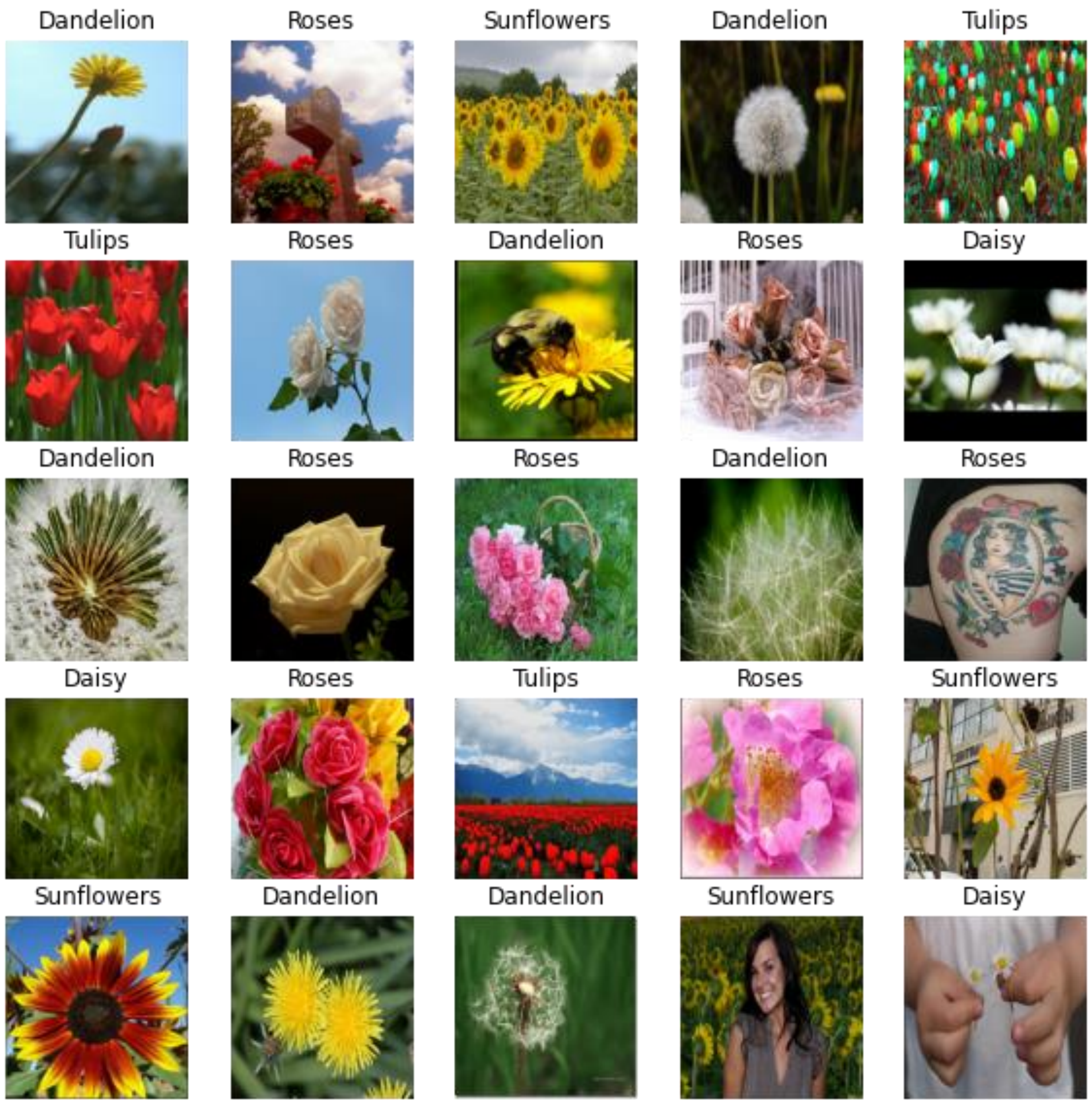
# Classifying Flower Images with CNN

Fardous Sabnur

Machine Learning 725



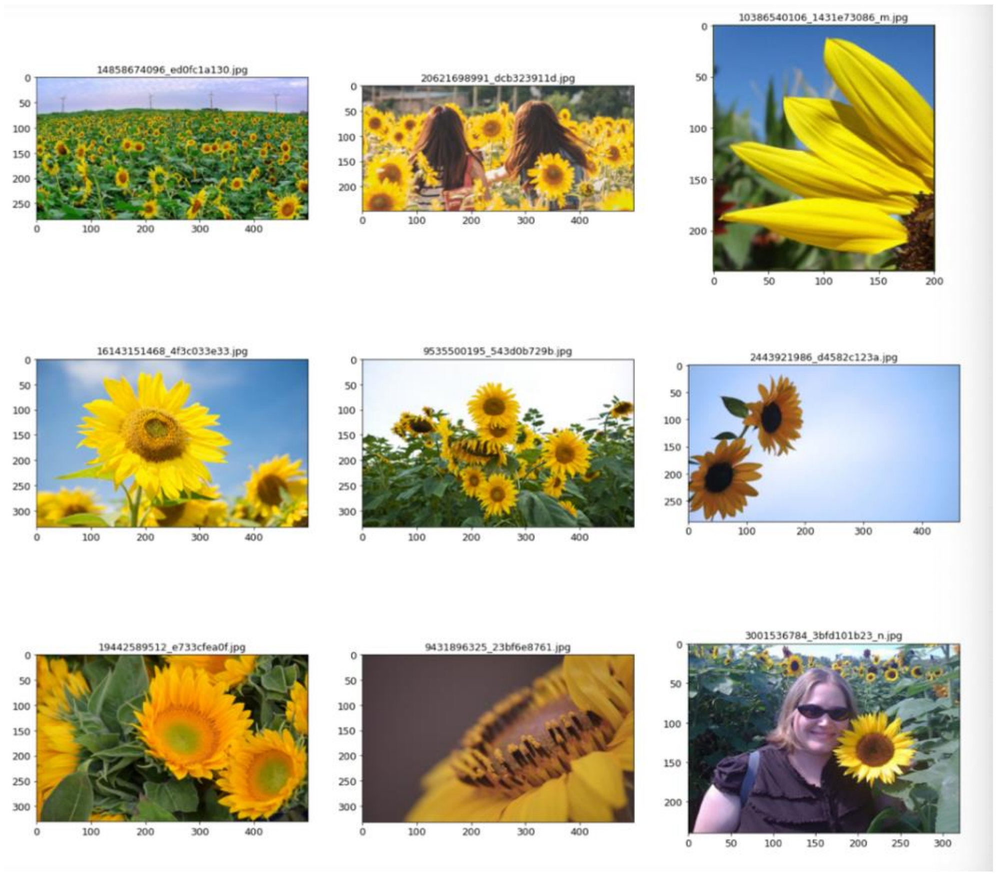
## 1 Introduction

This paper will provide classification for a multiclass flower dataset in Python. First, I will describe the data and perform some basic data analysis tasks. Next, I will implement a simple Convolutional Neural Network model and the ResNet 50. Then I will look at their performance and effectiveness in modeling and classifying the flower images.

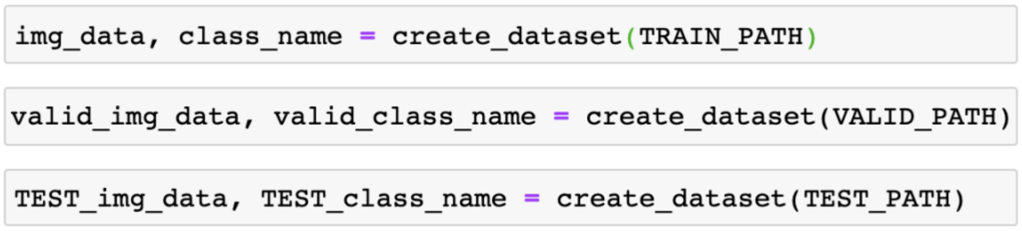
## 2 Data

I used the Multiclass Flower Classification dataset from Kaggle. The dataset includes raw jpg images of five types of flowers. The flower types are daisy, dandelion, roses, sunflowers, tulips.

