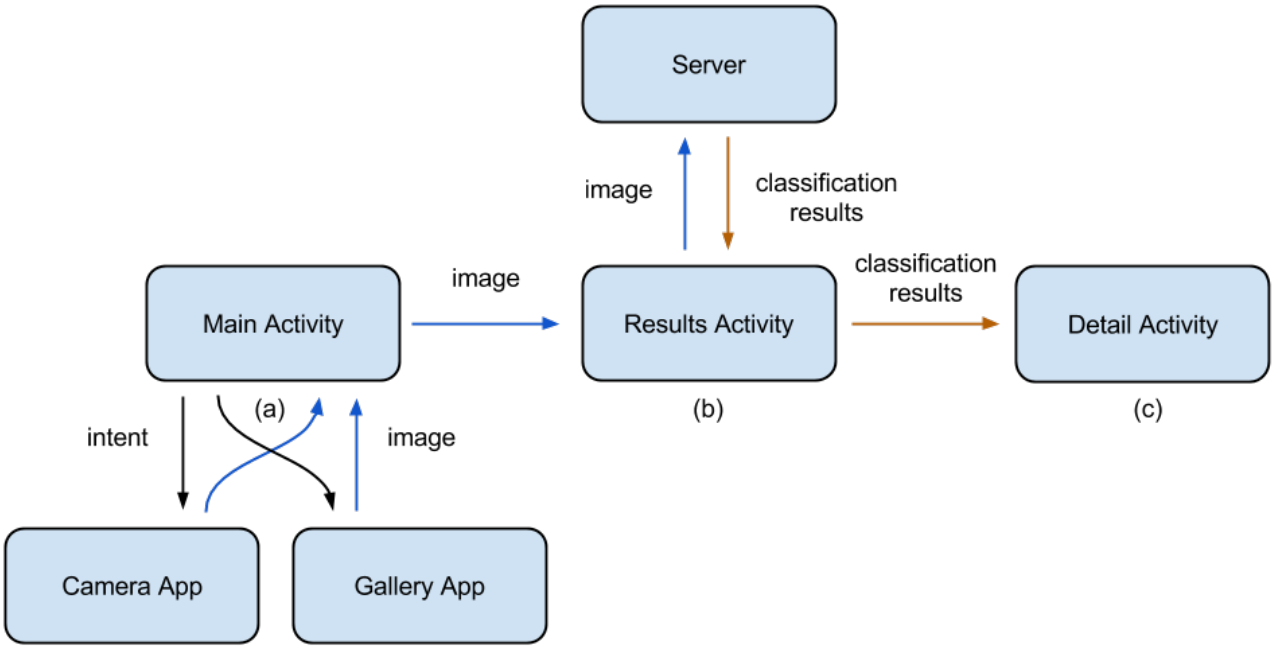


Figure 4.2: Diagram showing the information flow between different sections of the client

in Figures 1.2 and 5.1.

4.1.1 Taking the photo

After the application is launched, the Main Activity of the application (Figure 4.2 (a)) presents the user with two buttons, which allow the user to take a photo, or upload a previously taken photo from a gallery. Both of these buttons trigger intent calls. Intents[2] are objects native to the Android system, which allow an application to request an action from another application component. The intent call starts the built-in camera or gallery application, and allows the user to take or choose a photo without having to write that code into the application.

Once a photo is taken or chosen, the second activity of the application (Figure 4.2 (b)) is called, and the filename of the photo is passed to it.

[2] http://developer.android.com/reference/android/content/Intent.html

4.1.2 Uploading the photo

When the Results Activity (Figure 4.2 (b)) is called, a HTTP connection is opened, connecting to the URL of the server. A HTTP POST request is started, and the photo is uploaded to the server. The filename is appended to the POST request header as the value in a key-value pair. The key is predetermined and is used by the server to extract the image from the body of the request.

The object which contains the code to upload the image extends a native Android object called an AsyncTask[3]. The AsyncTask starts a new thread, and runs the upload code on it. This is done because the photo upload can take several seconds. If this were to take place on the user interface thread, user interface features such as scrolling and use of the navigation buttons could not take place until the uploading had been completed. This makes the application seem like it has frozen and is bad user experience design.

4.1.3 Receiving results

Once the photo has been uploaded to the server, and the classification has been carried out, the server responds with a JSON array containing the classification results and information about those flowers.

