# Machine Learning Engineer Nanodegree

## Capstone Proposal

Khang Vu

March 11th, 2019

## Proposal

## Domain Background

### Background

Consider the case when we are passionate about flowers, and we are curious at the same time about what type of flower they are, what name people usually call them, etc.

What if we can do just that with a little help from technology? Imagine we can use our own phone to take pictures of the flowers, and right away, its name appears afterwards on the screen, and we satisfy. Here comes a use case where we can apply Machine Learning (ML) algorithm to make predictions.

This classification use case is one of the problem hosted by `Kaggle`, where:

> Kaggle is a platform for predictive modelling and analytics competitions in which statisticians and data miners compete to produce the best models for predicting and describing the datasets uploaded by companies and users. This crowdsourcing approach relies on the fact that there are countless strategies that can be applied to any predictive modelling task and it is impossible to know beforehand which technique or analyst will be most effective.[Kaggle]

Click here for more information in Kaggle, about `Oxford 102 Flower Pytorch – 102 Flower Classification Created by Enthusiast's` competition. Even though the competition requires a solution in Pytorch, we will instead use Keras in this project.

*References*

- Kaggle

### Purposes and Motivation

The main goal for this project is to create an intelligent Machine Learning (ML) model using different techniques to help us recognize most of the common flower types. And this model could be used to integrate into mobile apps, devices to help predict and open more opportunities for developers to innovate, especially if they are passionate about flowers.

This project will be very helpful and diverse in technical term, by using various ML techniques from Supervised Learning, Exploratory Data Analysis (EDA), to Deep Learning, etc. Apart from that, recognizing objects has been an interesting topic in recent years since it can make our applications smarter by learning and making predictions by themselves in different categories without being explicitly programmed, which is interestingly motivated to put in the efforts.

## Datasets and Inputs

There are 102 flower categories commonly occurring in the United Kingdom. *Maria-Elena Nilsback* and *Andrew Zisserman*, in *Department of Engineering Science* at the University of Oxford, have decided to create a dataset, corresponding to the aforementioned 102 flower categories, or so-called classes interchangeably. In details, each class consists of 40 to 258 images. Visualization about each class (name, image, label number, etc.) can be found at this site.

According to Visual Geometry Group at the University of Oxford:

> The images have large scale, pose and light variations. In addition, there are categories that have large variations within the category and several very similar categories. The dataset is visualized using isomap with shape and colour features.(4)

This dataset has already contained good amount of different flower types, and it especially contains images for each class. Hence, we can directly use them to start training our model without having to bring in more data images from another sources to fill up missing classes.

Within this dataset, we have three folders *Training*, *Validation*, and *Testing*, which stand for their own purposes, respectively. Each folder should contains 102 categories, and each category contains 40 to 258 images. *Training* and *Validation* folders are used for training our model, then we use *Testing* folder to validate our model after training in order to avoid *overfitting problem*. This can count as part of the pre-processing step for our dataset. Additionally, It is better to verify that our model performs well with new set of images (and not just from images that it already knew). This way we can raise our confidence that it can predict flowers' names that it has never seen before.

*References*:

- http://www.robots.ox.ac.uk/~men/
- http://www.robots.ox.ac.uk/~az/
- http://www.robots.ox.ac.uk/~vgg/data/flowers/102/categories.html
- http://www.robots.ox.ac.uk/~vgg/data/flowers/102/

# Problem Statement

## Quantifiable

Given a batch of different types of flowers, and we want to classify them by matching with their corresponding type names. In other words, we will label these flower types by printing their corresponding names under their images as results. In order to figure out the corresponding names for the flowers, we can calculate the probabilities for each classes represented by an output layer from a Deep Neural Network, which should produces the maximum likelihood of those classified names.

> A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output.(5)

And from there, we can predict the corresponding name for the image by taking the class with largest probability within 102 output predictions. These outputs are represented in form of probabilities because we use Softmax function to calculate probabilities distribution across our classes.

> In mathematics, the softmax function, also known as softargmax or normalized exponential function, is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities.

## Measurable

*Accuracy* is a metric we can use to measure our predicting performance since we can clearly observe the percentage of how many images (each represents one flower type) being correctly classified out of 102 flower types.

## Replicable

This classification problem should be reproducible by taking images of different flowers and making predictions accordingly again and again.

*References*:

- Softmax function - wikipedia
- Deep Neural Network - wikipedia

## Solution Statement

Firstly, we need to make sure our dataset is clean by pre-processing it using various Exploratory Data Analysis (EDA) techniques. Then we will be using a Deep Neural Networks (DNN) at the core to train our data, and calculate the probabilities as final output after making predictions, which will potentially tell us the flower types. We could also use Image Augmentation technique to vary the input types so that the network can learn better in terms of diversity, as part of data pre-processing step. We then use `Accuracy` metric to see how accurate our model performs after training.

> Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.(6)

After all, developers can make use of CoreML (iOS) or ML Kit (Android) to convert and incorporate this ML model into their platforms and start making predictions right on their respective devices as real applications.