**Figure 1:** Below are few images of sunflowers from the raw dataset before doing any analysis. Notice how they have different shapes and orientations


14858674096_ed0fc1a130.jpg
20621698991_dcb323911d.jpg
10386540106_1431e73086_m.jpg
16143151468_4f3c033e33.jpg
9535500195_543d0b729b.jpg
2443921986_d4582c123a.jpg
19442589512_e733cfea0f.jpg
9431896325_23bf6e8761.jpg
3001536784_3bfd101b23_n.jpg

- The initial zip-file was partitioned into test, training, and validation folders. I created datasets for each of the folders using their path directories. The code snippet below shows the datasets for the three different folders.

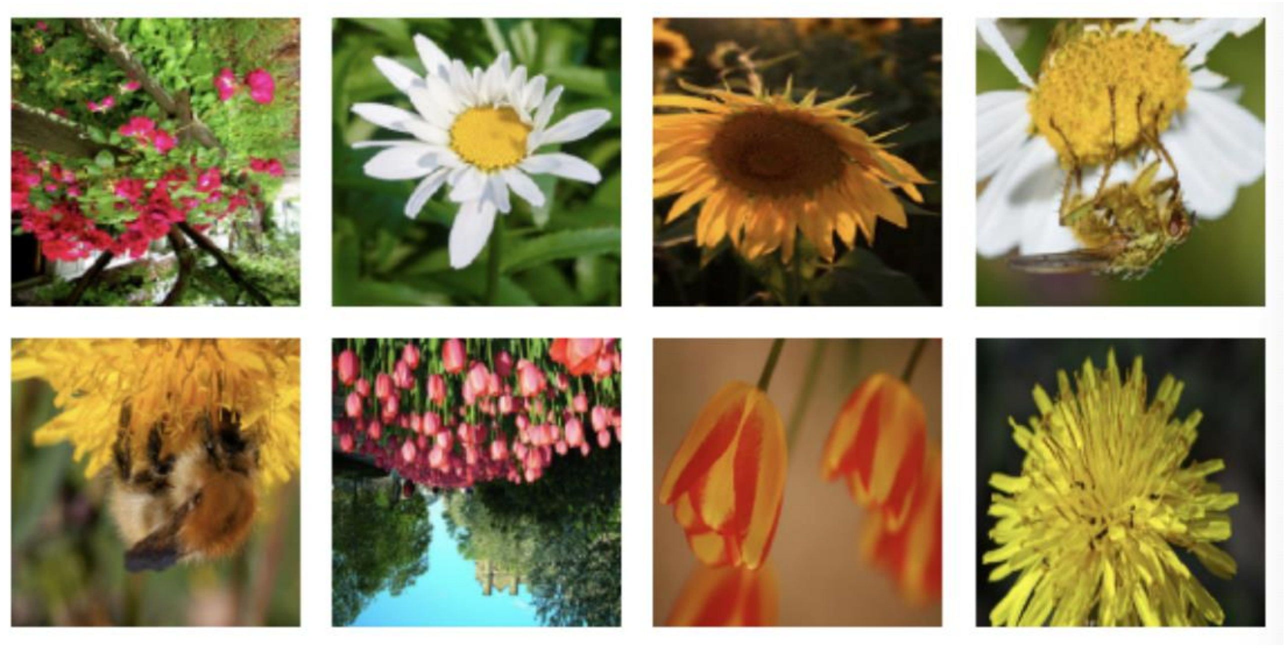


```
img_data, class_name = create_dataset(TRAIN_PATH)
```

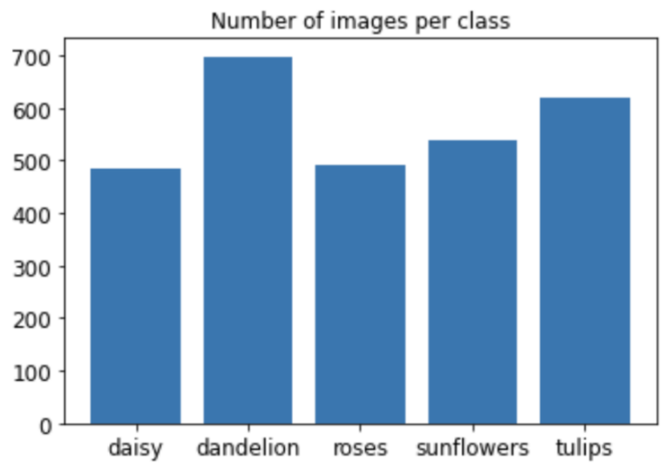
```
valid_img_data, valid_class_name = create_dataset(VALID_PATH)
```

```
TEST_img_data, TEST_class_name = create_dataset(TEST_PATH)
```

- **Figure 2:** Next, I augmented the data using the ImageDataGenerator function which generates batches of tensor image data with real-time data augmentation. I tuned the hyperparameters of this function to rescale all the images to have the same shape. Then it randomly flipped the images horizontally and vertically. Data augmentation is used to prevent overfitting.