The array consists of a number of flower objects, each of which contains the flower name, species, the URL of an example image, a snippet of information about the flower from Wikipedia and a links to that flower's Wikipedia page and google search.

The results are placed in a list, shown in Figure 5.1 (d, e), and the images are downloaded from their URLs using an image downloading and cacheing library named Picasso[4]. Picasso allows for the downloading, display and cacheing of an image with a single line of code. If the user scrolls down the list and an image scrolls off the screen, it is destroyed in order to save memory space. If the user scrolls up again, the cacheing implemented by Picasso allows that same image to be loaded from the cache, rather than be downloaded again. This improves performance as loading from memory is essentially instantaneous while downloading the image can take a few seconds.

[3] http://developer.android.com/reference/android/os/AsyncTask.html

[4] http://square.github.io/picasso

If an image in the list is clicked, a detailed view of that flower is shown (Figure 5.1 (f, g, h)). This shows the user extra information about the flower, and presents them with buttons which allow them to Google search the flower or go to the flower's Wikipedia page.

4.2 Server architecture

The server side consists of two servers, shown in Figure 4.3. The first (front end) receives the image from the client, saves it to a folder and passes the filename to the second (back end) server. The second server calls a Matlab instance, and runs the Matlab classification script on the image. It constructs a JSON array of results and passes it to the first server which returns the results to the client.

This two-server architecture improves security as file input and output is kept separate from the classification logic.

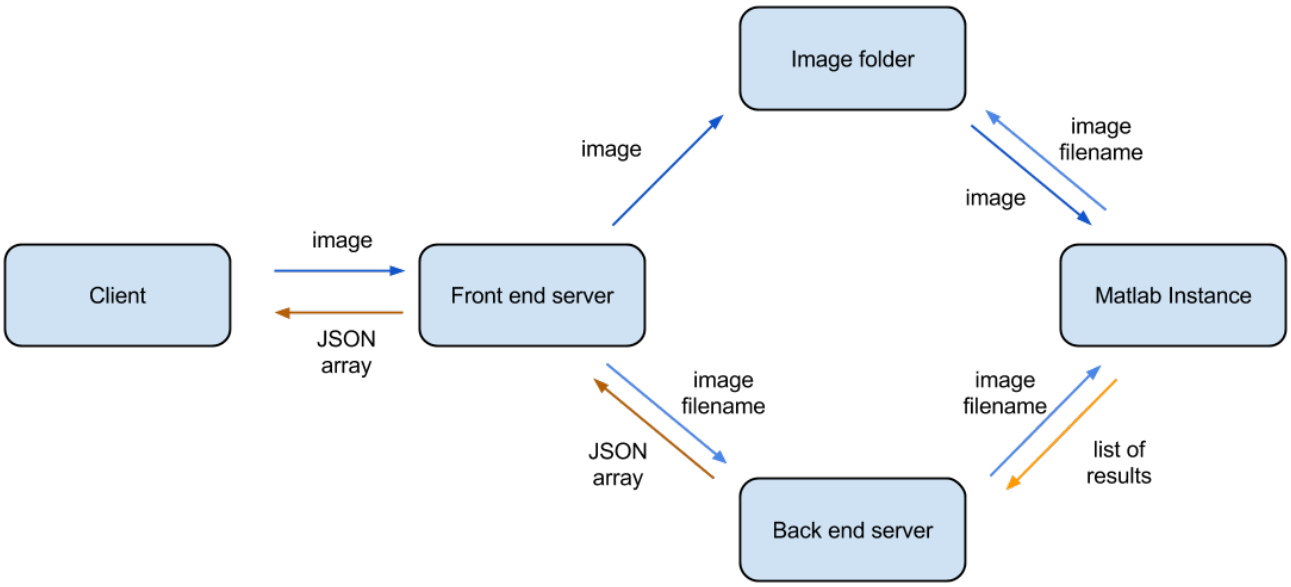


Figure 4.3: Diagram showing information flow between the different sections of the server side

4.2.1 Flask server

The front end server runs the Flask server framework[5]. The server sets up a URL and checks it for incoming POST requests. When a POST request is detected, it checks the request header for the filename, using a predetermined key. It then uses the filename to download the image and saves it to the image folder.

Error catching is implemented as the incoming POST requests cannot be assumed to necessarily be benevolent. The filetype is checked and only image files are downloaded. Flask's `secure_filename` function is used to generate filenames for the images before they are saved. The secure filename function stops filenames which could negatively affect computer behaviour, and prevents name clashes.

