# Model Building

| | |
|---|---|
| Team Id | PNT2022TMID07306 |
| Project Name | AI-powered Nutrition Analyzer for Fitness Enthusiasts |

## Importing The Model Building Libraries

```
### Importing Neccessary Libraries

import numpy as np#used for numerical analysis
import tensorflow #open source used for both ML and DL for computation
from tensorflow.keras.models import Sequential #it is a plain stack of layers
from tensorflow.keras import layers #A layer consists of a tensor-in tensor-out computation function
#Dense layer is the regular deeply connected neural network layer
from tensorflow.keras.layers import Dense,Flatten
#Faltten-used fot flattening the input or change the dimension
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Dropout #Convolutional layer
#MaxPooling2D-for downsampling the image
from keras.preprocessing.image import ImageDataGenerator
```

## Initializing The Model

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method.

```
model=Sequential()
```

## Adding CNN Layers

- We are adding a two convolution layer with activation function as “relu” and with a small filter size (3,3) and the number of filters (32) followed by a max-pooling layer.
- Max pool layer is used to down sample the input. (Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter)
- Flatten layer flattens the input. Does not affect the batch size
-

```
In [13]: pretrained_model = tf.keras.applications.MobileNetV2(
             input_shape=(224, 224, 3),
             include_top=False,
             weights='imagenet',
             pooling='avg'
         )
         pretrained_model.trainable = False
```

## Adding Dense Layers

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.

Understanding the model is a very important phase to properly using it for training and prediction purposes. Keras provides a simple method, a summary to get the full information about the model and its layers.

```
In [*]: inputs = pretrained_model.input

        x = tf.keras.layers.Dense(128, activation='relu')(pretrained_model.output)
        x = tf.keras.layers.Dense(128, activation='relu')(x)

        outputs = tf.keras.layers.Dense(36, activation='softmax')(x)

        model = tf.keras.Model(inputs=inputs, outputs=outputs)

        model.compile(
            optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )

        history = model.fit(
            train_images,
            validation_data=val_images,
            batch_size = 32,
            epochs=5,
            callbacks=[
                tf.keras.callbacks.EarlyStopping(
                    monitor='val_loss',
                    patience=2,
                    restore_best_weights=True
                )
            ]
        )
```

## Configure The Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

## Train The Model

Now, let us train our model with our image dataset. The model is trained for 20 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 20 epochs and probably there is further scope to improve the model.

### fit_generator functions used to train a deep learning neural network Arguments:

- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

- Epochs: an integer and number of epochs we want to train our model for.

- validation_data can be either:
  - an inputs and targets list
  - a generator
  - inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.
- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

```
## Fitting the model

classifier.fit_generator(
        generator=x_train,steps_per_epoch = len(x_train),
        epochs=20, validation_data=x_test,validation_steps = len(x_test))# No of images in test set
```

## Save The Model

```
In [53]: model.save('Fruit_Recognition.h5')
```

## Test The Model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.
Load the saved model using load_model

```
In [ ]: # Predict the label of the test_images
pred = model.predict(test_images)
pred = np.argmax(pred,axis=1)
# Map the label
labels = (train_images.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred1 = [labels[k] for k in pred]
pred1
```

```
In [ ]: def output(location):
    img=load_img(location,target_size=(224,224,3))
    img=img_to_array(img)
    img=img/255
    img=np.expand_dims(img,[0])
    answer=model.predict(img)
    y_class = answer.argmax(axis=-1)
    y = " ".join(str(x) for x in y_class)
    y = int(y)
    res = labels[y]
    return res
```

```
In [52]: img = output('dataset\\test\\soy beans\\Image_1.jpg')
img
```

```
1/1 [==============================] - 1s 1s/step
```

```
Out[52]: 'soy beans'
```