Machine Learning Engineer Nanodegree — Claude Michèle Nkuimi N.
**Capstone Project** — **29 january 2018**

# FLOWER CLASSIFICATION & ALGORITHM FOR AN APP

## I. Definition

- Project Overview

Flowers are viewed as symbol of grace, purity and serenity; they have unique beauty and enticing scents. Flowers are used for different purposes that can vary from ornament, medicine/pharmacy(roses & vitamine C), cooking(sunflower & cooking oil), health care(Blue cohosh & to tone the uterus in preparation for labor in the last month of pregnancy)[1], beauty... etc. Flowers are actually 'gifts of nature'. Moreover, it is widely and intensively used and bought; the consumption of flowers and indoor plants in Europe rose by 1.0% in 2016 to EUR 35.9 billion[2].

Since knowledge is the fuel that drives human life, gaining knowledge is deemed the most primary activity that prepares man for a long and successful life; subsequently, gaining knowledge about flower could be beneficial.

In this project we will try to use methodologies, functions and theories of Tensorflow library and keras Api to create Convolutional Neural Networks(CNN) and a Multi-Layer Perceptron(MLP) to classify flowers. The end objective is to create and algorithm for a flower identification App that will provide a spur-of-the moment information, namely the name of flower that is uploaded by the user.

In order to accomplish this task, Oxford 102 category flower Dataset, that contains 1020 flower training images, 1020 flower validation images and 6149 flower test images has been used.

The academic paper, machine-learning-for-flower-recognition, has been a great inspiration for this project.

[1]http://www.flowerweb.com/en/article/170044/12-Amazing-Facts-About-%20Flowers
[2]https://www.royalfloraholland.com/nl/overig/zoeken-in-nieuws/v48154/europese-consumptie-van-bloemen-en-kamerplanten-stijgt

## - Problem Statement

It is amazing how many flower types exist, getting to know about them as simple as it gets to their names is very challenging unless one spend a huge amount of time studying from encyclopedias or few websites on the Internet, which obviously is very time consuming. Imagine while walking in the nature you see a very beautiful flower that you want for your garden; you will need at least its name to be able to purchase it. Wouldn't it be perfect to take a picture, upload it on the phone and instantly get an answer?

The solution to this situation is an App(algorithm) to identify flowers.

This project aims at creating and developing a Convolutional Neural Network(CNN) to classify flowers in which each input describes the properties of an observed flower's features and the output is a flower specie. This is simply said an image classification problem; but because the dataset has more than two classes/categories it actually a multi-class classification problem for my image recognition problem.

A CNN architecture works well for the image classification task because it can be used for deep learning. Deep learning allows the process and classify data in a way similar to how the human brain processes visual input(hierarchical). This implies that each layer of a CNN is trained to recognize higher level features than the previous layer. For example, the first layer of a CNN would recognize only the color and shape of a flower; the second layer would combine this shape and color to identify higher level shapes. The next layer could recognize features like patterns on petals and texture. Then the last layer would classify the specific object which we try to classify.

The tasks involved are as follow:

- Download and preprocess the Oxford 102 category flower Dataset;
- Train a classifier that can recognize the type of flower;

Furthermore,

- Extract the flowers' features that will be fed to a classifier to recognize type of flowers;
- Train the classifier to recognize the type a flower on an image using the flowers' features;
- Write functions that will detect flowers on images and make predictions;
- Write and algorithm that will provide the name of a flower if detected or an error if no flower is detected;
- Test the algorithm with couples of images.

## - Metrics

The dataset used for this project is Oxford 102 category flower. It has 102 flower categories, a maximum of 40 images for each category. This implies that some categories might have more sample images than others and are therefore considered not be represented equally; we

can conclude that it is an imbalanced dataset. That been said, to measure classification performance or evaluating the quality of my models' predictions,
a Log loss and F1 score will be used as evaluation metrics.

Using the log_loss function from sklearn.metrics: Log loss= log_loss(y_true, y_pred, eps=1e-15)

Where y_true is ground truth (correct) labels for n_samples samples or targets and y_predict is Predicted probabilities, as returned by a classifier's predict_proba method and eps is undefined for p=0 or p=1, so probabilities are clipped to max(eps, min(1 - eps, p)).

Using the f1_score function from sklearn.metrics: F1 score= f1_score(*y_true*, *y_pred*, *labels=None*, *average='macro')*

Where y_true is ground truth (correct) labels for n_samples or targets and y_predict is predicted probabilities, as returned by a classifier's predict_proba method, labels are column indices; the parameter , average='macro', is required for multiclass/multilabel targets and calculate metrics for each label, and find their unweighted mean. It does not take label imbalance into account.

Moreover, an accuracy score will also be used just as an intuition, because it is not a good metrics for class imbalanced problems. It is the number of correct predictions made divided by the total number of predictions made; multiplying the result by 100 turns it into a percentage.

$$\text{Accuracy: } \frac{\text{Number of correct predictions}}{\text{Total predictions made}} \times 100$$

To sum up: Log Loss, F1 score and accuracy were used to measure and evaluate my CNN and MLP models and moreover to compare them.
Because CNN(transfer learning) was purposely and mostly used for the flower App(algorithm), it was not compared with neither CNN nor MLP models. This explains why validation loss, and accuracy were used just to intuitively evaluate the model.

# II. Analysis

## - Data Exploration

102 Category flower Dataset was used for this project[3]. This dataset contains images of flowers belonging to 102 different categories. The images were acquired by searching the web and taking pictures; these are color images taken from different angles, distances and have different sizes, pixel and qualities.

