#IMPORTING DATASET

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

#IMPORTING LIBRARIES

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

#DATA PREPROCESSING

###Training Image Preprocessing

```
training_set= tf.keras.utils.image_dataset_from_directory(
    '/content/drive/MyDrive/Fruits_Vegetable_Recognition/train',
    labels = 'inferred',
    label_mode = 'categorical',
    class_names = None,
    color_mode = 'rgb',
    batch_size = 32,
    image_size = (64,64),
    shuffle = True,
    seed = None,
    validation_split = None,
    subset = None,
    interpolation = 'bilinear',
    follow_links = False,
    crop_to_aspect_ratio = False
)
```

```
Found 3114 files belonging to 36 classes.
```

###Validation Image preprocessing

```
validation_set = tf.keras.utils.image_dataset_from_directory(
    '/content/drive/MyDrive/Fruits_Vegetable_Recognition/validation',
    labels = 'inferred',
    label_mode = 'categorical',
    class_names = None,
    color_mode = 'rgb',
    batch_size = 32,
    image_size = (64,64),
    shuffle = True,
    seed = None,
```

```
    validation_split = None,
    subset = None,
    interpolation = 'bilinear',
    follow_links = False,
    crop_to_aspect_ratio = False
)
```

```
Found 351 files belonging to 36 classes.
```

#BUILDING MODEL

```
cnn = tf.keras.models.Sequential()
```

###Building Convolution Layer

```
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,activation='re
lu',input_shape=[64,64,3]))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/
convolutional/base_conv.py:113: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
#To Reduce the size of the layer to focus on the important feature
running it twice
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,activation='re
lu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))

cnn.add(tf.keras.layers.Dropout(0.5)) # To avoid Overfitting

cnn.add(tf.keras.layers.Flatten())

cnn.add(tf.keras.layers.Dense(units=128,activation='relu'))

cnn.add(tf.keras.layers.Dense(units=36,activation='softmax')) #Output
layer
```