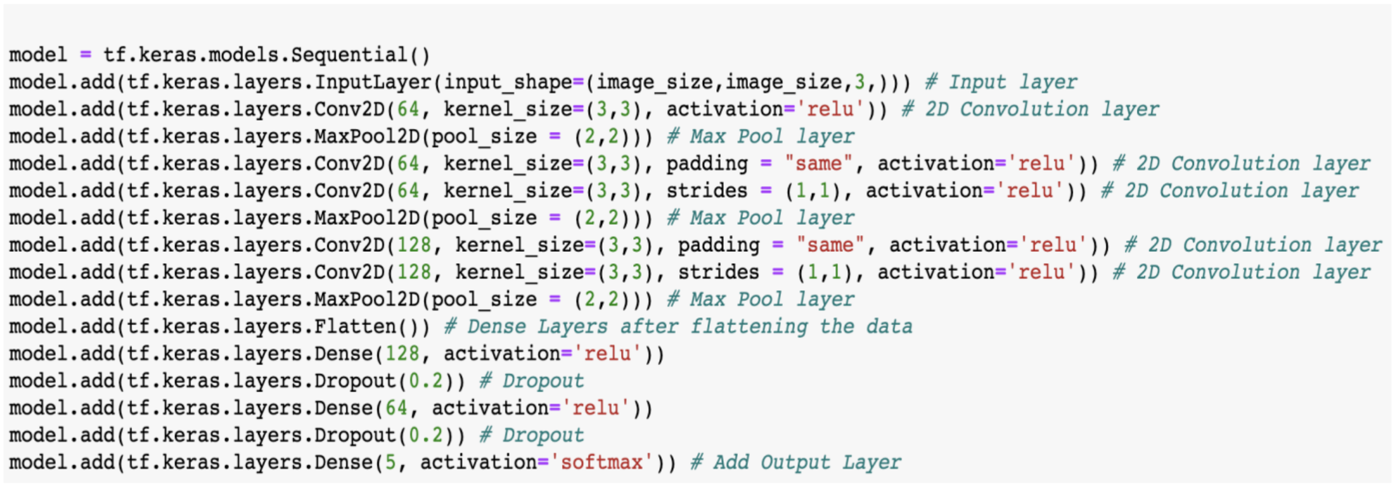
- **Figure 3**: The histogram below is checking the number of flower images per categories. Dandelion has the highest number of images, and daisy has lowest number of images. The difference between the different classes isn't that drastic, so I decided not to change anything for the number of images per class.



## 3 Analysis Methods

### 3.1 Simple CNN model

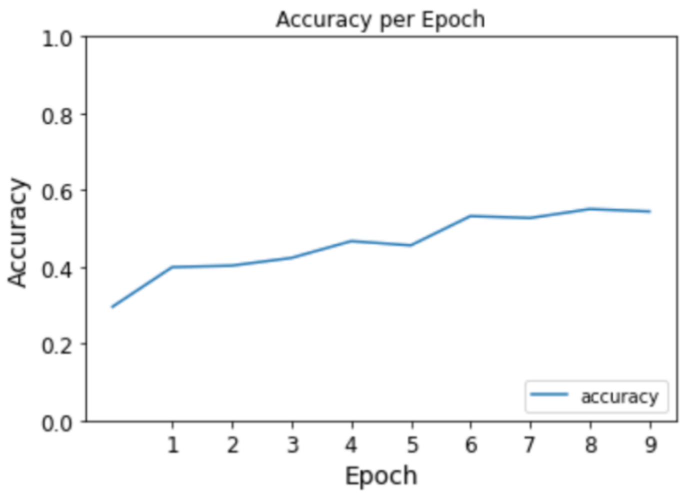
- The code snippet shows a simple Convolutional Neural Network. The model structure repeats two convolutional layers followed by a max pooling layer. The convolutional layers use the "relu" activation function. Next, it has two hidden dense layers and a dense output

