Image classification

This project demonstrates how to identify flower photos. It uses a tf.keras.Sequential model to generate an image classifier and tf.keras.utils.image dataset from directory to load data. You'll get hands-on practise with the following concepts:

Loading a dataset from disc in a quick and efficient manner. Detecting overfitting and using strategies like data augmentation and dropout to mitigate it. A basic machine learning workflow is followed in this tutorial:

Analyze and comprehend data Construct an input pipeline. Create the model. Educate the model Validate the model. Improve the model and go through the process again.

## **Explore the dataset by downloading it.**

After downloading, you should now have a copy of the dataset available. There are 357 total images

Let's use the tf.keras.utils.image dataset from directory utility to load these photos from disc. In just a few lines of code, you can go from a directory of photographs on disc to a tf.data.Dataset. You can alternatively visit the Load and preprocess pictures page to develop your own data loading code from scratch if you like.

When constructing your model, it's a good idea to employ a validation split. Let's say we'll utilise 80% of the images for training and 20% for validation.

Found 357 files belonging to 7 classes.

Using 286 files for training.

Found 357 files belonging to 7 classes.

Using 71 files for validation.

## **Visualize the data**

In a moment, you'll feed these datasets to Model.fit to train a model with them. You may also manually traverse through the dataset and retrieve batches of photographs if you prefer.

The image\_batch is a tensor of the shape (32, 180, 180, 3). This is a batch of 32 images of shape 180x180x3 (the last dimension refers to color channels RGB). The label\_batch is a tensor of the shape (32,), these are corresponding labels to the 32 images.

You can call .numpy() on the image\_batch and labels\_batch tensors to convert them to a numpy.ndarray.

## **Configure the dataset for performance**

Let's make sure we're using buffered prefetching so we can get data from disc without I/O becoming blocked. When loading data, you should use the following two methods:

After the images are loaded from disc during the first epoch, Dataset.cache maintains them in memory. This will prevent the dataset from becoming a bottleneck during the training of your model. You can also use this method to establish a performant on-disk cache if your dataset is too huge to fit in memory. During training, Dataset.prefetch combines data preprocessing and model execution. In the Prefetching part of the Better performance with the tf.data API guide, interested readers can learn more about these methods, as well as how to cache data to disc.

## **Standardize the data**

The RGB channel values are in the [0, 255] range. This is not ideal for a neural network; in general you should seek to make your input values small.

Here, you will standardize values to be in the [0, 1] range by using tf.keras.layers.Rescaling

# Create the model

Each of the three convolution blocks (tf.keras.layers.Conv2D) in the Sequential model has a max pooling layer (tf.keras.layers.MaxPooling2D). A ReLU activation function ('relu') activates a fully linked layer (tf.keras.layers.Dense) with 128 units on top of it. The purpose of this lesson is to demonstrate a standard approach, not to tune this model for high accuracy.

### **Compile the model**

Choose tf.keras.optimizers for this lesson. tf.keras.losses and Adam optimizer Loss function for SparseCategoricalCrossentropy. Pass the metrics parameter to Model.compile to see training and validation accuracy for each training period.

## **Model summary**

View all the layers of the network using the model's Model.summary method:

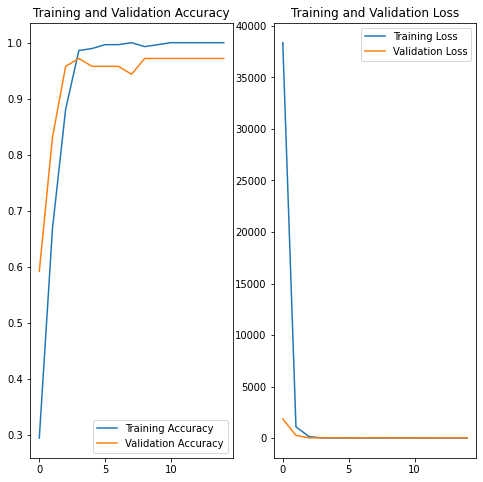
## **Train the model**

Epoch 15/15

9/9 [==============================] - 18s 2s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val\_loss: 16.3510 - val\_accuracy: 0.9718

## **Visualize training results**

On the training and validation sets, plot the loss and accuracy:



The plots show that training accuracy and validation accuracy are bit close to each other, and the model has achieved around 95% accuracy on the validation set. The training accuracy is increasing timely. In case of overfitting or less accuracy of validation, we can use data augmentation and droupout techniques to overcome the overfitting and better accuracy of validation data

## **Predict on new data**

Finally, let's put our model to the test by classifying an image that wasn't in the training or validation sets.

his image most likely belongs to Left\_Turn with a 100.00 percent confidence.