#COMPILING AND TRAINING PHASE

```
cnn.compile(optimizer='rmsprop',loss='categorical_crossentropy',metric
s=['accuracy'])

training_history=cnn.fit(x=training_set,validation_data=validation_set
,epochs=30)
```

```
Epoch 1/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 505s 5s/step - accuracy: 0.0506 - loss: 21.4685 - val_accuracy: 0.0826 - val_loss: 3.4854
Epoch 2/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 170s 1s/step - accuracy: 0.0612 - loss: 3.8367 - val_accuracy: 0.1368 - val_loss: 3.4252
Epoch 3/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 145s 1s/step - accuracy: 0.0789 - loss: 3.8087 - val_accuracy: 0.1880 - val_loss: 3.1744
Epoch 4/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.1316 - loss: 3.9875 - val_accuracy: 0.2422 - val_loss: 3.0347
Epoch 5/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 152s 1s/step - accuracy: 0.1824 - loss: 4.0521 - val_accuracy: 0.2991 - val_loss: 3.3456
Epoch 6/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 112s 1s/step - accuracy: 0.2257 - loss: 4.2061 - val_accuracy: 0.1538 - val_loss: 2.9527
Epoch 7/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 112s 1s/step - accuracy: 0.2478 - loss: 3.1876 - val_accuracy: 0.4929 - val_loss: 2.1993
Epoch 8/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 138s 1s/step - accuracy: 0.3586 - loss: 3.5720 - val_accuracy: 0.1595 - val_loss: 10.8230
Epoch 9/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.3779 - loss: 3.3457 - val_accuracy: 0.2593 - val_loss: 6.7759
Epoch 10/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 113s 1s/step - accuracy: 0.4432 - loss: 2.6413 - val_accuracy: 0.4843 - val_loss: 2.2517
Epoch 11/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.4887 - loss: 2.3002 - val_accuracy: 0.6838 - val_loss: 1.5681
Epoch 12/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 113s 1s/step - accuracy: 0.5425 - loss: 2.0595 - val_accuracy: 0.6182 - val_loss: 1.8376
Epoch 13/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 150s 1s/step - accuracy: 0.5765 - loss: 2.0362 - val_accuracy: 0.3875 - val_loss: 5.9175
Epoch 14/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 126s 1s/step - accuracy: 0.5797 - loss: 2.9642 - val_accuracy: 0.7977 - val_loss: 1.1512
Epoch 15/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 104s 1s/step - accuracy: 0.6332 - loss: 1.7093 - val_accuracy: 0.5442 - val_loss: 5.1295
Epoch 16/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 152s 1s/step - accuracy: 0.6639 - loss: 2.0076 - val_accuracy: 0.8091 - val_loss: 1.2323
Epoch 17/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 104s 1s/step - accuracy: 0.6833 - loss:
```

```
1.6138 - val_accuracy: 0.8490 - val_loss: 1.1664
Epoch 18/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 111s 1s/step - accuracy: 0.7453 - loss:
1.1329 - val_accuracy: 0.6553 - val_loss: 3.4590
Epoch 19/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 114s 1s/step - accuracy: 0.7022 - loss:
1.8725 - val_accuracy: 0.8746 - val_loss: 1.2475
Epoch 20/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 140s 1s/step - accuracy: 0.7597 - loss:
1.3762 - val_accuracy: 0.8319 - val_loss: 1.5897
Epoch 21/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 136s 1s/step - accuracy: 0.7144 - loss:
1.9724 - val_accuracy: 0.7806 - val_loss: 1.9912
Epoch 22/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 138s 1s/step - accuracy: 0.7686 - loss:
1.1720 - val_accuracy: 0.8718 - val_loss: 1.1783
Epoch 23/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 124s 1s/step - accuracy: 0.7998 - loss:
1.0449 - val_accuracy: 0.9060 - val_loss: 1.0870
Epoch 24/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 123s 1s/step - accuracy: 0.7760 - loss:
1.3447 - val_accuracy: 0.8889 - val_loss: 1.1897
Epoch 25/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 150s 1s/step - accuracy: 0.8173 - loss:
1.0037 - val_accuracy: 0.8661 - val_loss: 1.6119
Epoch 26/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 114s 1s/step - accuracy: 0.8072 - loss:
1.2024 - val_accuracy: 0.9345 - val_loss: 1.2440
Epoch 27/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 102s 1s/step - accuracy: 0.8248 - loss:
1.0429 - val_accuracy: 0.6980 - val_loss: 3.1603
Epoch 28/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 119s 1s/step - accuracy: 0.7963 - loss:
1.1025 - val_accuracy: 0.9316 - val_loss: 0.9205
Epoch 29/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 135s 1s/step - accuracy: 0.8227 - loss:
1.0825 - val_accuracy: 0.8889 - val_loss: 1.6452
Epoch 30/30
98/98 ━━━━━━━━━━━━━━━━━━━━ 111s 1s/step - accuracy: 0.8398 - loss:
0.9519 - val_accuracy: 0.8575 - val_loss: 1.6696
```