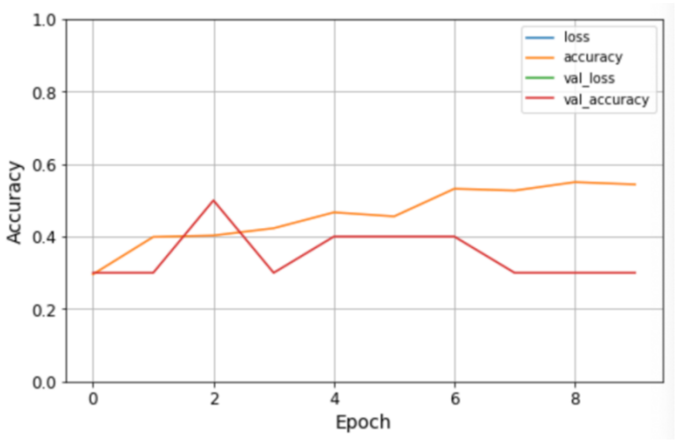


```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.InputLayer(input_shape=(image_size,image_size,3,))) # Input layer
model.add(tf.keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu')) # 2D Convolution layer
model.add(tf.keras.layers.MaxPool2D(pool_size = (2,2))) # Max Pool layer
model.add(tf.keras.layers.Conv2D(64, kernel_size=(3,3), padding = "same", activation='relu')) # 2D Convolution layer
model.add(tf.keras.layers.Conv2D(64, kernel_size=(3,3), strides = (1,1), activation='relu')) # 2D Convolution layer
model.add(tf.keras.layers.MaxPool2D(pool_size = (2,2))) # Max Pool layer
model.add(tf.keras.layers.Conv2D(128, kernel_size=(3,3), padding = "same", activation='relu')) # 2D Convolution layer
model.add(tf.keras.layers.Conv2D(128, kernel_size=(3,3), strides = (1,1), activation='relu')) # 2D Convolution layer
model.add(tf.keras.layers.MaxPool2D(pool_size = (2,2))) # Max Pool layer
model.add(tf.keras.layers.Flatten()) # Dense Layers after flattening the data
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.2)) # Dropout
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dropout(0.2)) # Dropout
model.add(tf.keras.layers.Dense(5, activation='softmax')) # Add Output Layer
```

layer with the softmax activation function. Then the inputs are flattened since a dense network expects a 1D array of features for each instance. The model also has two dropout layers, with a dropout rate of 20% each, to reduce overfitting. I could add better activation functions and batch normalization for the model to perform a lot better but I'll keep this one simple to set the expectations for the more complex model ResNet 50.

- **Figure 4 and Figure 5:** The graph on the left side shows the accuracy rate per epoch for the simple model above. The graph on the right shows the training accuracy and validation accuracy rate. With the simple model, the accuracy rates for classifying flower images are around 54%



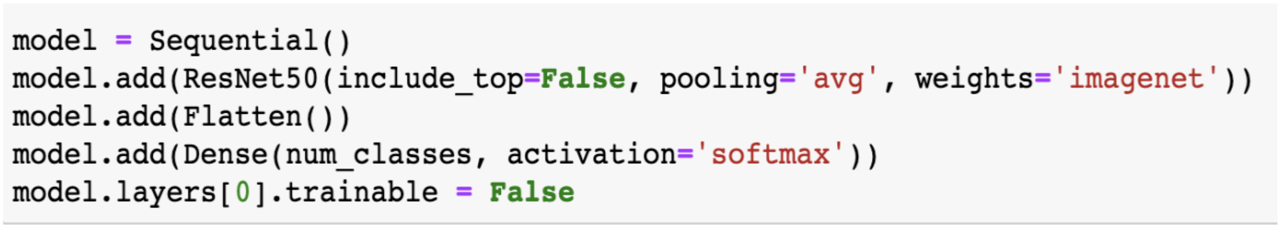


## 3.2 Resnet 50

The residual network is a very well-performing CNN. The model keeps getting deeper and deeper, with fewer and fewer parameters. This deep network is trained using skip connections (also called shortcut connections): the signal feeding into a layer is also added to the output of a layer located a bit higher up the stack. The architecture of ResNet is just like the GoogLeNet (except without a dropout layer). In between is just a very deep stack of simple residual units. Each residual unit is composed of two convolutional layers without any pooling layer, with

Batch Normalization (BN) and ReLU activation function, using 3 × 3 kernels and preserving spatial dimensions with stride 1, Padding = “same”.