*References*:

- https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15
- https://developer.apple.com/machine-learning/
- https://developers.google.com/ml-kit/

## Benchmark Model

Before diving into any algorithms, we can make a simple benchmark model to compare our model to. We need to first, splitting our dataset into *Training*, *Validation*, and *Testing* sets, and second, start using some simple pre-trained models, which have been trained through millions of images on ImageNet. We also need to check if how much our dataset resembles the images trained on ImageNet, as well as our datasets size because these two factors will affect our decisions when we fine-tune / optimize our model later. Since our dataset is pretty similar to images in ImageNet database, and our dataset is considered as small enough, we can try to use the pre-trained weights, and substituting the last layer to an output layer with 102 categories in order to fit into our problem requirements while freezing all the other layers in the model. We can treat this as one benchmark model. Additionally, it is a good idea to improve

this benchmark model by unfreezing few layers prior to the last layer and train them only. This way we can see how good the default pre-trained models can perform before making our ultimate model using Transfer Learning.

> With transfer learning, instead of starting the learning process from scratch, you start from patterns that have been learned when solving a different problem. This way you leverage previous learnings and avoid starting from scratch.[Transfer Learning]

With this approach, the benchmark model should be able to give us a sense of how accurate it can performs with the pre-trained weights without us modifying too much. It is also a good idea to do benchmarking on a simple pre-trained model, and also benchmark by using another complex model to observe better the overfitting and underfitting problem on the given model.

*References*

- Transfer Learning

## Evaluation Metrics

As mentioned in Problem Statement, we will be using `Accuracy` to measure our model's performance. In some cases, `Accuracy` is not enough to measure our performance properly and we need to use F-beta score by determining whether we need a high `Recall` score or a high `Precision` score. However, in this case, either `False Positives` or `False Negatives` is not a big problem, because they just simply mean "No, this is not a *xyz* type". So we can use `Accuracy` to measure our model performance.

*Accuracy* can be calculated as following:

```
Accuracy = (True Positives + True Negatives) / 102 (dataset size)
```

where:

- True Positives is the number of images which are correctly classified / predicted.
- True Negatives, which is not available in this case for a classification problem since we only have either "this type of flower" or "that type of flower".

Hence the formula can be shortened as:

```
Accuracy = True Positives / 102
```

Optionally, we can multiply the result by 100 to turn it into percentage format, if necessary.

# Project Design

## Data Pre-processing

Since this dataset is already guaranteed to contain about 40 to 258 images for each class (or category), each category will be represented as one folder inside a parent folder, which is one of the *Training*, and *Validation* folders. The *Testing* folder in the other hand will contain all the images for testing purpose only. We will use *Validation* folder to test our *Training* model in the process and help improve it over time. After finished, *Testing* folder will make sure our model does not overfit or underfit our validation dataset.

As the number of images in each folder varies from 40 to 258 images, it is not really balanced in general and it might affects our performance later. This can be taken care of using Synthetic Minority Over-sampling Technique (SMOTE) to make our dataset inputs become more even when fed into our network in every batch. We can also use Image Augmentation technique to generate more data using different types of transforms (Crop, Rotate, Resize, Flip transforms, etc.) to make sure our datasets is diverse for training.

From here, our dataset should be ready to be loaded into our project. We will use `load_files` method in `Scikit-learn` library, the `sklearn.datasets` module by importing it as following:

```
from sklearn.datasets import load_files
```

and load the corresponding images into our `train_files`, `valid_files`, and `test_files` variables, respectively.

## Create our DNN

We will use a Convolutional Neural Network (CNN) in Keras Neural Network API to train our dataset. While Keras using TensorFlow as backend, the CNN will require a 4-D array as input:

```
(nb_samples,rows,columns,channels)
```

Therefore, we will need to provide appropriate info taken from each image to convert to this format, we call it *Tensor*. We also need to get these 4-D *Tensors* ready for the pre-trained model (we will choose `VGG19` model in this case), by converting the RGB image into BGR (reordering the channels). And this can be done by using `preprocess_input` method, included

in any pre-trained model.

```
from keras.applications.resnet50 import preprocess_input
```

Next, we will define our DNN model using *Transfer Learning* technique, where a pre-trained model like VGG19 will be used, which has already been trained through million of images in ImageNet database. Since our current problem is an image recognition task, we can make use of most of the pre-trained weights from VGG19 to aid our own model by substituting the output layer so that it can predict flower types instead. This can be done by extracting the features when initiating the model from Keras as following:

```
from keras.applications.vgg19 import VGG19
model = VGG19(weights='imagenet', include_top=False)
```

As mentioned above, this model will first serve as a benchmark model, before we improve it by fine-tuning parameters and training some desired layers in the network as discussed in the `Benchmark` section.

## Testing and Visualization

Eventually, once our model is trained, we can check the results by visualizing them using a *Confusion Matrix* to improve on it (e.g. add more data augmentation to the classes that have been misclassified, etc.). Then we merge the validation set into the original training set, and test it against *Testing* folder to make sure it is not overfitting or underfitting our validation set. Also calculate and observe the accuracy of the model to make sure it performs well enough.

Once we are satisfied, we are ready to use this trained model to predict flower types for our problem using the maximum of the probability outputs.