#SAVING MODEL

```
cnn.save('trained_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
```

```
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
```

```python
training_history.history # Return Dictionary of history
```

```
{'accuracy': [0.04367373138666153,
  0.0581245981156826,
  0.08413615822792053,
  0.1364804059267044,
  0.1917148381471634,
  0.201991006731987,
  0.2649325728416443,
  0.35709697008132935,
  0.3856775760650635,
  0.45407834649086,
  0.4900449514389038,
  0.5407835841178894,
  0.5828516483306885,
  0.614964663982391 4,
  0.6425818800926208,
  0.6769428253173828,
  0.6833654642105103,
  0.7283236980438232,
  0.7145150899887085,
  0.7594733238220215,
  0.752729594707489,
  0.7668593525886536,
  0.7854849100112915,
  0.7825947403907776,
  0.8086062669754028,
  0.8127809762954712,
  0.8169556856155396,
  0.7999357581138611,
  0.8195247054100037,
  0.8355812430381775],
 'loss': [7.991447925567627,
  3.788217544555664,
  3.7795357704162598,
  4.593155384063721,
  4.134185314178467,
  5.616292476654053,
  3.209615707397461,
  3.483926773071289,
  2.961479663848877,
  2.5358078479766846,
  2.3468966484069824,
  2.0783424377441406,
  1.8963251113891602,
  2.1765940189361572,
  1.6501083374023438,
```

```
  1.597287654876709,
  1.8596857786178589,
  1.275895118713379,
  1.8813726902008057,
  1.283402919769287,
  1.435881495475769,
  1.2203389406204224,
  1.2710164785385132,
  1.3570436239242554,
  1.1068068742752075,
  1.1995667219161987,
  1.0726715326309204,
  1.1295652389526367,
  1.0840001106262207,
  0.9491913318634033],
 'val_accuracy': [0.08262108266353607,
  0.1367521435022354,
  0.18803419172763824,
  0.2421652376651764,
  0.29914531111717224,
  0.1538461595773697,
  0.4928774833679199,
  0.15954415500164032,
  0.25925925374031067,
  0.4843304753303528,
  0.6837607026100159,
  0.6182336211204529,
  0.38746437430381775,
  0.7977207899093628,
  0.5441595315933228,
  0.809116780757904,
  0.8490028381347656,
  0.6552706360816956,
  0.874643862247467,
  0.8319088220596313,
  0.7806267738342285,
  0.8717948794364929,
  0.9059829115867615,
  0.8888888955116272,
  0.8660968542098999,
  0.934472918510437,
  0.6980056762695312,
  0.9316239356994629,
  0.8888888955116272,
  0.8575498461723328],
 'val_loss': [3.4853904247283936,
  3.4251604080200195,
  3.1744134426116943,
  3.034712791442871,
```

```
  3.345641851425171,
  2.9527382850646973,
  2.1992554664611816,
  10.823009490966797,
  6.775928974151611,
  2.25167131423950 2,
  1.5680787563323975,
  1.8376468420028687,
  5.917484760284424,
  1.1512103080749512,
  5.1295247077941895,
  1.2322850227355957,
  1.1663790941238403,
  3.458970069885254,
  1.2474936246871948,
  1.5897057056427002,
  1.9911737442016602,
  1.1783422231674194,
  1.0869905948638916,
  1.189653754234314,
  1.6118786334991455,
  1.2439942359924316,
  3.1602985858917236,
  0.9204903244972229,
  1.6451780796051025,
  1.6695502996444702]}
```

```
# RECORDING HISTORY WITH JSON
import json
with open('training_hist.json','w')as f:
  json.dump(training_history.history,f)

print(training_history.history.keys())

dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

#CALCULATING ACCURACY OF MODEL ACHIEVED ON VALIDATION SET

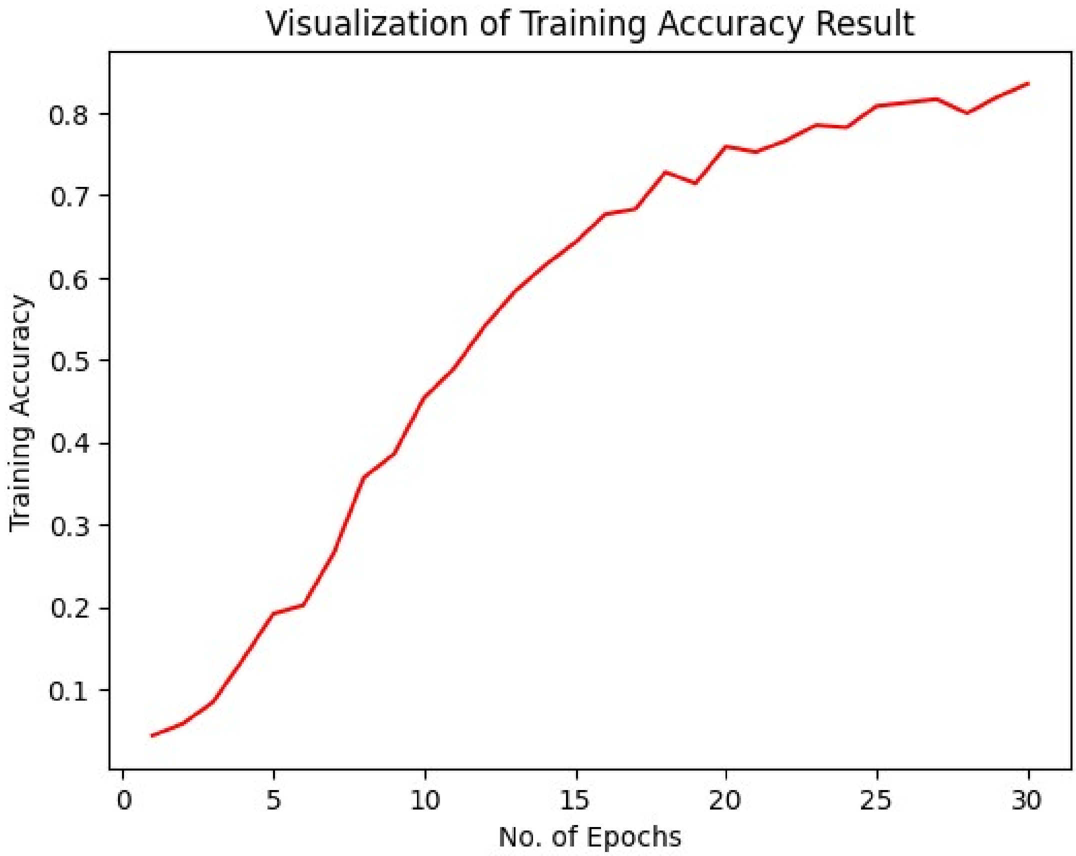
```
print("Validation set accuracy:
{}".format(training_history.history['val_accuracy'][-1]*100))

