CNN application for classifying different types of flowers

Our main task is to train some models to be able to identify which type of flower is the right one and then determine which model is the best,

By using TensorFlow and karas libraries

- First step is to connect our data to google-colab then copy our data path

- Split the data to train and test (already received) and validate by using ImageDataGenerator (used for getting the input of the original data and further, it makes the transformation of this data on a random basis and gives the output resultant containing only the data that is newly transformed. It does not add the data) We will be able to rescale our image to make it easy for the machine to learn, Adjust the batch size (defines the number of samples that will be propagated through the network) for controlling the accuracy and ram usage issues

Train, Test, and Validation will be split with their own parameters with validation split equals to 0.1 which means 90% for train and 10% for validation

Then plotting some images and its labels to see how the machine reads the images and visualize some images from different classes

Simple Model:

Our **first model** started with a ridiculously small parameter in a Sequential model is (appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor)

then adding conviloution2d layers which applies sliding convolutional filters to 2-D input

- 16 filters with 3x3 kernel size ,1 stride, activation (ReLu) and input size as (150,150,3)
- Maxpooling
- Another conviloution2d layer with 32 filters with 3x3 kernel size and relu activation

- Maxpooling
- Another conviloution2d layer with 16 filters with 3x3 kernel size and relu activation
- Flatten layer
- Dense layer (256 output)
- Dense layer (1 output)

With Adam optimizer

loss SparseCategoricalCrossentropy

It didn't go quite well we realized we needed 104 dense at the end because we had 104 classes after all it gave us about 40% accuracy

We didn't use learning rate

After some updates we made **second model** with:

Learning rate = 0.00001

With Adam optimizer

loss SparseCategoricalCrossentropy

- 32 filters with 5x5 kernel size, activation (ReLu) and input size as (192,192,3) default size
- Maxpooling
- 64 filters with 3x3 kernel size, activation (ReLu) with padding
- Maxpooling
- 96 filters with 3x3 kernel size, activation (ReLu) with padding
- Maxpooling
- 96 filters with 3x3 kernel size, activation (ReLu) with padding
- Maxpooling
- flatting
- 512 Dense layers with activation (ReLu)
- 104 Dense layers with activation (SoftMax)

It gave us accuracy = 40%

Val_accuracy = 30%

Loss = 60%

Val_loss = 70%

Test loss 2.31

Test accuracy : 50.2%

And we tried **another one:**

Learning rate = 0.0001

With Adam optimizer

loss SparseCategoricalCrossentropy

- 200 filters with 3x3 kernel size, activation (ReLu) and input size as (175,175,3) default size
- 150 filters with 3x3 kernel size, activation (ReLu)
- Maxpooling(3x3)
- 120 filters with 3x3 kernel size, activation (ReLu) and input size as (175,175,3) default size
- 80 filters with 3x3 kernel size, activation (ReLu) and input size as (175,175,3) default size
- 50 filters with 3x3 kernel size, activation (ReLu) and input size as (175,175,3) default size
- Maxpooling(3x3)
- Flatten
- Dense (7000)
- Dense (4000)
- Dense (1800)
- Dense (800)
- Dense (104) with SoftMax

It gave us accuracy = 80%

Val_accuracy = 60%

Loss = 20%

Val_loss = 40%

Test accuracy = 50.5%

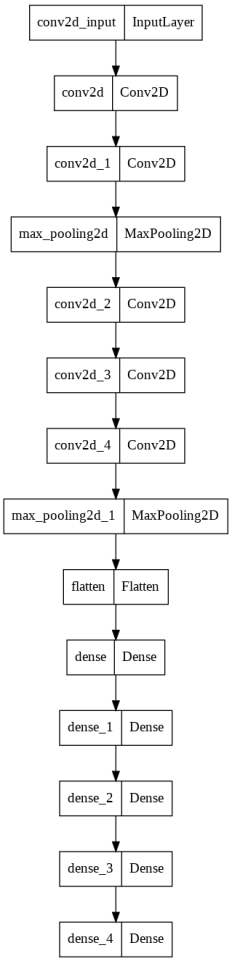
Test_loss = 2.3%

Fscore = 55%

Recall = 50%

Precession = 61%

Total parameters 138,447,304



VGG Model:

VGG is an innovative object-recognition model that supports up to 19 layers. Built as a deep CNN, VGG also outperforms baselines on many tasks and datasets outside of ImageNet. VGG is now still one of the most used image-recognition architectures

We scaled the images to (224,224,3)

Learning rate = 0.0001

- 64 filters with 3x3 kernel size, activation (ReLu)
- 64 filters with 3x3 kernel size, activation (ReLu)
- MaxPooling pool size 2 and stride with 2
- 128 filters with 3x3 kernel size, activation (ReLu)
- 128 filters with 3x3 kernel size, activation (ReLu)
- MaxPooling pool size 2 and stride with 2

