Food Calorie Predictor

Submitted to

SANKALP
Ministry of Skill Development
& Entrepreneurship

FITT

MPSSDEGB
कौशल से रोजगार, समृद्धि अपार

[Your Name]

[Your Affiliation/Institution]

[Date]

Abstract

In the contemporary era of heightened health consciousness, the Food Calorie Predictor project aims to address the escalating interest in nutritional awareness by developing a sophisticated tool for predicting calorie content in various foods. This abstract provides a comprehensive overview of the project's objectives, methodology, and potential impact.

The project's genesis lies in the recognition of the significance of nutritional tracking in maintaining a healthy lifestyle. The introduction outlines the primary objectives, including the creation of a user-friendly tool and the provision of a convenient solution for individuals seeking to monitor their calorie consumption. Leveraging Python programming language, TensorFlow, and Scikit-learn, the project proposes a machine learning-based approach for accurate calorie predictions. The web-based interface, developed using Flask, ensures seamless user interaction.

A thorough literature survey explores existing calorie prediction solutions and studies on nutritional awareness, providing essential insights into the methodologies and approaches adopted in the field. By incorporating the best practices and lessons learned from the existing body of knowledge, the project aims to contribute to the advancement of effective calorie prediction solutions.

The project planning and management section delineates a structured timeline, breaking down the project into phases such as data collection, model development, and user interface implementation. The modular design approach enhances organization and efficiency throughout the project's execution. Specific tasks, including data preprocessing and model training, are allocated to defined timeframes, ensuring a systematic and well-managed development process.

Methodologically, the project focuses on data preprocessing and model training. Image resizing and data augmentation techniques are employed to optimize the dataset, while convolutional and recurrent neural networks are utilized for effective feature extraction and sequence modeling. The methodology underscores a meticulous approach to developing a robust and accurate Food Calorie Predictor.

The system design section provides insights into the architectural aspects of the predictor, detailing the integration of the trained model with the user interface. Coding and testing considerations encompass the use of Python, TensorFlow, Scikit-learn, and Flask, with a comprehensive testing strategy ensuring the reliability of the application.

Results and discussions showcase the accuracy of the Food Calorie Predictor, with user feedback providing valuable insights for potential improvements. The conclusion summarizes key findings, emphasizing the project's significance in promoting informed dietary choices. Future enhancements, such as dataset expansion and real-time image recognition, are discussed, charting a path for continued refinement and innovation in the realm of nutritional tracking. The project, rooted in a commitment to healthier living, contributes to the evolving landscape of technology-driven solutions for personal well-being.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

The purpose of this project is to take food images as input, process the input, training the model, to recognise the food item. The second step is to estimate the amount of calories that a user would gain on consuming the food item. This project is restricted to globally recognisable food. It can be further developed to recognise local food items. It further engulfs around the concept of CNN.

1.2 SCOPE OF THE PROJECT

Our project proposes on recognising/detecting food items in a food image and show its calorie value by using convolutional neural network(popular for image recognition). will train our model to recognise food items, then with the help of support vector machines(SVM), classify those food items into different categories(e.g burger,pizza etc)

1.3 Software Requirements Specification

PRODUCT FUNCTIONS

- To provide an easy interface to input the object image
- User would be able to upload the image
- System would be able to pre-process the given input to align it
- System would be able to detect food items present in the image
- System would be able to estimate calories

USER CLASSES

Naive users: Key factor in uploading the images which serves the secondary resource for the data set, in order to improve a model.

Analyst: Studies the problem and works on the data collected. decides the best suitable algorithm to be applied etc.

USER INTERFACES

An Android App/website for the interaction with the user. Input of food items is given from a picture taken through the smartphone. Once the image is uploaded, it is categorized to different food categories with the help of image processing. And when compared with a trained dataset it will display the calorie count of the food items to the user as output.

HARDWARE INTERFACES

Mobile Phone which is running on Android version 6.0 or higher with a camera is the hardware used by the user to give input. Backend management of processing is done by systems present on the server side which have graphical processors to preprocess, categorize and detect the food item and calculate the amount of calories.

SORFWARE INTERFACES

- Tensorflow API
- GoogLeNet Inception V3

CHAPTER 2

LITERATURE REVIEW

[1] Issues in dietary intake assessment of children and adoloscents

There has been a number of proposed methods for measuring daily food diet information. In this, one existing system which asks the user to give the details of food and drinks he/she had consumed in 24 hours to the instructor or dietitian but the problem with this type of method is sometimes people won’t be able to remember exactly what they ate with content and amount. It is hard for the user to explain and give details of everything he/she consumed in the last 24- hours.