Validation set accuracy: 85.75498461723328
```

#ACCURACY VISUALIZATION

###Training Visualization

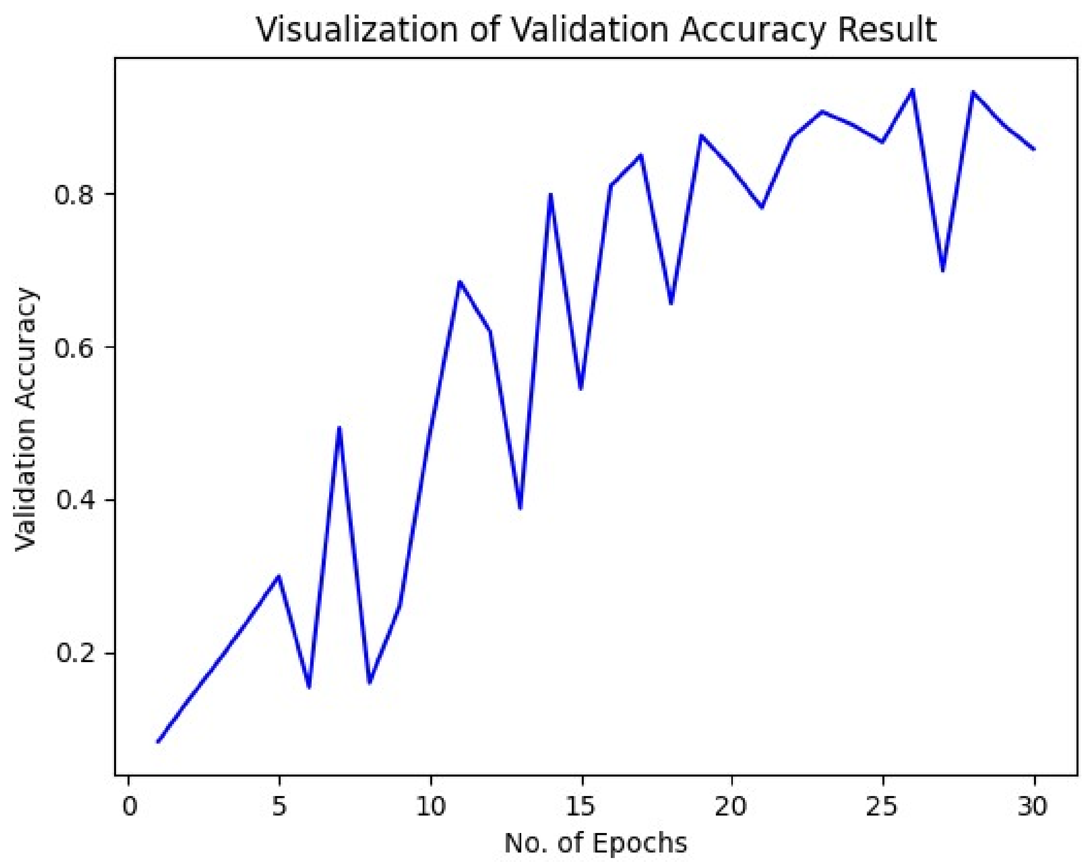
```
epochs = [i for i in range(1,31)]
plt.plot(epochs,training_history.history['accuracy'],color = 'red')
plt.xlabel('No. of Epochs')
```

```
plt.ylabel('Training Accuracy')
plt.title('Visualization of Training Accuracy Result')
plt.show()
```



###Validation Accuracy

```
plt.plot(epochs,training_history.history['val_accuracy'],color='blue')
plt.xlabel("No. of Epochs")
plt.ylabel('Validation Accuracy')
plt.title('Visualization of Validation Accuracy Result')
plt.show()
```



Visualization of Validation Accuracy Result
Validation Accuracy
0.2
0.4
0.6
0.8
0
5
10
15
20
25
30
No. of Epochs