# Image Classifier for 'Oxford flowers102' data set

Sravani Dhara[1] and Dheeraj Tippani[2]

[1]sravani.dhara@st.ovgu.de
[2]sai.tippani@st.ovgu.de

January 2020

**Abstract**

The purpose of this project is to build a classifier for 'Oxford flowers102' data set. This is achieved using CNN (Convolutional Neural Networks). This report will give a comprehensive explanation of the approach, methods and problems faced during the process.

# Contents

# 1 Introduction

The goal of this project is to achieve more than 25 percent accuracy in classifying the 'Oxford flowers102' image data set. This section will be followed by methodology, classification, problems faced during implementation and rectifications.

# 2 Methodology

The images from the data set are organised into respective class labels, which are in turn organized for training, validation and testing purposes. Convolutional Neural Networks are known and used for extracting effective features from the image data set. The hyper parameter choices and justification will be given in sections ahead.

The software tools and packages used for this project are presented below.

- Programming language: Python 3.6
- Integrated Development Environment: Pycharm, Google Collab notebook
- Libraries:
  - TensorFLow: An end-to-end open source machine learning platform.
  - Keras: Keras is an open-source neural-network library written in Python.
  - matplotlib: for visualisation of results and other important.
  - OS: operating system interface.
  - sklearn: Scikit-learn is a free software machine learning library

## 2.1 Oxford flowers102 Data set

The data set contains 8189 images of flowers belonging to 102 species/classes, with aspect ratio ranging aspect ratio (width/height) from 1 to 1.5. Most of the images of flowers contain cluttered backgrounds, wide range of shapes, colours and postures, in both inter-species and intra-species. The computational challenge of figuring out feature space which can uniquely distinguish across the data set of arbitrary images is extremely difficult, let alone with limited samples for several classes.

From each class 60% of the images are organised in training, 20% in validation and 20% in testing directories. The images are resized, augmented and re-scaled to normalize the images among all classes.

## 2.2 Hyper parameters

The following are the hyper parameters of the Convolution neural network that is built for the classification. At every stage of the classifier there are certain decisions taken, this section will be explaining and justifying the said decisions and choices.

- Input: Oxford flowers102 Data set contains images of flowers both in raw form and segmented form. The segmented images are not chosen because of the following reasons.
  - Not all flowers are well segmented.
  - many flowers are not centered in the images, the data augmentation of segmented images might lose certain important portions.
  - The segmented images are simply not yielding as better results as raw images.
- Data Augmentation: Extremely basic data augmentations are performed on data sets so as to increase the size of data set for certain classes with fewer images, without losing important information. This will help in reducing over fitting. Said augmentations include horizontal flipping, width shift (10%), height shift (10%). Augmentation is also necessary because the network runs for around 100 epochs, which could lead to serious over fitting.
- CNN: Convolutional Neural Networks are shift invariant and space invariant neural networks. Hand crafted features can not effectively represent the images and also cannot uniquely distinguish between classes, because many of the classes have drastically different looking flowers in colour, shape and orientation. Many inter specie flowers look similar and many intra specie flowers look different.
- Layers: The layers of the network are deeper than wider. A wider network will easily over-fit, where as a deeper network is needed for a data set like 'Oxford flowers102'. Deeper networks learn more interesting features from their previous layers.
- Activation function: Relu activation function is used for all the layers except for the final output layer, where Soft-max' activation function is used. In a deep neural network like CNN, back propagation has to reach to the very initial layers, for that relu is the best activation function.
- Dropout: 10% dropout is added at each convolution layer to reduce the over fitting happening during the training. This specific value is tweaked by trial and error.
- Epochs: The number of epochs depend on both the network and data set. We chose to run the network for 100 epochs with drop out of 10%.

# 3 Classification

## 3.1 Training

The network is trained with 4874 training images of respective classes. The image size at the first layer is 80 in width and 60 in height, which is around 1/8 the original size of the image. The scaled down version reduces the computation time. The first two layers have 12 and 20 kernels of size (3, 3) followed by a 2D max-pool layer with kernel size (2, 2) after each, followed by a dropout of 30%

There is further 1 such convolution layers with increasing number of kernels, from 20 to 40 kernels in third convolution layer. With a max-pooling and dropout layer after every convolution layers. The last convolution layer is flattened to give 1600 inputs for next Dense layer. There are 2 dense layers, one of which is output layer with soft-max as activation function. The architecture of the network can be found in Figure 1.

Learning rate is dynamic with Adam optimiser. A training accuracy of 86% is achieved. Although, the validation accuracy is up to 62% which is indicating over fitting.