4.2.2 Backend server

The backend server acts as a Python wrapper around the Matlab classification code. It uses the Python to Matlab bridge Mlwrap[6], which allows Matlab functions as if they were Python functions.

Upon starting, the server opens an instance of Matlab and loads the convolutional neural network and the weight matrix W. Due to the large size of these files (245MB and 3.2MB respectively) loading takes several seconds but it only needs to be done once. By completing them upon start, they don't need to be run when a classification request is received, speeding up the classification time.

It receives the filename from the front end server, and passes the filename and CNN to the Matlab classification function. This Matlab function returns a list of the category numbers of the eight top classification results.

The backend server then constructs the JSON Array. Lists of flower names, species, image URL, Wikipedia text and Wikipedia and Google search URLs are stored. The appropriate information is chosen from the category numbers. This JSON array is passed back to the front end server which

[5] http://flask.pocoo.org/

[6] http://mlabwrap.sourceforge.net/

passes it to the client.

Chapter 5

Application design

This section covers the design of the application, the technical details of which were discussed in Chapter 4. The chapter is split into two sections: user interface and user experience. User interface considers how the application looks and how the user interacts with the application. User experience is concerned with the feel of the application; how fast, simple and intuitive it is to use. Screenshots of the application are given in Figure 5.1.

5.1 User interface design

The user interface has been optimised by following Android's design guidelines[1]. This results in an Application which looks and behaves like applications the user has seen before, making it familiar and intuitive to use.

5.1.1 Material theme

The application implements Android's recently released Material theme[2]. Material is the latest design standard released by Google. It introduces shadows, animations and code snippets which allow easy implementation of new features. Some examples of applications running the Material