[3] Downloaded from http://www.robots.ox.ac.uk/~vgg/data/flowers/102/

There are a maximum of 40 images for each category. The images are contained in 102flowers.tgz and the image labels in imagelabels.mat file. Additionally, 4 distance matrices were provided D_hsv, D_hog, D_siftint, D_siftbdy but were not looked into because they were not useful for this project and subsequently not used at all.

The data splits used in this paper are specified matlab data file, setid.mat; it contains a training file (trnid) , a validation file (valid) file and a test file (tstid). The dataset  contains :

- 1020 flower training images,
- 1020 flower validation images,
- 6149 flower test images, making a total of 8189 total flower images.

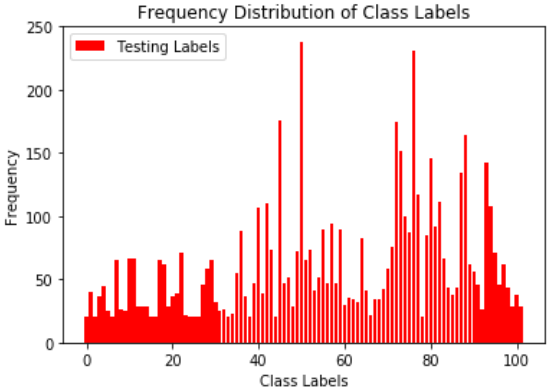
The images have large scale, pose and light variations and of sizes that vary from 532x500 up until 762x500 pixels.

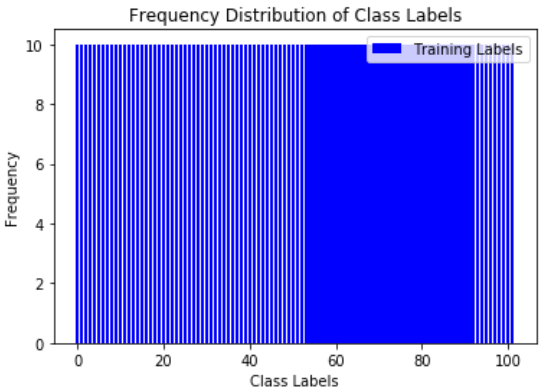
Below are sample images from the dataset:

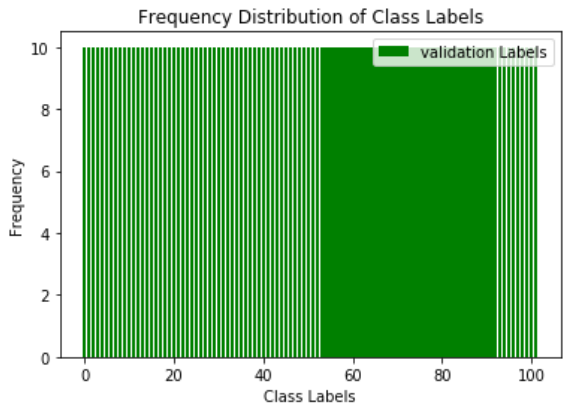
We could clearly see from the images above that they have too much data related to the background that the models might have trouble understanding. Also, some images were taken from a far distance, which might make it difficult for the models to picking up on.

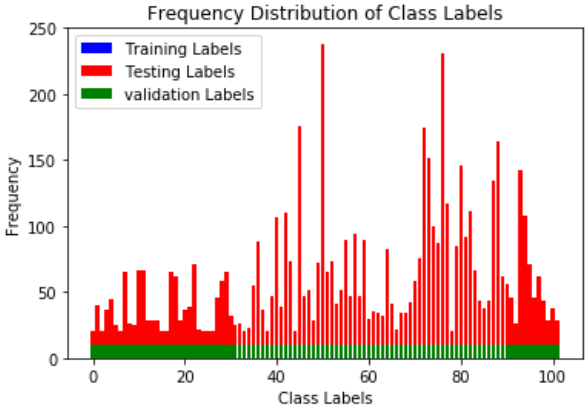
- Exploratory Visualization

We have seen from the images above how the flowers look like, in order to explore the distribution of the dataset, we make use of plots:






These plots confirm our expectation in terms of having to deal with an imbalanced dataset.