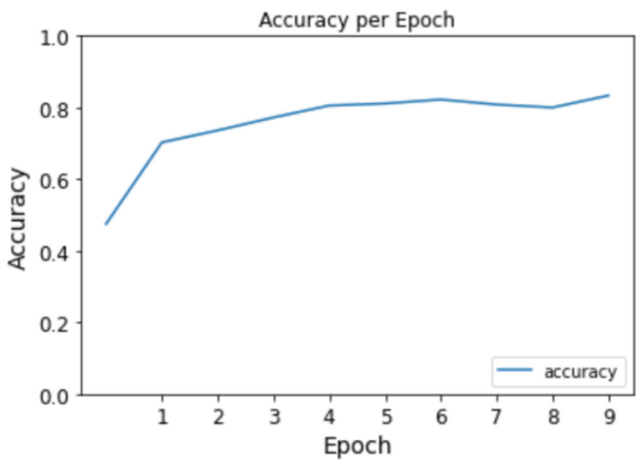
The images I’m using had to be preprocessed in a way to fit this model. This is what the resulting model looked like:

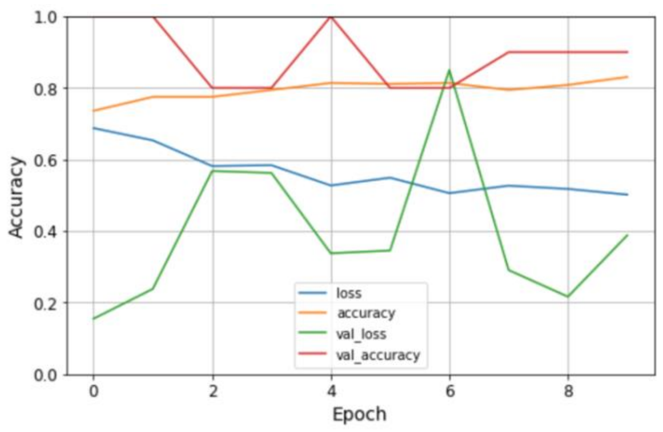


```
model = Sequential()
model.add(ResNet50(include_top=False, pooling='avg', weights='imagenet'))
model.add(Flatten())
model.add(Dense(num_classes, activation='softmax'))
model.layers[0].trainable = False
```

The weights for the pretrained model were taken from the “imagenet”, then I just flattened the data and added a dense layer with softmax activation function.

- **Figure 6 and Figure 7:** Just like the simple model, the graph on the left side shows the accuracy rate per epoch for the ResNet model above. The graph on the right shows the training accuracy, training loss and validation accuracy, validation loss. With the simple model, the accuracy rates for classifying flower images were around 20% but for this model the accuracy rates for classifying the training data for the flower images are around 80%

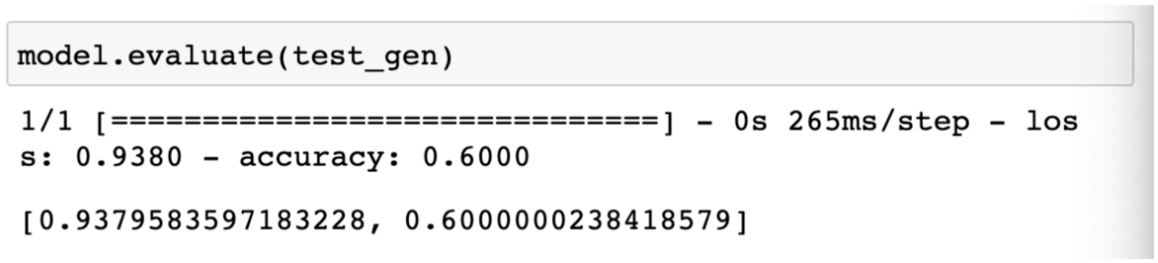




# 4 Conclusion

**Performance of test set:** For the ResNet model the accuracy rate for test set was 76% and for the simple model it's only 60%. The simple model has higher loss rate than ResNet model.