[2] Determination of food portion size by image processing

Researchers trying to improve on this technique and in the paper [2] author uses a new idea in which the user takes a picture of the food before eating and using a calorie card as a reference, it tells the calorie value of the food. The card should be placed next to the food while capturing the picture . Drawback of this system is that it will not work without the card.There is another system which is based on support vector machine but use the thumb for calibration of each and every food image, it requires long calculation for measuring nutrition of the food photo taken with the camera of a mobile phone, but the use of thumb of patient for calibration, solves the problem of carrying cards or special trays. More specifically, an thumb image is captured and stored with its measurements in the first usage time (first time calibration).

[3] Self-monitoring dietary intake: Current andfuture practices

In this, another method had been proposed by the author where a user have the PDA(personal diet assistant) app. In which the user record the daily food intake information on a mobile phone . but it has been shown that result of the portion has significant error and take long time for the user to enter the record.

[4] Healthaware:

Tackling obesity with health aware smart phone systems Yet another approach appeared in [4] where user can take a picture of the food from a smartphone and its compared with the predefined similar picture with it’s known nutrition value stored in the database.The main disadvantage of this system is that it does not consider the size of the food, which is extremely important.

[5]Food recognition using statistics of pairwise local features

Food recognition is a difficult task since appearance of the food are various even they belongs to the same category. In [6] the author proposed a method for recognition multiple images which first detect food put region by several detector next recognize by extracted color, texture, gradient and SIFT using multiple kernel learning.

[6] Recognition of multiple-food images by detecting candidate regions

The TADA dietary assessment system [6] has food identification and quantity estimation, although it has some restriction that food must be put on white dishes and food photos must be taken with a checkerboard to food quantity estimation.

CHAPTER 3

PROJECT PLANNING AND MANAGEMENT

Planning and managing a food calorie prediction project requires careful consideration of various aspects, including team organization, task allocation, timelines, and resources.

Project Initiation: Define the project goals and objectives clearly. Determine the scope of the project, including the types of food items to be included and the level of accuracy expected in calorie predictions.Identify key stakeholders and establish communication channels to ensure alignment throughout the project

Team Formation: Assemble a multidisciplinary team with expertise in machine learning, data science, nutrition, software development, and project management.Assign roles and responsibilities to team members based on their skills and expertise. This may include roles such as data scientists, software engineers, domain experts, and project managers.

Project Planning: Develop a project plan outlining the tasks, dependencies, timelines, and milestones for each phase of the project.Use project management tools such as Gantt charts, Kanban boards, or agile methodologies to organize and track progress.Break down the project into manageable sprints or iterations, with regular checkpoints for review and feedback.

Data Acquisition and Preparation: Identify and acquire relevant datasets containing information about food items and their nutritional content.Clean and preprocess the data to handle missing values, outliers, and inconsistencies. This may involve data cleaning, normalization, feature engineering, and data augmentation techniques.

Model Development and Evaluation: Develop machine learning models for predicting food calories based on the prepared data.Train the models using appropriate algorithms and techniques, such as regression, ensemble methods, or deep learning.Evaluate the models using cross-validation techniques and performance metrics such as mean absolute error, mean squared error, or accuracy

Monitoring and Maintenance: Implement monitoring mechanisms to track the performance of the deployed system and collect user feedback.Continuously monitor and optimize the models based on feedback and changes in data or user requirements.Provide ongoing support and maintenance to address any issues or updates required for the system.

Implement monitoring mechanisms to track the performance of the deployed system and collect user feedback.

Monitoring and Maintenance: Continuously monitor and optimize the models based on feedback and changes in data or user requirements.Provide ongoing support and maintenance to address any issues or updates required for the system.

Documentation and Knowledge Sharing: Document the entire project, including data sources, methodologies, algorithms, and implementation details.Share knowledge and insights gained from the project with relevant stakeholders through reports, presentations, or documentation.Foster a culture of knowledge sharing and collaboration within the team to leverage learnings for future projects.

CHAPTER 4

METHODOLOGY

Data Collection: Gather a diverse and comprehensive dataset of food items along with their corresponding nutritional information, especially calories. This data can be sourced from reputable databases, nutritional labels, or crowdsourced information.

Data Preprocessing : Clean and preprocess the data to handle missing values, outliers, and inconsistencies. Standardize units and formats to ensure consistency across the dataset. This step may also involve categorizing foods into groups or classes for better prediction.

Feature Engineering: Identify relevant features that contribute to predicting calorie content. Features might include serving size, macronutrient composition (carbohydrates, proteins, fats), and micronutrients. Additionally, consider incorporating information about cooking methods, food preparation, and recipe details.

Model Selection: Choose an appropriate machine learning model for regression, as predicting calorie content is essentially a regression problem. Common models include linear regression, decision trees, random forests, and neural networks. The choice of model may depend on the complexity of the problem and the amount of data available.