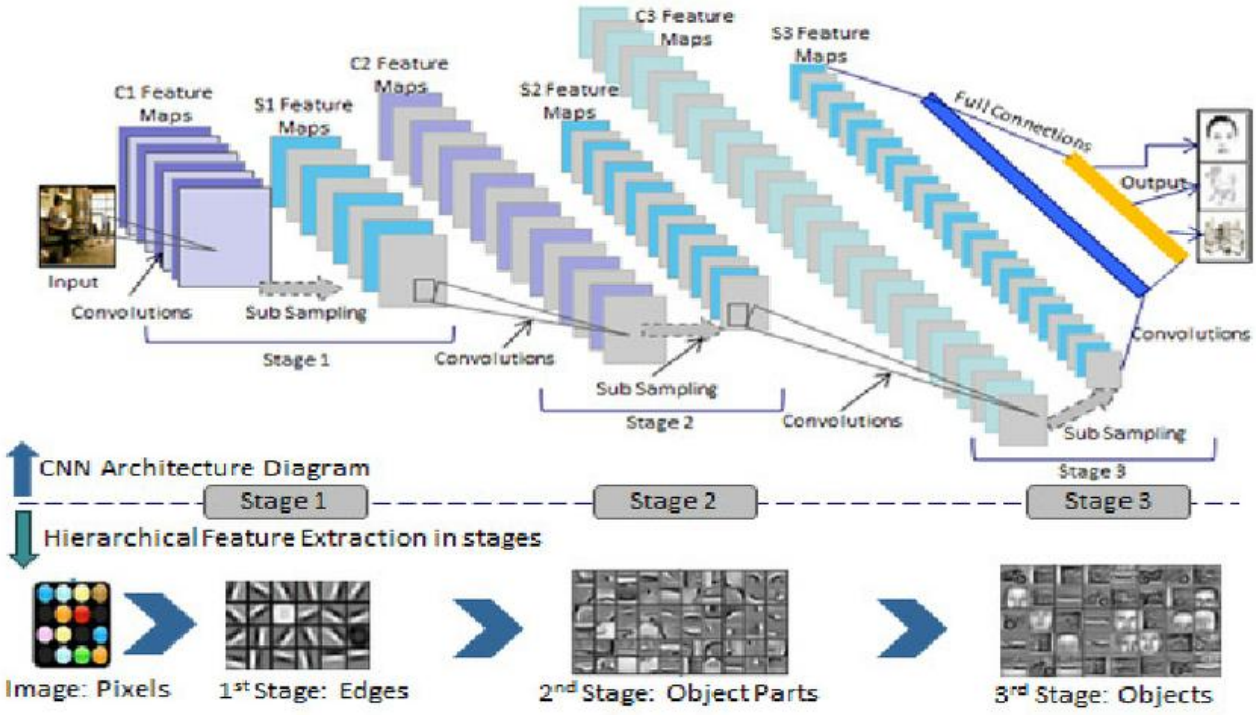
As we can see, training and validation datasets have the same distribution frequency pattern; The distribution frequency of the test dataset is highly imbalanced and completely differs from the training and validation datasets. This might make sense since they do not have the same amount of images. In other words, the training and validation datasets seem to have the same distribution pattern and frequency but are both very different from the test dataset with which they do not have the exact same number of instances in each class.

In this kind of situation where the testing dataset is totally different from the training dataset, it is hard to predict the behavior of the classifier. For example our models could have achieved very high accuracies and very low Log loss and f1_score  just because the classifiers looked at the data and cleverly decide that the best thing to do is to always predict "Class-1" and achieve high accuracy.
So, models' architecture, parameters and hyper parameters had to be cleverly selected; and we ended up with good evaluation metric values, but representative of good predictive models.

- Algorithms and Techniques

For this project, a deep learning Convolutional Neural Network(CNN) will be used.  CNNs are popular because people are achieving state-of-the-art results on difficult image classification tasks with them.

A CNN is very effective at finding patterns within images by using filters to find specific pixel groupings that are important. It needs a large amount of training data to be effective and efficient and 102 category flower dataset is large enough.
Below is a visual of how a CNN works and an overview of the entire pipeline:




Source: Udacity's review material

This project had two folds work flow:

1 CNN to classify flower and a neural network Multi-Layer Perceptron(MLP) as a benchmark model and
2 CNN using transfer learning[4] for the flower App.

In order to optimize the performance of the models, several parameters were taken into consideration and fine-tuned when possible and needed:

- Batch size: The batch size is one of the aspects of getting training right; tuning it is also very important because if the batch size is too small, then there will be a lot of variance within a batch, whilst if it is too large, the GPU will run out of memory to hold it, or training will progress too slowly.

- (Numbers of) Epochs, which is related to the number of rounds of optimization that are applied during training. More rounds of optimization leads to the reduction of the error on training data but it is also possible that at a certain point the network becomes over-fit to the training data and start to lose performance in terms of generalization to unseen data.

- The architecture of the models in terms of numbers and type of layers for example . The ConvNet architecture of my CNN stacks a few CONV-RELU layers and are followed with

[4] **Transfer learning** or inductive transfer is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem(source:https://en.wikipedia.org/wiki/Transfer_learning

POOL layers; this pattern is repeated three times. Then the transition was made to fully-connected layers. The last fully-connected layer holds the output, such as the class scores.

The CNN model's architecture looks like :

`INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2` where the * indicates repetition

INPUT [shape of training data] will hold the raw pixel values of the image(width x height x depth).

CONV layers will compute the output of neurons that are connected to local regions.

'ReLu' was added as activation function to all the hidden layers (except the last one) because it helps the function to attain a much better results.

FC (fully-connected) layer will compute the class scores, As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the number of classes.

POOL layers will perform a down-sampling operation along the spatial dimensions; their role is to reduce dimensionality of our arrays(can lead to overfitting) that might be caused by the number of filters.

102 was used as number of nodes in the final layer because it is also the number of classes in the dataset; We add two dense layers (one hidden) to generate the flower type predictions. The dense layers allow for non-linear relationships of the features to be used to classify flowers.

Finally, 'Softmax' was added as activation function to the final fully connected layer to ensure that the network outputs an estimate for the probability that each potential digit is depicted in the image.

Because I want to pad the input in such a way that output x, y dimensions are same as that of input(not loosing some nodes in the convolutional layer), I set padding = 'same'.

Our dataset is not complicated (hasn't got many different object category), it therefore does not need a very large number of filters. A filter size of 16 x32x64 allows us to express more powerful features of the input, and with fewer parameters;

The choice of optimization algorithm for your deep learning model is very important because it can influence the results. 'rmsprop' was used in this project because it is effective the field of deep learning and achieve good results.