Simple model:

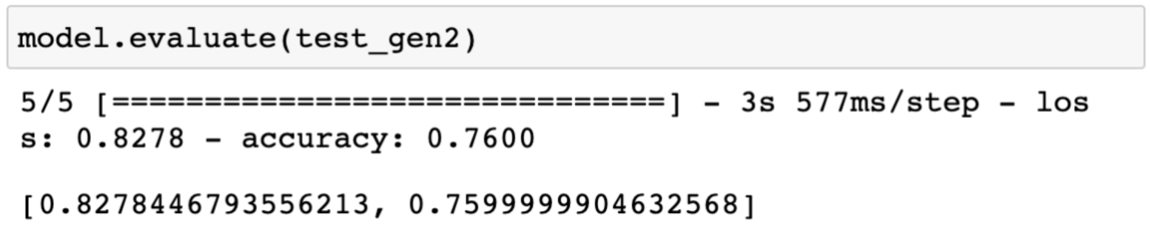


```
model.evaluate(test_gen)

1/1 [==============================] - 0s 265ms/step - los
s: 0.9380 - accuracy: 0.6000

[0.9379583597183228, 0.6000000238418579]
```

ResNet Model:

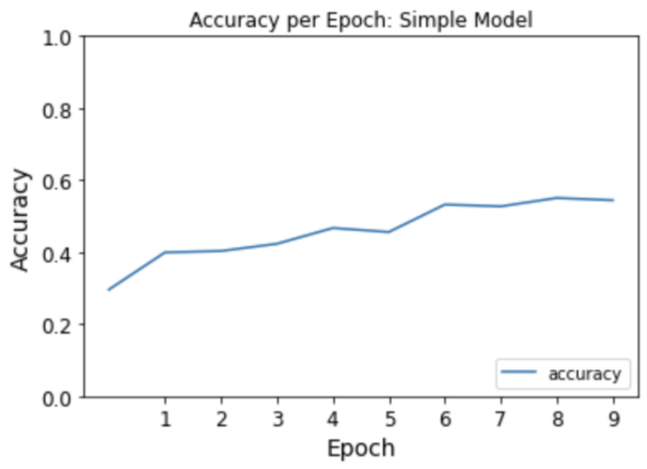


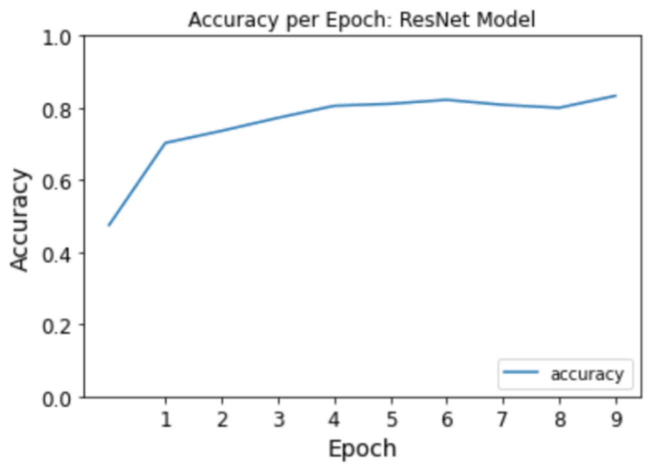
```
model.evaluate(test_gen2)

5/5 [==============================] - 3s 577ms/step - los
s: 0.8278 - accuracy: 0.7600

[0.8278446793556213, 0.7599999904632568]
```

**Training Performance:** The simple model's accuracy rate slowly increases while the ResNet model's accuracy has a steep increase in just one epoch and then it increase slowly.





## References

[1] Agarwal, S. (2021, April 18). Flower Classification. Kaggle. https://www.kaggle.com/sauravagarwal/flower-classification. Accessed 8 May 2021

[2] Géron Aurélien. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, And Tensorflow: Concepts, Tools, And Techniques. O'REILLY MEDIA.

[3] tf.keras.preprocessing.image.ImageDataGenerator. TensorFlow. (n.d.). https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator.