Train And Save The Model

Compile the model

After adding all the required layers, the model is to be compiled. For this step, loss function, optimizer and metrics for evaluation can be passed as arguments.

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

Fit and save the model

Fit the neural network model with the train and test set, number of epochs and validation steps. Steps per epoch is determined by number of training images//batch size, for validation steps number of validation images//batch size.

```
model.fit(x_train,epochs=10,steps_per_epoch=89,validation_data = x_test, validation_steps = 27)
```

Accuracy, Loss: Loss value implies how poorly or well a model behaves after each iteration of optimization. An accuracy metric is used to measure the algorithm's performance in an interpretable way. The accuracy of a model is usually determined after the model parameters and is calculated in the form of a percentage.

The weights are to be saved for future use. The weights are saved in as .h5 file using save().

```
model.save("fruit.h5")
```

model.summary() can be used to see all parameters and shapes in each layer in our models.

Train And Save The Model

In [13]:
```
model.compile(optimizer='adam', loss = "categorical_crossentropy" , metrics =['accuracy'])
```

In [15]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2D  (None, 63, 63, 32)       0
 )

 flatten (Flatten)           (None, 127008)            0

 dense (Dense)               (None, 40)                5080360

 dense_1 (Dense)             (None, 20)                820

 dense_2 (Dense)             (None, 6)                 126

=================================================================
Total params: 5,082,202
Trainable params: 5,082,202
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```
model.fit(x_train,epochs=10,steps_per_epoch=89,validation_data = x_test, validation_steps = 27)


model.save("fruit.h5")
```

```
Epoch 1/10
89/89 [==============================] - 945s 11s/step - loss: 1.1761 - accuracy: 0.6246 - val_loss: 66.9958 - val_accuracy: 0.7940
Epoch 2/10
89/89 [==============================] - 477s 5s/step - loss: 0.5927 - accuracy: 0.8090 - val_loss: 129.4430 - val_accuracy: 0.6516
Epoch 3/10
89/89 [==============================] - 234s 3s/step - loss: 0.4787 - accuracy: 0.8441 - val_loss: 227.8628 - val_accuracy: 0.5243
Epoch 4/10
89/89 [==============================] - 122s 1s/step - loss: 0.3456 - accuracy: 0.8835 - val_loss: 233.2232 - val_accuracy: 0.5359
Epoch 5/10
89/89 [==============================] - 85s 959ms/step - loss: 0.2847 - accuracy: 0.9040 - val_loss: 633.7368 - val_accuracy: 0.3704
Epoch 6/10
89/89 [==============================] - 68s 767ms/step - loss: 0.2261 - accuracy: 0.9235 - val_loss: 681.6103 - val_accuracy: 0.3993
Epoch 7/10
89/89 [==============================] - 59s 663ms/step - loss: 0.2459 - accuracy: 0.9125 - val_loss: 233.5868 - val_accuracy: 0.6343
Epoch 8/10
89/89 [==============================] - 52s 587ms/step - loss: 0.2116 - accuracy: 0.9245 - val_loss: 600.8589 - val_accuracy: 0.4167
Epoch 9/10
89/89 [==============================] - 51s 572ms/step - loss: 0.1742 - accuracy: 0.9431 - val_loss: 729.3225 - val_accuracy: 0.4167
Epoch 10/10
89/89 [==============================] - 52s 587ms/step - loss: 0.1638 - accuracy: 0.9437 - val_loss: 778.6277 - val_accuracy: 0.3681
```