Training the Model: Split the dataset into training and testing sets to evaluate the model's performance. Train the chosen model on the training set and adjust parameters to optimize performance. Use cross-validation techniques to ensure robustness and prevent overfitting.

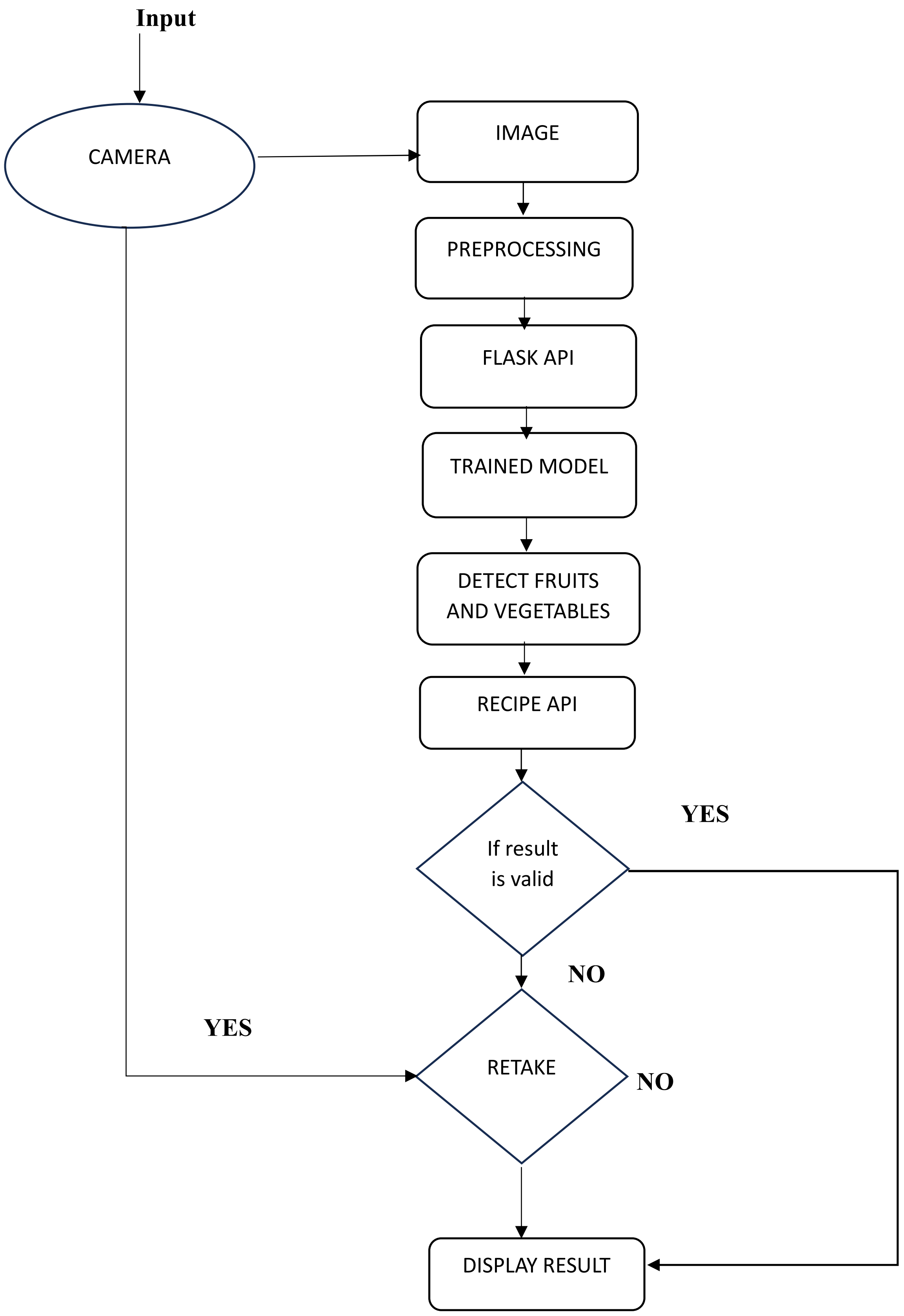
Model Interpretability: Depending on the complexity of the chosen model, consider methods for interpreting the results. This is especially important for understanding which features contribute the most to the calorie predictions.

User Interface (UI) Development: If the food calorie predictor is intended for user interaction, design a user-friendly interface. This could be a mobile app, web application, or any other platform where users can input information about the food they are consuming and receive calorie predictions.

Deployment: Deploy the model and the user interface in a production environment. Ensure that the system is scalable and can handle multiple requests. Integrate any necessary security measures, especially if handling user data.