- 256 filters with 3x3 kernel size, activation (ReLu)
- 256 filters with 3x3 kernel size, activation (ReLu)
- 256 filters with 3x3 kernel size, activation (ReLu)
- MaxPooling pool size 2 and stride with 2
- 512 filters with 3x3 kernel size, activation (ReLu)
- 512 filters with 3x3 kernel size, activation (ReLu)
- 512 filters with 3x3 kernel size, activation (ReLu)
- MaxPooling pool size 2 and stride with 2
- 512 filters with 3x3 kernel size, activation (ReLu)
- 512 filters with 3x3 kernel size, activation (ReLu)
- 512 filters with 3x3 kernel size, activation (ReLu)
- MaxPooling pool size 2 and stride with 2
- Flatten layer
- Dense (4096) activation (ReLu)
- Dense (4096) activation (ReLu)
- Dense (1000) SoftMax

accuracy = 80%

Val_accuracy = 70%

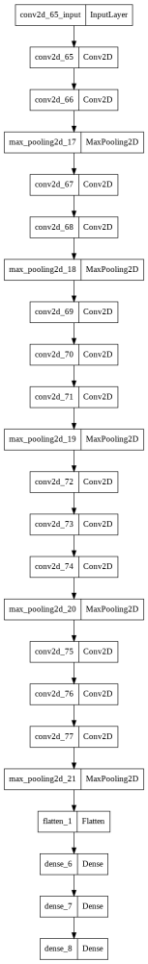
Loss = 20%

Val_loss = 30%

Test accuracy = 50.5%

Test Loss is 2.06

Total parameters 138 million



Google_Net Model:

Google Net is a convolutional neural network that is 22 layers deep. You can load a pretrained version of the network trained on either the ImageNet [1] or Places365 [2] [3] data sets. The network trained on ImageNet classifies images into 1000 object categories, such as keyboard, mouse, pencil, and many animals

We created a function called inception_block it takes a parameter filter it's the number of filters which itself returns sequential model

Google net takes the input shape as (224,224,3)

learning rate = 0.0001

- 64 filters with 1x1 kernel size, activation (ReLu)
- 192 filters with 3x3 kernel size, activation (ReLu)
- MaxPooling
- inception_block([64,96,128,16,32,32])

- inception_block([128,128,192,32,96,64])
- MaxPooling
- inception_block([192,96,208,16,48,64])
- inception_block([160,112,224,24,64,64])
- inception_block([128,128,256,24,64,64])
- inception_block([112,144,288,32,64,64])
- inception_block([256,160,320,32,128,128])
- MaxPooling
- inception_block([256,160,320,32,128,128])
- inception_block([384,192,384,48,128,128])
- AveragePooling2D(pool_size=7, strides=1)
- Dropout
- Dense with 1000 layer

Total parameters 4,388,632

Transfer Learning:

Transfer of learning means the use of previously acquired knowledge and skills in new learning or problem-solving situations. Thereby similarities and analogies between previous and actual learning content and processes may play a crucial role.

We used a pre trained model (vgg16) with imagenet weights then adding maxpooling and dense then compile

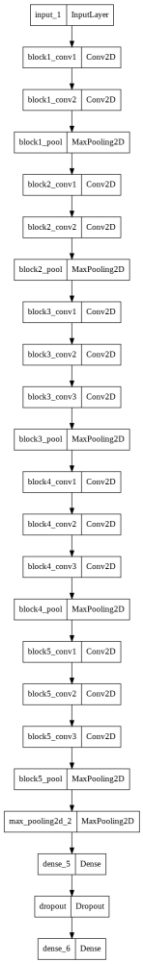
Plot Performance and Evaluation:

We used matplotlib to plot our loss and accuracy

Got the functions for F-score and precision and recall from sklearn

Then predict our test data then make our plotting

Number of parameters = 15,030,696



Residual neural network (ResNet):

A residual neural network (ResNet) is an artificial neural network (ANN). It is a gateless or open-gated variant of the HighwayNet, the first working very deep feedforward neural network with hundreds of layers, much deeper than previous neural network

We used pre-trained weights but we specified our input layer and output layer because the original resnet model may be trained on different layers, so we are making sure that it fits our data we gave it 180x180x3 input shape and used average pooling

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 2048)              23587712
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 2565
=================================================================
```

accuracy = 80%

Val_accuracy = 70%

Loss = 20%

Val_loss = 30%

Test accuracy = 50.5%

Test Loss is 2.06

Number of Parameters: 24,639,365

Trainable Parameters: 1,051,653

Ensemble Learning:

Ensemble learning techniques have been proven to yield better performance on machine learning problems. We can use these techniques for regression as well as classification problems. The final prediction from these ensembling techniques is obtained by combining results from several base models

We made a new model for the ensemble it will work as iteration model

We made two ensamples models one with new models (10 nets) second one with pretrained parameters and take the average ensemble

Adding 10 models together then train them individually and then mix the results into one model