### 3.1.1 CNN Architecture

The Convolutional neural network architecture used for this project can be seen in Figure 1.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 58, 78, 12)        336
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 29, 39, 12)        0
_________________________________________________________________
dropout (Dropout)            (None, 29, 39, 12)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 27, 37, 20)        2180
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 18, 20)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 13, 18, 20)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 16, 40)        7240
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 8, 40)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 8, 40)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 128)               204928
_________________________________________________________________
dense_1 (Dense)              (None, 102)               13158
=================================================================
Total params: 227,842
Trainable params: 227,842
Non-trainable params: 0
```

Figure 1: Architecture of CNN

### 3.2 Results

Accuracy of the test data achieved is 40%, where as the network achieved training accuracy of 50.5% and validation accuracy of 41%. This indicates some over fitting, this problem will be addressed and remedy will be discussed in further sections.

- Training:
  - Accuracy:50.56%
  - Loss: 1.732%
- Validation:
  - Accuracy:41.3%
  - Loss: 2.18%
- Testing:
  - Accuracy:40%
  - Loss: 2.43%

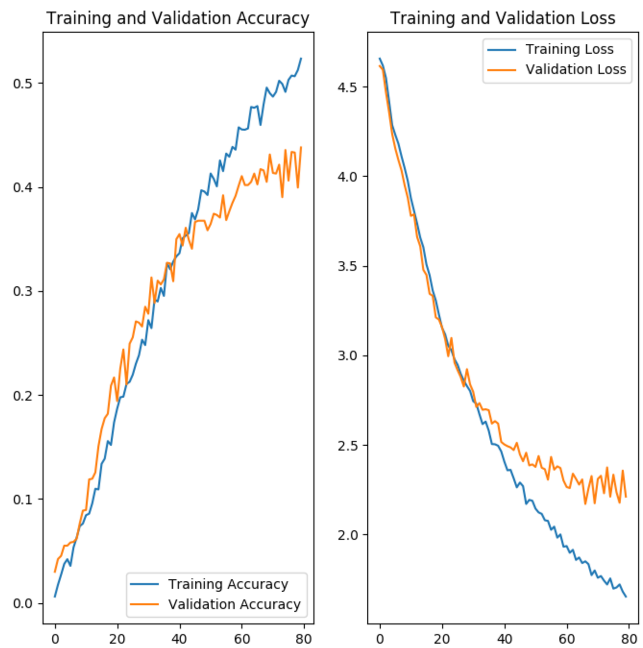


Figure 2: Loss and Accuracy values of test, train and validation data

### 3.2.1 Confusion Matrix

Figure 3 shows the confusion matrix extracted from the network from test data set. The red cells in the trace of the matrix represent the classes with zero 'true positive' instances.

These classes will be assessed further in coming sections.

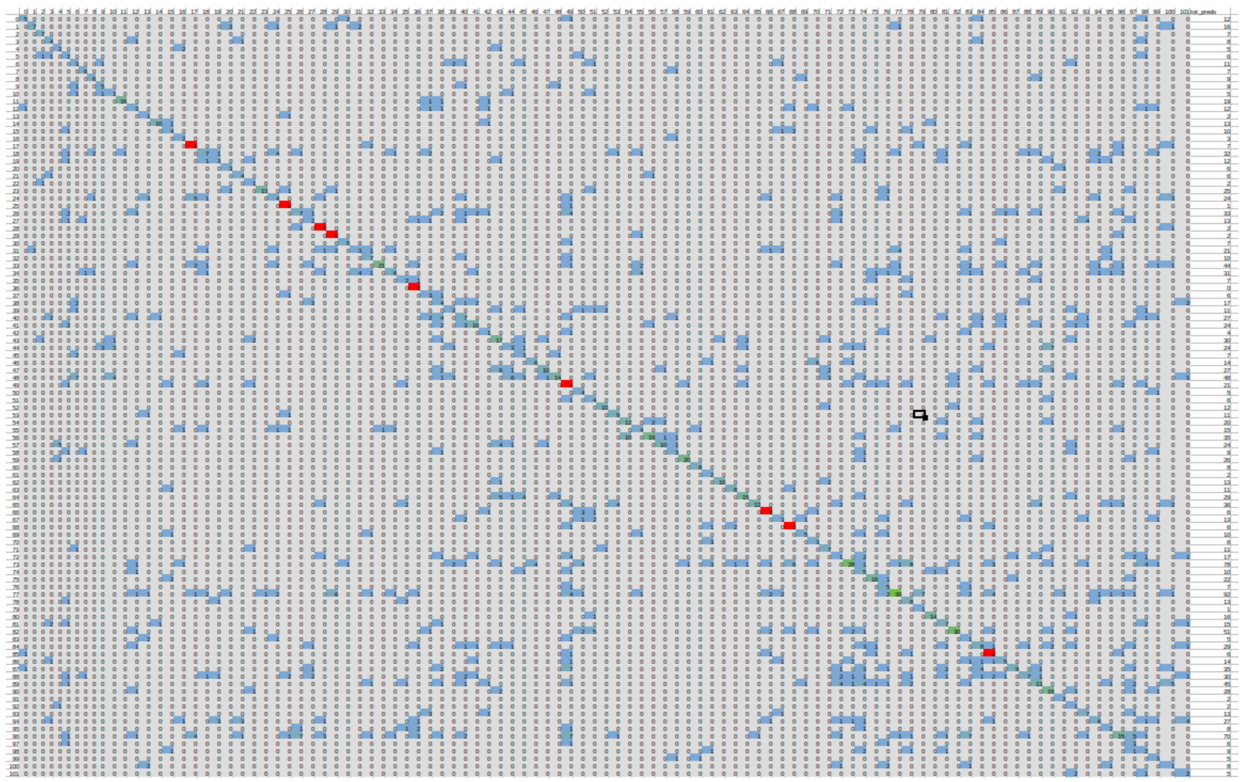

Figure 3: Confusion matrix extracted using 'sklearn'

### 3.2.2 Classification$_{r}eport$

Using sklearn library, we have extracted performance metrics like recall, precision, F1-Score and accuracy. Figure 4 shows the said metrics for each of the classes.

| | precision | recall | f1-score | | precision | recall | f1-score | | precision | recall | f1-score |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00.417 | 00.625 | 00.500 | 36 | 00.000 | 00.000 | 00.000 | 72 | 00.059 | 00.050 | 00.054 |
| 1 | 00.438 | 00.778 | 00.560 | 37 | 00.500 | 00.214 | 00.300 | 73 | 00.359 | 00.718 | 00.479 |
| 2 | 01.000 | 00.700 | 00.824 | 38 | 00.235 | 00.154 | 00.186 | 74 | 00.500 | 00.143 | 00.222 |
| 3 | 00.500 | 00.333 | 00.400 | 39 | 00.182 | 00.167 | 00.174 | 75 | 00.727 | 00.667 | 00.696 |
| 4 | 00.400 | 00.200 | 00.267 | 40 | 00.259 | 00.269 | 00.264 | 76 | 00.714 | 00.227 | 00.345 |
| 5 | 00.333 | 00.111 | 00.167 | 41 | 00.417 | 00.526 | 00.465 | 77 | 00.424 | 00.765 | 00.545 |
| 6 | 00.273 | 00.167 | 00.207 | 42 | 00.250 | 00.125 | 00.167 | 78 | 00.615 | 00.286 | 00.390 |
| 7 | 00.857 | 00.600 | 00.706 | 43 | 00.567 | 00.425 | 00.486 | 79 | 01.000 | 00.111 | 00.200 |
| 8 | 00.667 | 00.600 | 00.632 | 44 | 00.125 | 00.214 | 00.158 | 80 | 00.875 | 00.824 | 00.848 |
| 9 | 00.556 | 00.500 | 00.526 | 45 | 00.429 | 00.200 | 00.273 | 81 | 00.467 | 00.333 | 00.389 |
| 10 | 00.200 | 00.111 | 00.143 | 46 | 00.429 | 00.600 | 00.500 | 82 | 00.588 | 00.882 | 00.706 |
| 11 | 00.842 | 00.941 | 00.889 | 47 | 00.370 | 00.769 | 00.500 | 83 | 00.200 | 00.043 | 00.071 |
| 12 | 00.250 | 00.176 | 00.207 | 48 | 00.292 | 00.737 | 00.418 | 84 | 00.276 | 00.296 | 00.286 |
| 13 | 00.500 | 00.100 | 00.167 | 49 | 00.000 | 00.000 | 00.000 | 85 | 00.000 | 00.000 | 00.000 |
| 14 | 00.769 | 00.833 | 00.800 | 50 | 00.800 | 00.235 | 00.364 | 86 | 00.357 | 00.385 | 00.370 |
| 15 | 00.400 | 00.333 | 00.364 | 51 | 00.667 | 00.211 | 00.320 | 87 | 00.257 | 00.750 | 00.383 |
| 16 | 00.667 | 00.250 | 00.364 | 52 | 00.833 | 00.769 | 00.800 | 88 | 00.233 | 00.538 | 00.326 |
| 17 | 00.000 | 00.000 | 00.000 | 53 | 00.818 | 00.600 | 00.692 | 89 | 00.289 | 00.419 | 00.342 |
| 18 | 00.219 | 00.368 | 00.275 | 54 | 00.600 | 00.545 | 00.571 | 90 | 00.679 | 00.514 | 00.585 |
| 19 | 00.250 | 00.333 | 00.286 | 55 | 00.267 | 00.286 | 00.276 | 91 | 01.000 | 00.200 | 00.333 |
| 20 | 00.500 | 00.333 | 00.400 | 56 | 00.543 | 00.826 | 00.655 | 92 | 00.500 | 00.059 | 00.105 |
| 21 | 00.667 | 00.444 | 00.533 | 57 | 00.417 | 00.714 | 00.526 | 93 | 00.538 | 00.438 | 00.483 |
| 22 | 00.500 | 00.125 | 00.200 | 58 | 00.333 | 00.333 | 00.333 | 94 | 00.222 | 00.429 | 00.293 |
| 23 | 00.520 | 00.929 | 00.667 | 59 | 00.769 | 00.909 | 00.833 | 95 | 00.250 | 00.200 | 00.222 |
| 24 | 00.167 | 00.250 | 00.200 | 60 | 01.000 | 00.800 | 00.889 | 96 | 00.214 | 00.455 | 00.291 |
| 25 | 00.000 | 00.000 | 00.000 | 61 | 01.000 | 00.182 | 00.308 | 97 | 00.500 | 00.115 | 00.188 |
| 26 | 00.212 | 00.412 | 00.280 | 62 | 00.769 | 00.909 | 00.833 | 98 | 00.222 | 00.105 | 00.143 |
| 27 | 00.154 | 00.182 | 00.167 | 63 | 00.727 | 00.727 | 00.727 | 99 | 00.600 | 00.214 | 00.316 |
| 28 | 00.000 | 00.000 | 00.000 | 64 | 00.517 | 00.714 | 00.600 | 100 | 00.125 | 00.059 | 00.080 |
| 29 | 00.000 | 00.000 | 00.000 | 65 | 00.333 | 00.923 | 00.490 | 101 | 00.400 | 00.154 | 00.222 |
| 30 | 00.714 | 00.625 | 00.667 | 66 | 00.000 | 00.000 | 00.000 | | | | |
| 31 | 00.143 | 00.333 | 00.200 | 67 | 00.231 | 00.273 | 00.250 | accuracy | 0.4042806 | 0.404281 | 0.40428 |
| 32 | 00.700 | 00.467 | 00.560 | 68 | 00.000 | 00.000 | 00.000 | macro avg | 0.4340789 | 0.390856 | 0.37236 |
| 33 | 00.477 | 00.955 | 00.636 | 69 | 00.500 | 00.625 | 00.556 | weighted avg | 0.4272337 | 0.404281 | 0.37717 |
| 34 | 00.194 | 00.500 | 00.279 | 70 | 00.500 | 00.231 | 00.316 | | | | |
| 35 | 00.286 | 00.222 | 00.250 | 71 | 00.636 | 00.438 | 00.519 | | | | |

- Accuracy = 40.42%
- Precision = 42.7%
- Recall = 40.42%
- F1 score = 37.7%

Notice the classes highlighted in red, these classes showed zero 'true positive' in the confusion matrix. The classes 18, 26, 28, 29, 36, 49, 66, 68, 85 were miss-classified as other classes. The reasons for each of those classes are class specific in nature.

### 3.2.3 Inference from result

Some of the classes mention in earlier section have inexplicable reasons. How ever, some of the classes can be assessed based on the metric data obtained. One such class is class18, it has too many variation between all flowers of same class.

# 4 Problems faced and rectification

## 4.1 Processing time

### 4.1.1 Problem

There are around 15 hyper parameters which should be tweaked when a desired result is not evident. These parameters include number of convolution layers, number of dense layers, number of kernels per layer, optimizer functions, loss functions, stride, drop outs, number of epochs, data augmentation etc. While the network took around 1-2 hours at least per configuration of these hyper parameters, this makes it important to trial with extreme caution.

### 4.1.2 Rectification

- Reduction of image target shape by factor of 2, 4, 8.
- Adding max pooling layers after each convolution layer to reduce the number of trainable parameters.
- Introduction of Stride in initial layers is experimented with, in the initial layers.

## 4.2 Over fitting

### 4.2.1 Problem

With the disparity in training and test data accuracy we can tell that the network is over fitting. The problem persists due to large number of epochs and limited images in certain classes.

### 4.2.2 Rectification

- Data augmentation of the images in data set.
- Reducing number of epochs.

# 5 Conclusion

The were able to achieve 40% of test data accuracy with our Convolution neural network based classifier. There is future scope in reduction of over fitting and experimenting with transfer learning.