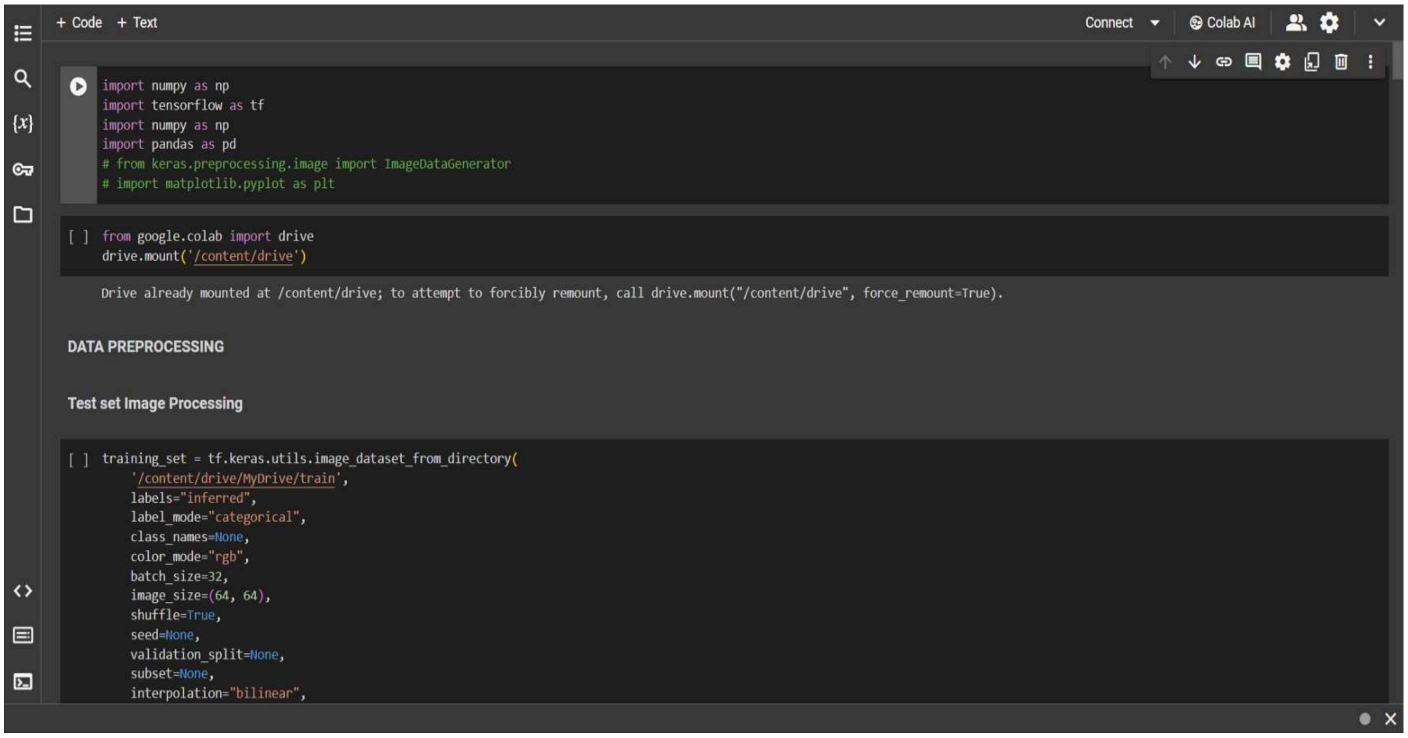
CHAPTER 5

SYSTEM DESIGN



CHAPTER 6

CODING AND TESTING



```
import numpy as np
import tensorflow as tf
import numpy as np
import pandas as pd
# from keras.preprocessing.image import ImageDataGenerator
# import matplotlib.pyplot as plt
```

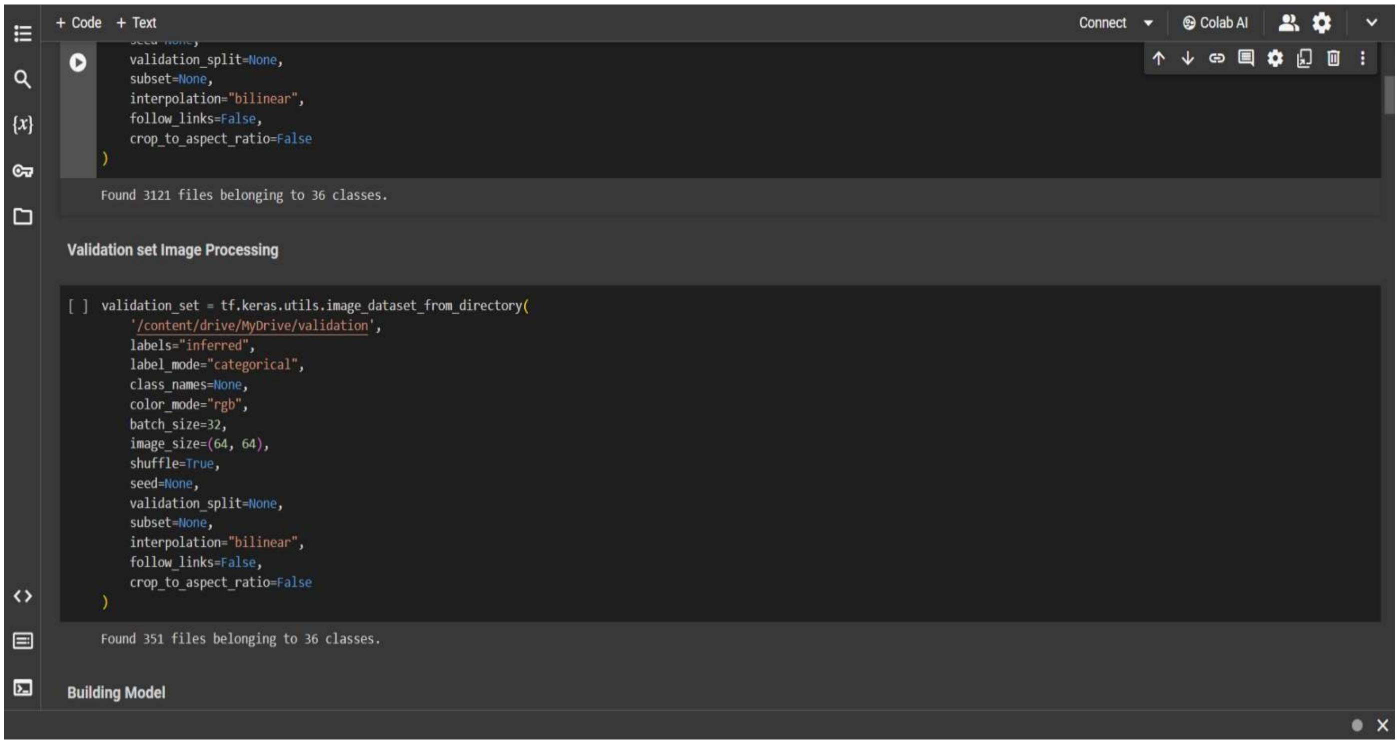
```
from google.colab import drive
drive.mount('/content/drive')
```

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

DATA PREPROCESSING

Test set Image Processing

```
training_set = tf.keras.utils.image_dataset_from_directory(
    '/content/drive/MyDrive/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",
```



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

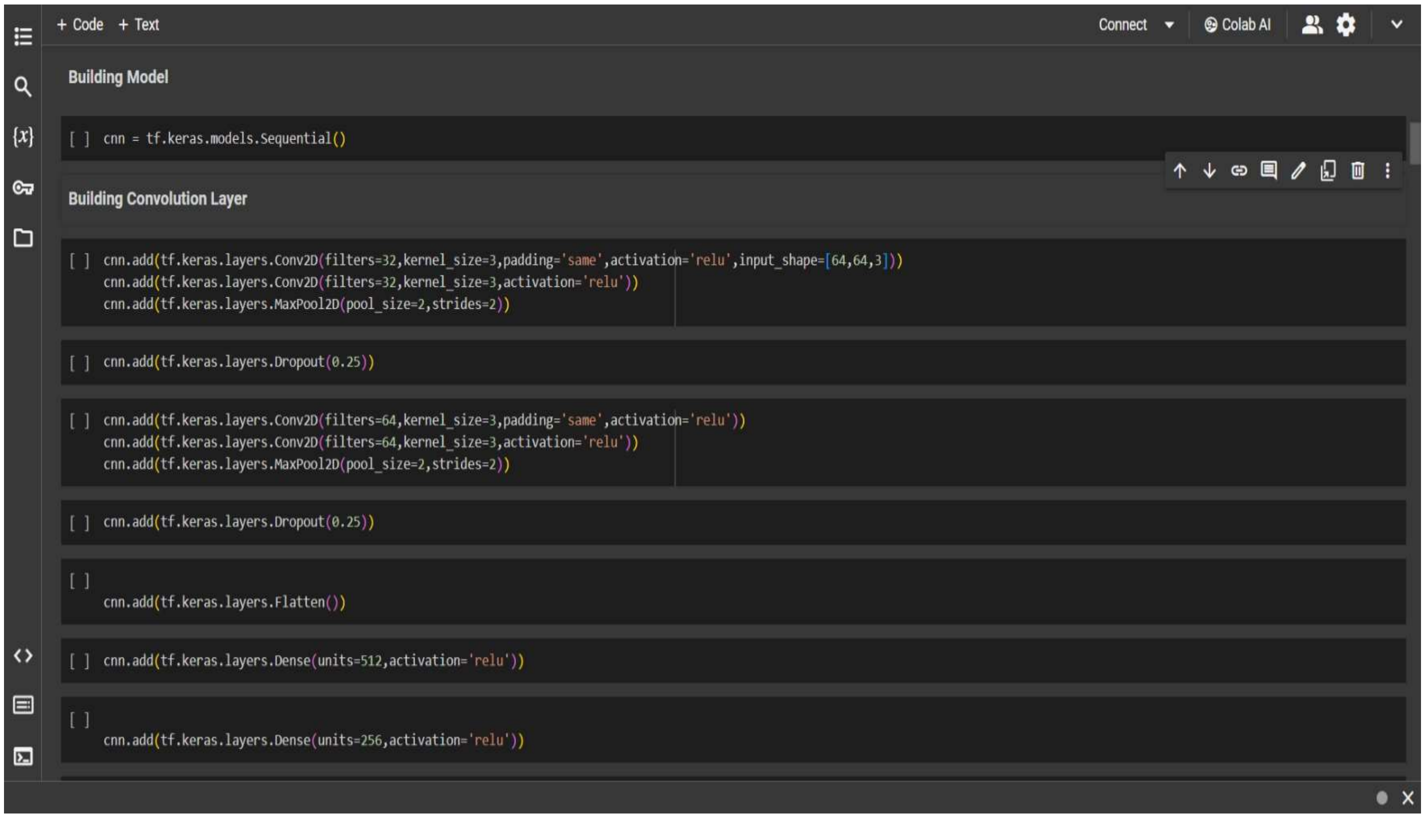
```
Found 3121 files belonging to 36 classes.
```

Validation set Image Processing

```
validation_set = tf.keras.utils.image_dataset_from_directory(
    '/content/drive/MyDrive/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



Building Model

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

Building Convolution Layer

```
cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding='same',activation='relu',input_shape=[64,64,3]))
cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Dropout(0.25))
```

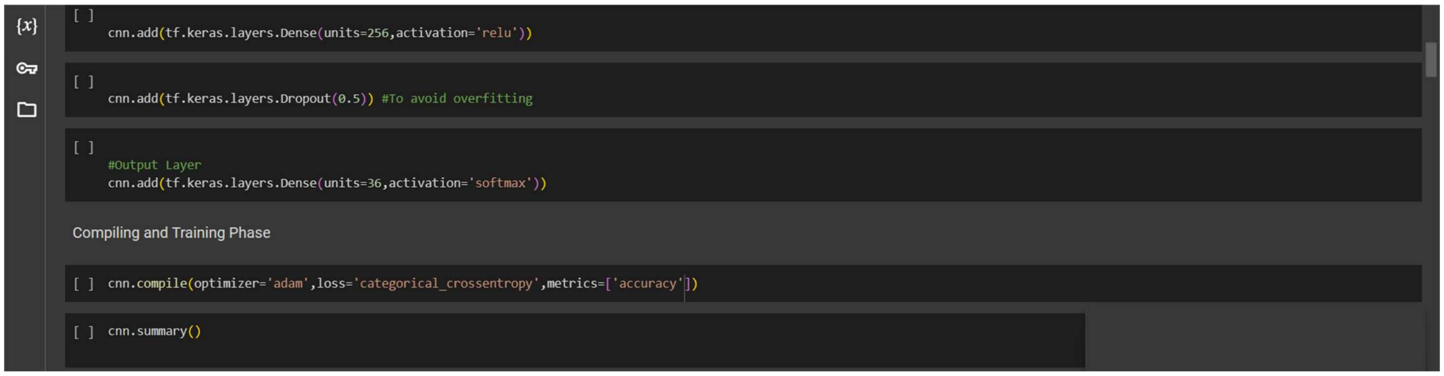
```
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Dropout(0.25))
```

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

```
cnn.add(tf.keras.layers.Dense(units=512,activation='relu'))
```

```
cnn.add(tf.keras.layers.Dense(units=256,activation='relu'))
```



```
cnn.add(tf.keras.layers.Dense(units=256,activation='relu'))
```

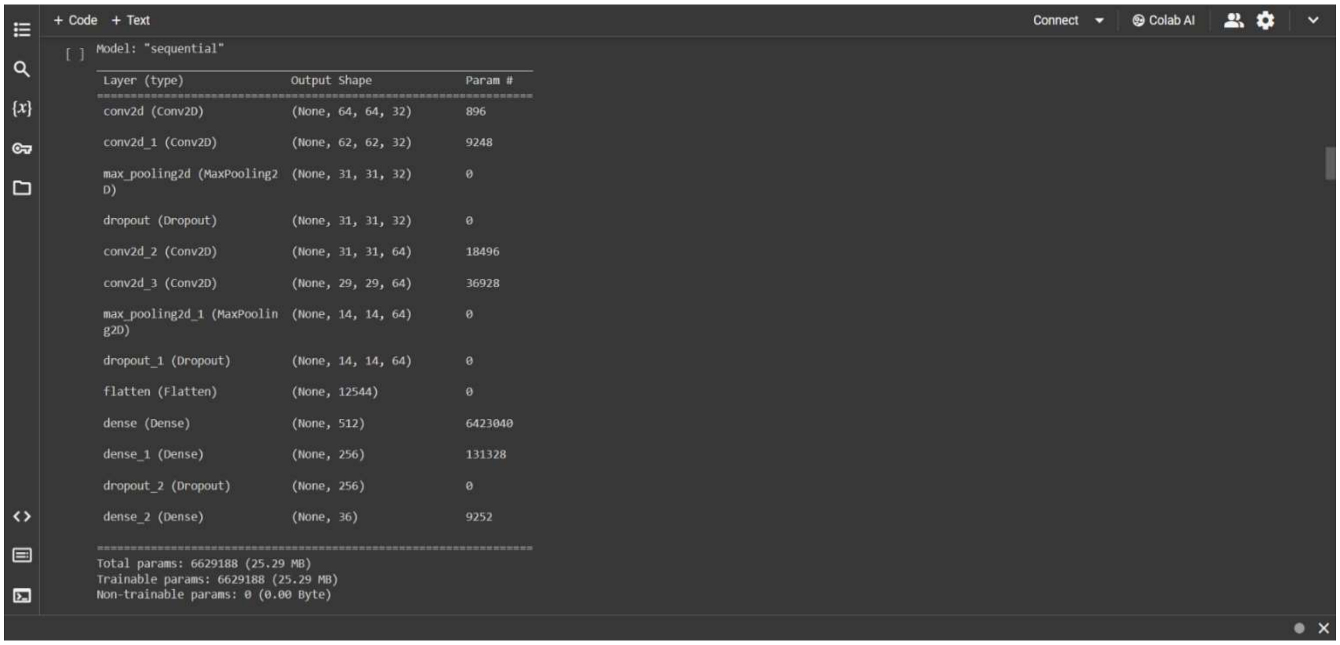
```
cnn.add(tf.keras.layers.Dropout(0.5)) #To avoid overfitting
```

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

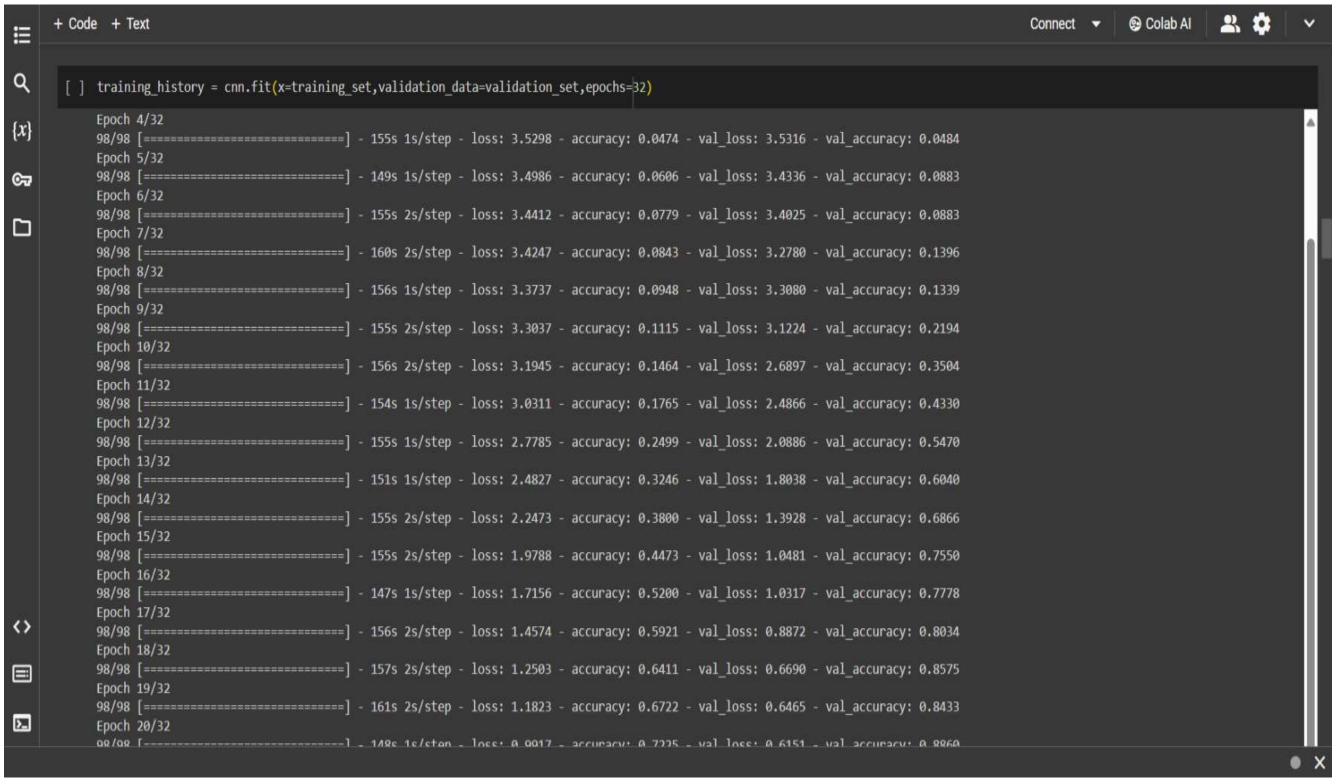
Compiling and Training Phase

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