The objective is to create a model that can accurately predict flower and measure the performance of the model; the loss function help us achieve this. My models was trained and improved by using 'sparse categorical entropy' loss function.

Dropout technique randomly ignores a subset of input weights to prevent over-fitting to the training dataset.

Concerning neural network MLP model, it's structure is as follow:

INPUT->·····→OUTPUT

it is structured by a parallel full-connection computation units arranged in layers just like the physiologic structure of the brain. There are multiple connections within and between the layers which indicate the strengths or weights between neurons that are learned under an optimization criterion.

This network consists of an input layer, one or more hidden layers of computation nodes and an output layer of computation nodes.

Class MLP classifier was used to train the model; this classifier uses a back-propagation algorithm in order to learn its parameters (weights).

The CNN model's(transfer learning) architecture looks like:

INPUT -> Globalaveragepoolinglayer-> FC

INPUT [shape of bottleneck features] will hold the raw pixel values of the image.

A Globalaveragepoolinglayer is a more extreme dimensionality reduction,  it was not needed to specify neither window size nor stride.

- Benchmark

As mentioned earlier, a CNN was used to classify flowers. The model that I used as benchmark is Neural Network[5] MLP (Multi-Layer Perceptron). Since my problem is that of a classification, I used class MLPClassifier  that implements a MLP algorithm that trains using backpropagation[6] to train the data.

Whether a model has achieved good results is somehow difficult to figure out, especially when one does not have an identical case (same problem definition, same objective and same dataset) to compare with. Let's consider accuracy which is the most common evaluation metrics used for classification; on the Internet there are hundreds of image classification examples with accuracy that differ from one another. For instance, in one example an accuracy of 9% is considered reasonable while in another one an accuracy of 70% is not satisfactory.

[5]Neural Networks are well known techniques and are generally used for classification problems, in which one can train the network to classify observations into two or more classes.

[6] A supervised learning method used to calculate the error contribution of each neuron after a batch of data is processed.

Results of MLP can been seen below:

| | Log loss | F1 score | Accuracy |
|---|---|---|---|
| Multi-Layer Perceptron | 2.6425 | 0.0565 | 11.9044% |

From these evaluation metrics, we can see that f1 score and accuracy are somehow low and Log Loss a bit high. This could be due to the fact that MLP network uses lots of parameters and all of the 2D information contained in an image are thrown away when it's matrix is flatten to a vector.

# III Methodology

- Data Preprocessing

102 category flower dataset already has a clearly separated training, validation and test datasets; it also contains image labels datasets. Those datasets were loaded using modules such as loadmat(scipy), laod_files(sklearn), glob(glob), os and shutil.

The data were then preprocessed :

For CNN:

- Keras CNNs(when using TensorFlow as backend) require a 4D array (which we'll also refer to as a 4D tensor) as input. We made use of a function (path_to_tensor) which took a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image and resizes it to a square image that is 224×224x224×224 pixels.

- The images format were converted and resized by using a format .astype('float32')/255.

For MLP:

In order to feed an image to a mlp, we must first convert the image to a vector. Additionally, to implement the MLP classifier model, data as np array was reshaped so that we could access data in CNN friendly format using .reshape((nr_train_data,1)) and labels were processed in CNN friendly format i.e. one-hot-encoding.

For CNN using tranfer learning

Data were rescaled using datagen_top = ImageDataGenerator(rescale=1. /255); training labels were converted to categorical vectors and before the algorithm was tested on images from the computer, the images were converted to RGB(*red, green, blue*).

At last, in order to use ImageDataGenerator from Keras for the  (bottleneck features), training and validation datasets had to be separated in folders. For example for training dataset:

*data→*

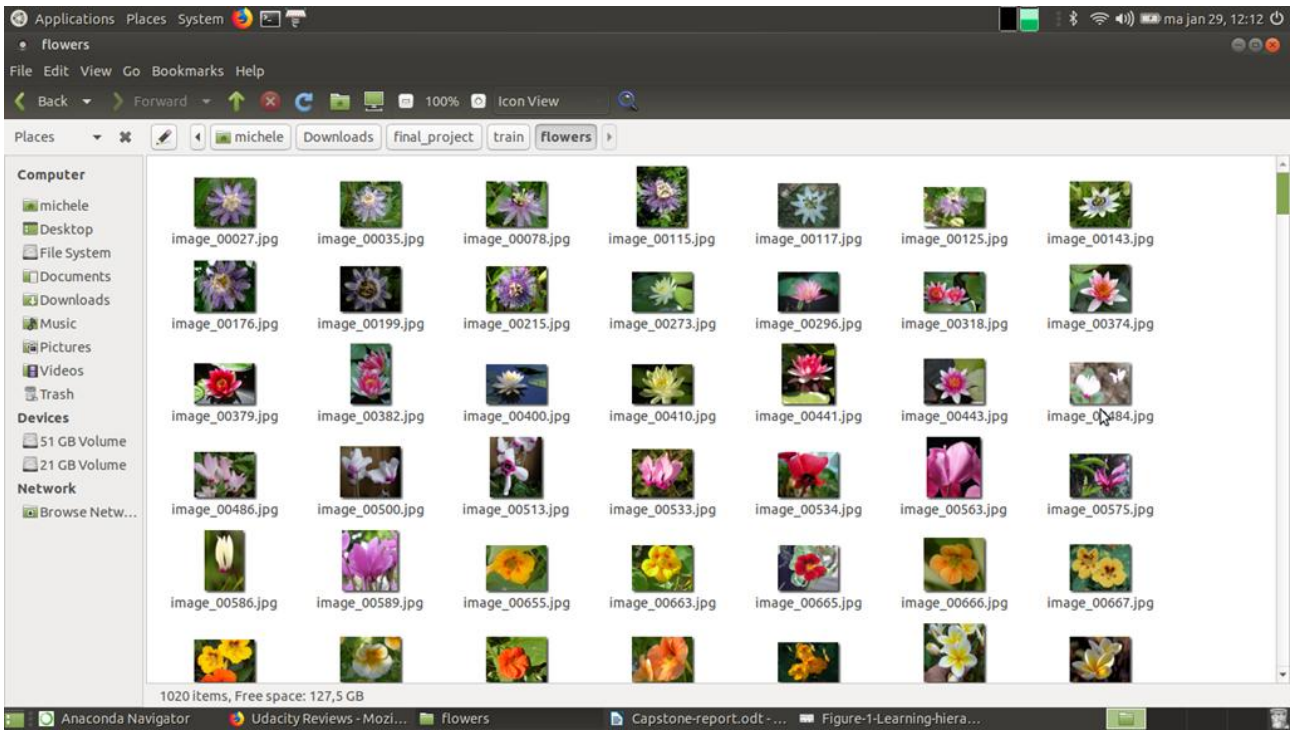
*training→*

*→ folder 1(for class 1)*

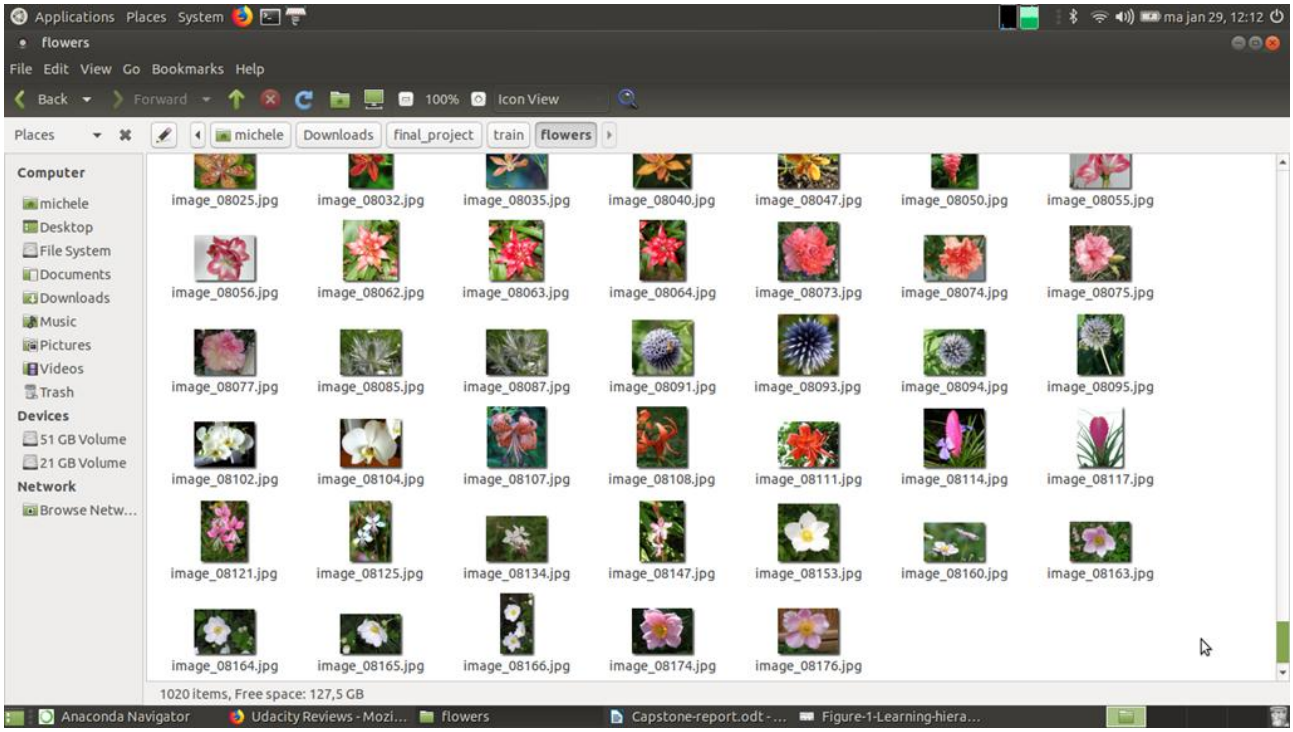
*→ folder1 (for class 2)…. etc up until class 102.*

A code was used to do this work, but 112 classes were found, instead of 102; this is because in these datasets approximately the last 30 images are somehow mixed up.

The screen-shot below shows images at the beginning of the training dataset. We could see a clear pattern: the first 10 images belong to one category, the second 10 images belong to another category and the third 10 images as well.



But, on the screen-shot below (end of the training dataset) we could clearly see that images were mixed up together and that this explains why the generator found more categories.




A manual work was done to fix this issue. Namely, cutting the images and pasting them next to similar images. And the problem was solved.

- Implementation

The CNN model consists of 3 convolutional layers, each layer consist of 2x2 kernel size with a ‘relu’ activation function, padding ‘same’, 2x2 kernel size and the depth of the filters are 16, 32 and 64. Each conv. layer is followed by a maxpool layer with 2x2 pool size. The maxpool layer is then followed by a dropout layer with 30% dropout rate. The dropout layer is followed by fully connected layers. Last fully connected layers consist of 500 nodes and a ‘relu’ activation function. This layer is followed by a dropout layer with 40% rate. The model outputs with a softmax activation. This model was created as such:

Data preprocessing →Implementing the architecture → compiling the model → training the model → loading the model with the best validation loss → testing the model.