[1] https://developer.android.com/design/index.html

[2] https://developer.android.com/design/material/index.html

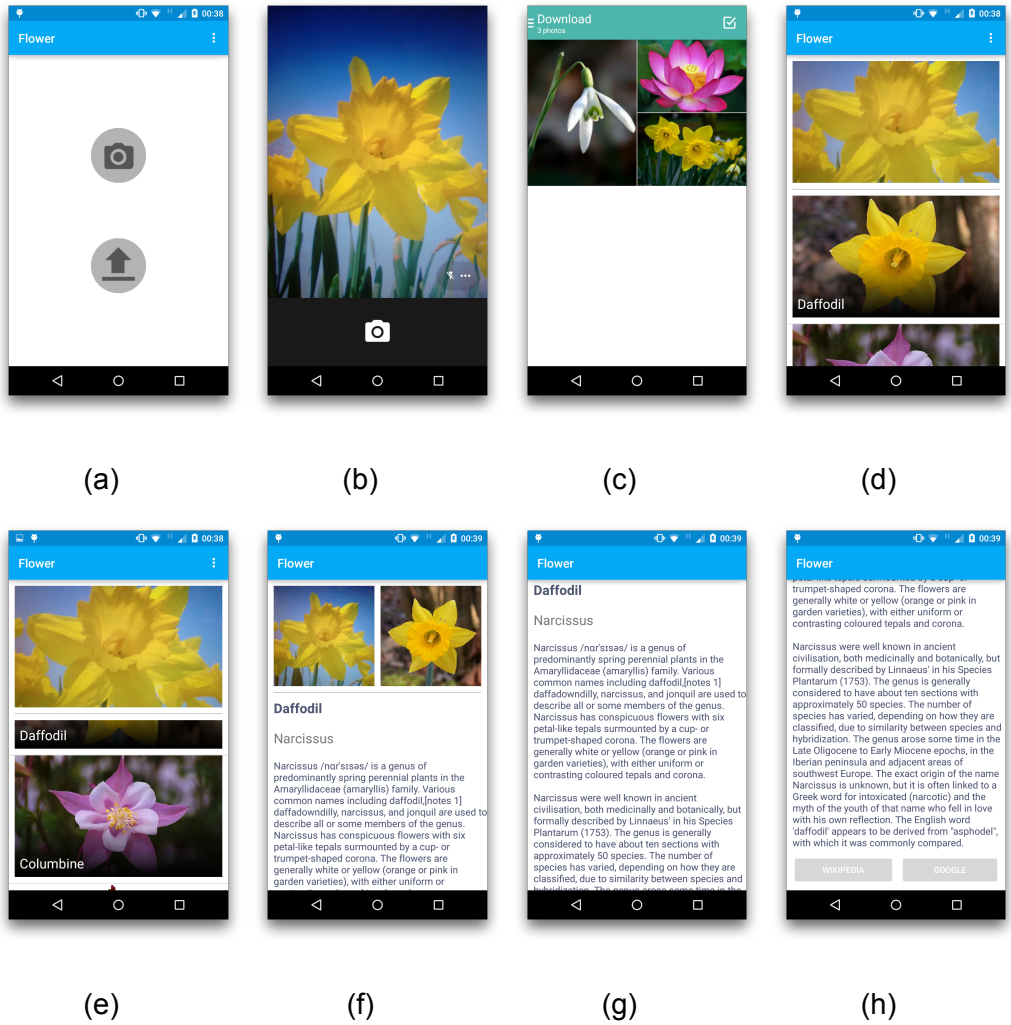


Figure 5.1: Screenshots of the application. (a): The Main Activity allows the user to choose whether to take or upload a photo. (b, c): The user takes a photo or chooses one to upload. (d, e): Results Activity. The photo is uploaded, classified and the results are displayed in a scrollable list. (f, g, h): Detail Activity. If one of the images in (d,e) is clicked, a detail view of that flower is shown.

theme are shown in Figure 5.2. The Material theme is used to keep the application familiar to Android users.

While the full Material theme can only run on devices running the latest version of Android, version 5.0 (API level 21), the look of the theme can be replicated on older devices by using Android's `AppCompat` theme.

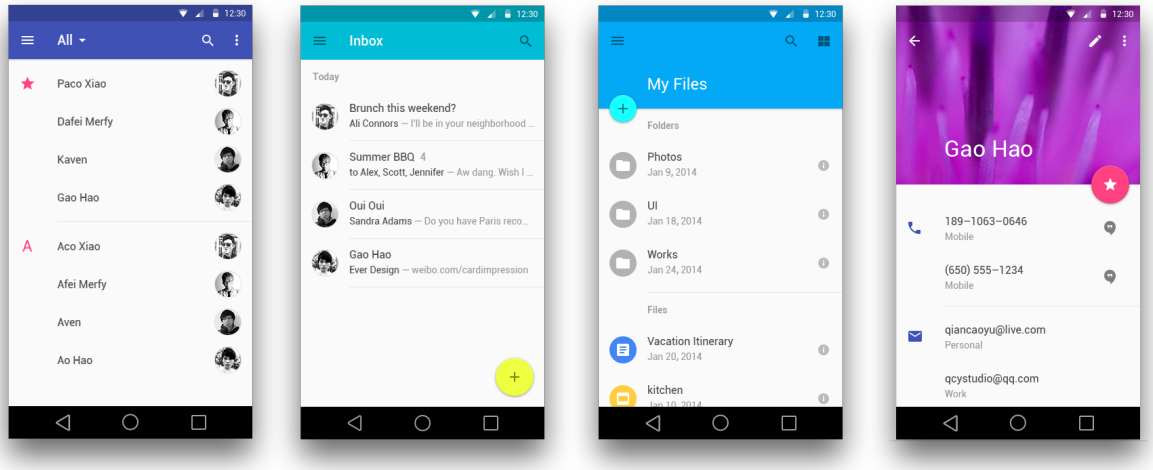


Figure 5.2: Examples of applications using the Android Material theme

5.1.2 List and detail views

The results of the classification are presented in a list (Figure 5.1 (d, e)), showing an image of the flower and the flower's common name. Above the list the photo the user has taken is shown, allowing for easy comparison. The images are cropped such that they are large enough to show the flower without taking up too much screen space. Each item on the list is clickable, and clicking reveals a page which details more information about that flower (Figure 5.1 (f, g, h)). This is a common way of presenting data and will be familiar to most users.

5.1.3 Button design

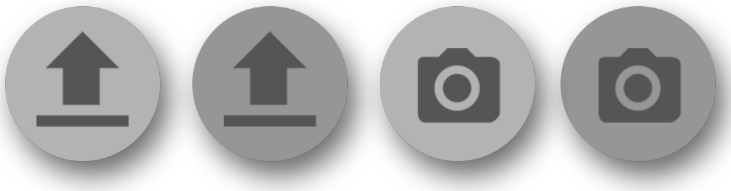


Figure 5.3: The unpressed and pressed versions of the Take Photo and Upload Photo buttons

The take photo and upload photo buttons are custom designed (Figure 5.3), and use icons rather than words for simplicity. The icons used are adapted from the official Android icon pack[3], and will be familiar to Android users. The buttons use flat colour, keeping with the Material theme. Both

[3]https://developer.android.com/design/downloads/index.html#action-bar-icon-pack

pressed and unpressed buttons have been made, and the change in colour gives visual feedback to the user that the button has been pressed.

5.2 User experience design

User experience has been optimised by focusing on making the classification as fast as possible and the application as simple to use as possible.

5.2.1 Simplification

By keeping the application simple, it is easier to use. The application presents the user with a single choice; whether to take a photo or to upload a photo which has been previously taken. Uploading and downloading happens automatically, and no menu options are provided except the ability to share the result of a classification.

5.2.2 Flower images

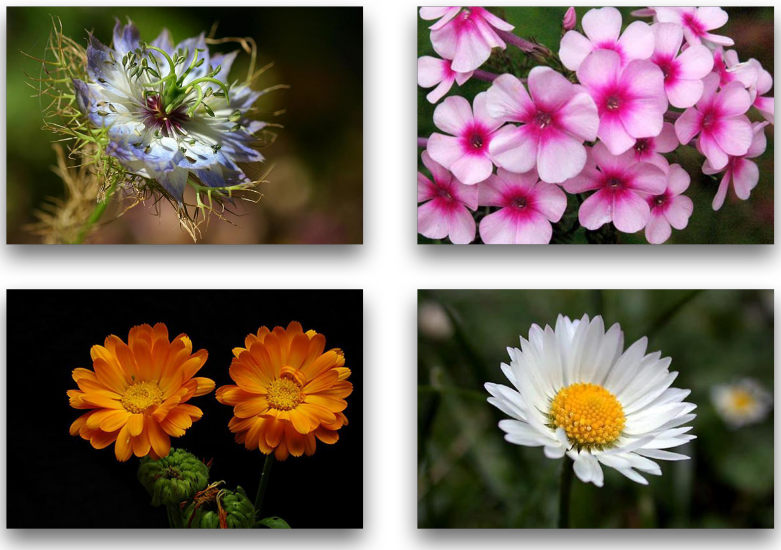


Figure 5.4: A selection of the hand-picked flower images shown to the user

After classification takes place, a ranked list of the six most likely flower species is passed to the

client and the user. An algorithm was written which finds the flower image in the dataset most similar to the photo uploaded by the user for each of the six species.

Similarity is found by finding the smallest difference $min|y_i - x|$ between the uploaded flower's feature vector x and the feature vector y_i for all i photos in the dataset.

Although this improves ease of use, this process also greatly increases classification time, as the image must be compared to each image in six categories, where the number of images in each category is up to 258. It also requires a copy of the feature vectors of all the images in the 102 dataset to be stored, which takes up 39MB of storage space. This slight increase in ease of use is not worth the increases in classification time and server footprint.

Instead of this, 102 example flowers have been hand-picked for their quality. They have been chosen because they all clearly display the flower in question whilst minimising any confusing background. The images are cropped to the dimensions they will be displayed at, minimising the amount of data which needs to be transferred and speeding up the downloading time.

Chapter 6

Summary and future work

6.1 Summary

This project applied modern computer vision techniques to the problem of flower species classification. The scope of the problem was limited to the 102 flower species laid out in the Oxford 102 category flower dataset. An accuracy of 85.6%, and a rank accuracy of 96.2% when considering the top six ranks was obtained.

A mobile application was designed and created with intuitiveness, simplicity and speed in mind. The classification pipeline was made accessible from the application, allowing free quick and accurate flower classification from anywhere in the world with an internet connection.

6.2 Future work

6.2.1 Expand flower species covered

The application currently classifies 102 flower species, and it should be a priority to expand this number as users will expect the application to work for a larger set of flowers. This could be done by obtaining a list of all common flower species in a certain geographical location, for example the UK, and then finding enough images of each of those species to train an SVM classifier. The images

could be found by contacting horticultural bodies who have access to such images, or by searching online image repositories such as Google Images or Flickr. More images for each category will increase the accuracy of the models. However, this project has obtained an effective accuracy of 96.2% using only 20 training images per species.

6.2.2 No good classification option

It would appear unpolished if the application gave normal classification results for an image which is not a flower. An additional step to the classification pipeline could be added which notifies the user if the image is not considered to be of a flower. Even if this is the case, the normal classification results would still be provided, in case the photo is in fact of a flower.