The CNN's model.summary() is as seen below.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 28, 28, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               25088500
_________________________________________________________________
dropout_2 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 102)               51102
=================================================================
Total params: 25,150,146
Trainable params: 25,150,146
Non-trainable params: 0
```

The MLP model has a training and architecture similar to:

```
mlp = MLPClassifier(activation='relu',verbose=False, random_state=1,
                    hidden_layer_sizes=70, max_iter=max_iter,
                    early_stopping=False, batch_size=batch_size,
                    alpha=1e-6, 'solver': 'adam', 'learning_rate': 'constant',
                    'learning_rate_init': 0.09)
mlp.fit(X_train, y_train)
```

The hyper paramaters that had to be tuned in order to achieve better results were hidden_layer_sizes, max_iter, learning_rate_init and batch size

For this model to be able to train, the loss function had to be changed from categorical_crossentropy to sparse_categorical_crossentropy. It was also time consuming to figure out how to reshape the training and test data to match the shapes of the training and test labels. MLP was created as such:
Data preprocessing → Implementing the architecture & training the model → testing the model.

‘MLPClassifier' object has no attribute 'summary', it is therefore not possible to provide a summary for this model.

The CNN (transfer learning) consists of one Conv layer(Globalaveragepooling) and outputs with a softmax activation function. It was also needed to extract the bottleneck features to feed the classifier with; this process is also important to classify flowers with a higher level of accuracy.  Moreover, a function was written to obtain the bottleneck features and another function to predict flowers(that takes a path to an image as input and returns the flower that is predicted by the model). Additionally, an algorithm (that accepts a file path to an image and first determines whether the image contains a flower or not and should return the predicted type provide otherwise should  indicate an error).

Finally, the algorithm was tested on several images downloaded from the Internet; it was created as such.

Data preprocessing →extracting bottleneck features → loading Implementing the architecture → compiling the model → training the model using bottleneck features → loading the model with the best validation loss → testing the model using bottleneck features → writing the algorithm for the App → testing the model using pictures from the computer.
The CNN’s(transfer learning) model.summary() is as seen below.

```
================================= 2018-01-27 09:41
================================
Model summary:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_2 ( (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 102)               52326
=================================================================
Total params: 52,326
Trainable params: 52,326
Non-trainable params: 0
_________________________________________________________________
```

Moreover, a time rule (tijdregel()) was implemented in the notebooks to speed up the learning/ training/ predicting processes of the models.

- Refinement

Due to the fact that several issues came up, the project took longer time to be finished:

I have encountered many problems with MLP model and CNN(transfer learning) models I had to create a smaller dataset at first. A smaller dataset was used to reduced the training time of models

and  to be able to fine tune parameters really effectively and quickly before applying them to the models for the last run.

Concerning mlp classifier,  I was having a f1 score and accuracy of 0,0000%  and a log loss of less than 1 %on the test set. I basically had to change and adjust 4 parameters, mainly (max_iter = 30, batch_size =120, learning_rate_init=0.09 and hidden_layer_size=70) to be able to improve the performance of the model. The parameter that has the most positive impact on the evaluation metric was the learning rate, which provides a good balance between the speed of the training and the results. Several combinations were tried, for example :

With max_iter =400  , batch_size= 4, hidden_layer_size= 4, learning_rate_init=.1 → accuracy = 0.0000%, Log Loss=4.7269 , f1 score =0.0000

With max_iter =200  , batch_size= 50, hidden_layer_size= 20, learning_rate_init=0.01 → accuracy = 2.3256%, Log Loss=2.6269 , f1 score =0.0075

With max_iter =100  , batch_size= 80, hidden_layer_size= 50, learning_rate_init=0.06 → accuracy = 3.7567%, Log Loss=2.7523 , f1 score =0.0094

Then the best combination I found was:

With max_iter =30  , batch_size= 120, hidden_layer_size= 70, learning_rate_init=0.09→ accuracy = 11.90447%, Log Loss=2.6425 , f1 score =0.0565

Changing all the other parameters did no help the classifier to obtain a better accuracy. Also, changing the optimizer did not improve the results, so ‘Adam’ was used as optimizer/solver.

Concerning CNN (transfer learning model), the first issue came out with the bottleneck features; at first the model could not train.
After lots of ready, the training data was reshaped and resized to match the length of number of classes(simply written as:  bottleneck_features_train = model.predict_generator(generator, nr_train_images//batch_size)

The training and validation labels were one hot encoded (e.g. train_labels = to_categorical(train_labels)

I got an accuracy of less than 1% at first.  I had first to implement some changes in the bottleneck features extraction by implementing it together with vgg16 model instead of just extracting them apart an uploading them to the model.
Additionally, I changed my model architecture from:

```
#vgg16_model = Sequential()
#vgg16_model.add(Flatten(input_shape=bottleneck_features_train.shape[1:]))
#vgg16_model.add(Dropout(.7)) # make this higher for larger images (+/- 0.9 for 168) if there is a single last layer
#vgg16_model.add(Dense(nr_classes, activation='relu'))
#vgg16_model.add(Dropout(.5))
#vgg16_model.add(Dense(nr_classes, activation='softmax'))
```

to :

```
vgg16_model = Sequential()
vgg16_model.add(GlobalAveragePooling2D(input_shape=bottleneck_features_train.shape[1:]))
vgg16_model.add(Dense(nr_classes, activation='softmax'))
```

That means using  a simple architecture with  a globalaveragepooling layer.

Finally, I changed the method to calculate my accuracy from np.argmax to evaluate_generator from datagen.flow. With this model, optimizer ‘adam’ was replaced with ‘rmsprop’ which is an optimizer that utilizes the magnitude of recent gradients to normalize the gradients. Then the model’s accuracy truly improved.

# IV. Results

- Model Evaluation and Validation

I had noticed from the beginning of this project that running the notebooks with the given datasets was taking too long and was slow, making it difficult for me to correct errors, and adjust parameters. What I found convenient was to work with notebooks using reduced data. The formula I used for that purpose looked as such:

```
# for testing: reduce dataset by 4 or 5
smallerplease = True

if smallerplease :
    reduce_by = 5
    print("Watch out! We have reduced the dataset by ", reduce_by)
    setid['trnid'] = np.array(setid['trnid'][0][::reduce_by])
    setid['tstid'] = np.array(setid['tstid'][0][::reduce_by])
    setid['valid'] = np.array(setid['valid'][0][::reduce_by])
```

For example with 204 training and 1230 testing data, the MLP model obtained an accuracy of 3.7398%, a Log loss of 3.1021 and f1 score of 0.0091; Clearly, not good results.  This process was done twice per model; when I had smoothly working notebooks then I used the entire(given) data. In a matter of facts and also from machine learning documentations a model with fewer data performs poorly since more/enough data is needed to train the model.

Because I went through those steps/processed and could actually experience improvement of results with more data, I can conclude with confidence that the models are robust enough and the results can be trusted.

Training the models, especially the bottleneck features took a very long time but achieved good results.

Moreover, CNN(transfer learning) achieved an accuracy of 56.57% on the test set and was even tested using randomly selected images from the Internet. Overall I am satisfied with the test on images sample from the Internet, although a bit uncomfortable that the classifier found a flower to match to a car.

For such difficult tasks and given the type of data and the simplicity of these models, the results achieved are reasonable.

- Justification

As mentioned earlier in this report, there is no template for a a model good performance since it all depends on the dataset, problem to solve, methodologies and objectives.

As far as this project is concerned, it' s results references were Udacity projects' material(to help me determine the aspirational target of evaluation metrics), specifically dog breed classifier project that was used as a template and guideline. That being said, I am satisfied with the models because they achieved good results.

Below are the results of the evaluation metrics of my models:

| | Log loss | F1 score | Accuracy |
|---|---|---|---|
| Convolution Neural Network | 4.4012 | 0.1551 | 17.7590% |
| Multi-Layer Perceptron | 2.6425 | 0.0565 | 11.9044% |

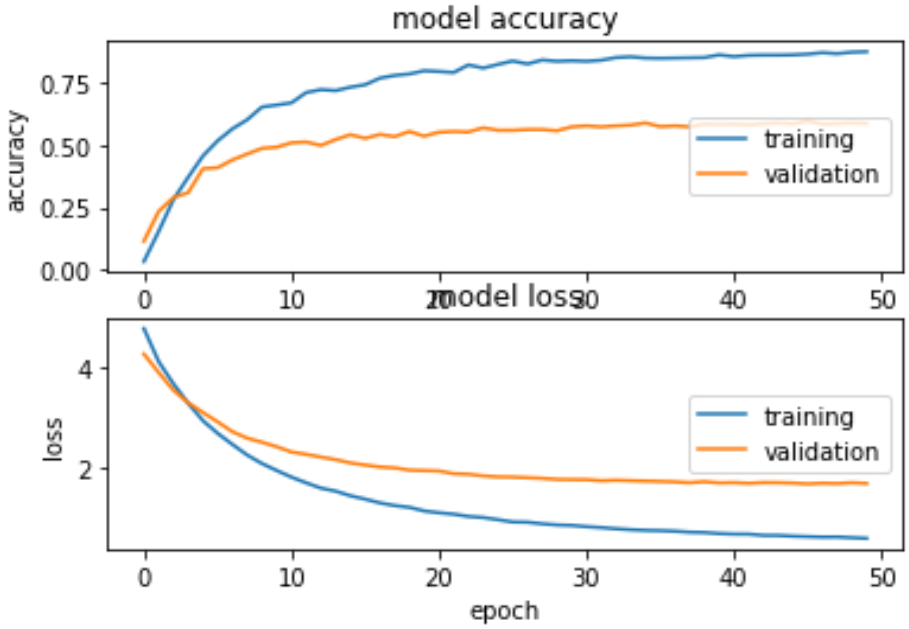
It is known from evaluation metrics documentations that F1 score reaches its best value at 1 and worst score at 0. Accuracy reaches its best value at 100 and worse at 0 whilst the lower the value of the Log loss the better it is(it varies from 0 till $\infty$). Looking at the results of the models in the table above, the performance of the CNN model was in general better than the performance of the MLP model ; but the former made much more incorrect classifications than the latter (this explains the higher value of the Log Loss).

In general the models achieved better results during the training than after the testing. This could be explained by the fact that the test set is different from the training and validation sets making predictions much more difficult.

## V. Conclusion

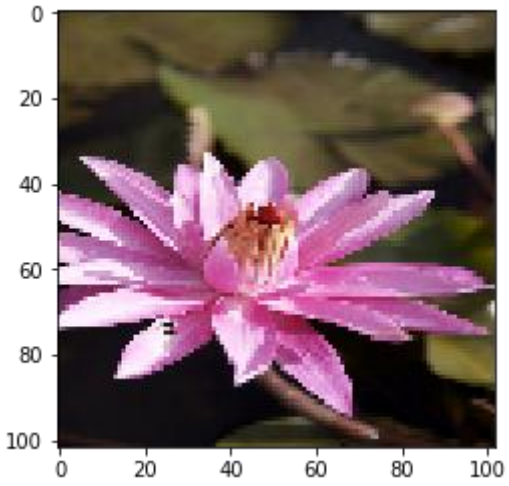
- Free-Form Visualization