This could be implemented in two ways. A loose-grained classification step could be added. This would check whether the image is indeed a flower, before the image is passed to the fine-grained classification. Implementing this would require training another one-vs-the-rest SVM classifier which is trained on all of the flower images marked as belonging to the flower category, and images of other common items marked as belonging to the 'rest' category. These other images would have to be found. At runtime, this extra classification step requires a single dot product to be calculated and will not add much to the total classification time, and if the SVM model is trained well it should accurately distinguish between photos of, and not of, flowers.

It could also be implemented by giving a warning if highest prediction value p_i was below some threshold value T, indicating that there was low similarity between the models user's image. This is easy to implement, as only a few lines of code would have to be added, although a suitable T would have to be found. This and adds very little to the classification time, however it may not be as accurate as the above method.

6.2.3 Storing flower data on phone

Flower data such as common names, species and images could be hard written into the application. After classification, rather than sending all data associated with the classification predictions, a simple list of identification number could be sent, from which the application constructs the list

presented to the user. This would reduce classification time slightly as less data would have to be downloaded, but would increase the size of the application by around 8MB, most of which is used to store the images.

6.2.4 Porting CNN to phone

All of the classification pipeline could easily be written into the application code with the exception of the convolutional neural network. The CNN consists of layers written in C++, connected together with Matlab code. Matlab code cannot easily be compiled down to code runnable in an Android environment, so porting the CNN to Android would involve rewriting the Matlab code in C or C++, and then compiling that to Android runnable code using the Android NDK[1].

Porting the CNN to the phone would allow for complete offline classification, if the flower data were also stored on the phone, as described in 6.2.3. The effect it would have on total classification depends on several factors. Uploading and downloading time is completely removed, but as the phone has less processing power than the server, the actual classification may take longer. Whether this improves classification time will depend on a user's individual circumstances, such as the quality of their internet connection and the power of their phone.

6.2.5 Database of flowers

The flower information, Wikipedia text and images are currently stored in lists at the index of the appropriate category. A more scalable system would be to refactor the code to use a database with an entry for each flower, indexed with a unique ID. Each entry stores the flower information, Wikipedia text, weight vector and image associated with it. When booting, a weight matrix could be generated using all available weight vectors. While this would not change functionality, it would allow for new flowers to be added with greater ease.

[1] http://developer.android.com/tools/sdk/ndk/index.html

6.2.6 iPhone app

Finally, an iPhone version of this application could be written. This would take the form of a client application which takes a photo, uploads it to the same server which has already been created and then receives and displays the results.

Bibliography

[1] Y. Chai. *Recognition between a Large Number of Flower Species*. 2011. url: http://www.robots.ox.ac.uk/~vgg/research/flowers_demo/docs/Chai11.pdf.

[2] K. Chatfield et al. ``Return of the Devil in the Details: Delving Deep into Convolutional Nets''. In: *British Machine Vision Conference*. 2014.

[3] Rong-En Fan et al. ``LIBLINEAR: A Library for Large Linear Classification''. In: *Journal of Machine Learning Research* 9 (2008), pp. 1871–1874.

[4] Imagenet. *ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012)*. [Online; accessed 12-April-2015]. 2015. url: http://image-net.org/challenges/LSVRC/2012/browse-synsets.

[5] Neeraj Kumar et al. ``Leafsnap: A Computer Vision System for Automatic Plant Species Identification''. In: *The 12th European Conference on Computer Vision (ECCV)*. 2012.

[6] LISA Lab. *Convolutional Neural Networks (LeNet): DeepLearning 0.1 documentation*. [Online; accessed 3-May-2015]. 2015. url: http://deeplearning.net/tutorial/lenet.htmll.

[7] Jiongxin Liu et al. ``Dog Breed Classification Using Part Localization.'' In: *ECCV (1)*. Ed. by Andrew W. Fitzgibbon et al. Vol. 7572. Lecture Notes in Computer Science. Springer, 2012, pp. 172–185. url: http://dblp.uni-trier.de/db/conf/eccv/eccv2012-1.html#LiuKJB12.

[8] M-E. Nilsback and A. Zisserman. ``Automated Flower Classification over a Large Number of Classes''. In: *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*. 2008.

[9] A. Vedaldi and K. Lenc. ``MatConvNet -- Convolutional Neural Networks for MATLAB''. In: *CoRR* abs/1412.4564 (2014).