The solution to our problem as mentioned in problem statement is an App (algorithm) to identify/classify flowers. Several steps were taken in order to create and implement a model (CNN using transfer learning), to write and test the algorithm that could be used for an App. The model achieved good results as seen from the training and validation plot below.

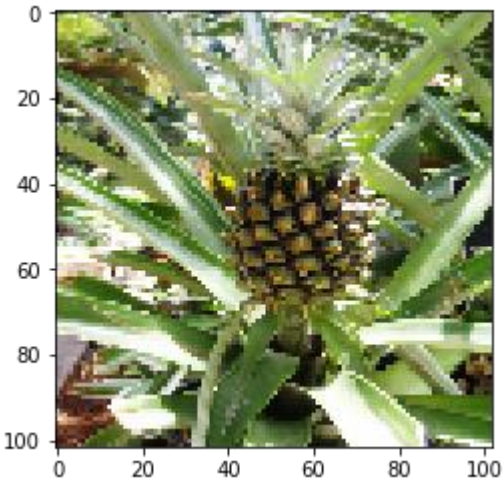



Additionally, based on the test set the model's performance resulted in 56.76% and 1.883 for accuracy and loss respectively.

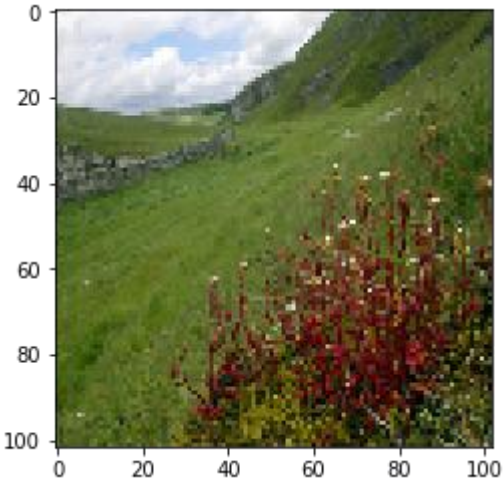
Furthermore, some images were downloaded from the Internet and used to test the algorithm. Below are some images and results provided by the algorithm.




The algorithm correctly classified this flower as water lily.




The algorithm misclassified this pineapple as a giant white arum lily.




The algorithm did a great job to classify a flower on an image taken from such a long distance as a bee balm. I, as a human would not know what type of flower is on the image.

Misclassification may be due to the algorithm functions that are neither robust enough or perfectly defined or may also due to the fact that there are too many features on the images to be learn by the classifier.

## - Reflection

The main process I used and the steps I took to accomplish this project are as follow:

* Gather the datasets;
* Process the datasets;
* Create my models
    - Implement the architecture of the models
    - Train the models
* Evaluate the performance of the models

- Get the models' predictions
- Test the models

* Obtain the results

I have come to realize that with Machine Learning to know the theory is one thing, putting it into practice is another thing. For instance, there are many things that have to be done to make a model to work such as :

- (Deep) neural network in terms of model architecture, number of layers, activation function, dropout technique?, regularizer? Convnets? MLP? pooling layers?, fully connected layers, kernel size, which 'padding', learning rate, loss functions, hidden_layer_size, batch_size etc...In other words, for a model to work and perform well there is a combination of parameters or hyper parameters that have to implemented and fine tuned. Several times, I faced issues related to the implementation of the techniques and the models because I was using wrong parameters, shapes or loss functions.

- The dataset: to understand it in terms of shape, size, resolution, features, distribution frequency and to adjust it accordingly with my models (because unless all the images are perfectly resized, the model will have great difficulty to work), was quite challenging.

Even though I found Transfer Learning fascinating, to figure out how to extract and use the bottleneck features took me a considerable amount of time.

I was amazed by how the performance of a model can be sensitive to parameters and hyper parameters and fine tuning them was quite fun.

The main objective of this project was to write an algorithm for an App to classify flowers; and because the algorithm is working I can proudly say that the objective has been attained. Considering that it is working, it can be used as solution for similar cases with of courses some improvements.

Based on the type of dataset I used, my problem definition and objective, my dog breed classification and various project from Udacity Machine Learning course, the evaluation metrics of my models scored much more than expected.

- Improvement

I have a very general rule in life: 'there is always room for improvements' and I strongly believe that someone with a strong coding skills and very deep understanding of deep learning would have achieved better result within relatively shorter time.

I believe my flower detection function does not work optimally and can be improved and in the same token the performance of the algorithm for the App would also perform better.

Only a vgg16 pretrained model was used during transfer learning; other models could also have been analyzed such as Resnet-50, InceptionV3, or Xception.

To further improve the algorithm, reducing the resolution of the images can have a positive impact on results. The images were tested using 224x224 re-sized images, whilst 168x168 or 112x112 or  even 56x56 can be more effective.