```
cnn.summary()
```

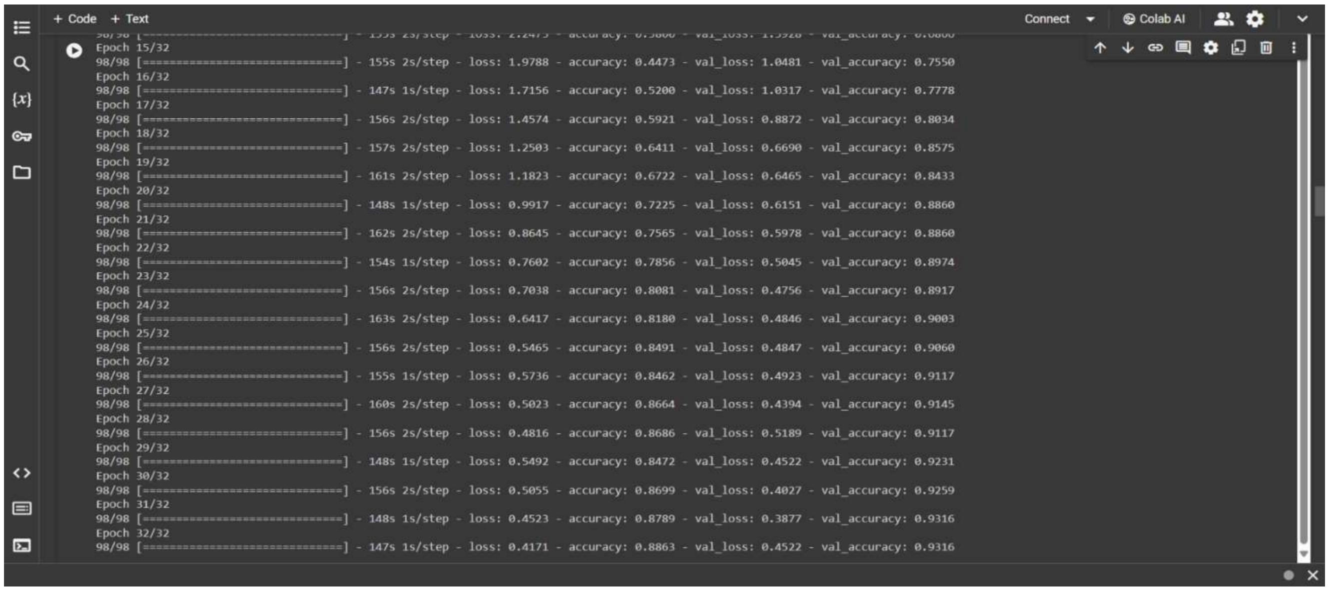


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        896

 conv2d_1 (Conv2D)           (None, 62, 62, 32)        9248

 max_pooling2d (MaxPooling2  (None, 31, 31, 32)        0
 D)

 dropout (Dropout)           (None, 31, 31, 32)        0

 conv2d_2 (Conv2D)           (None, 31, 31, 64)        18496

 conv2d_3 (Conv2D)           (None, 29, 29, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 dropout_1 (Dropout)         (None, 14, 14, 64)        0

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 512)               6423040

 dense_1 (Dense)             (None, 256)               131328

 dropout_2 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 36)                9252

=================================================================
Total params: 6629188 (25.29 MB)
Trainable params: 6629188 (25.29 MB)
Non-trainable params: 0 (0.00 Byte)
```

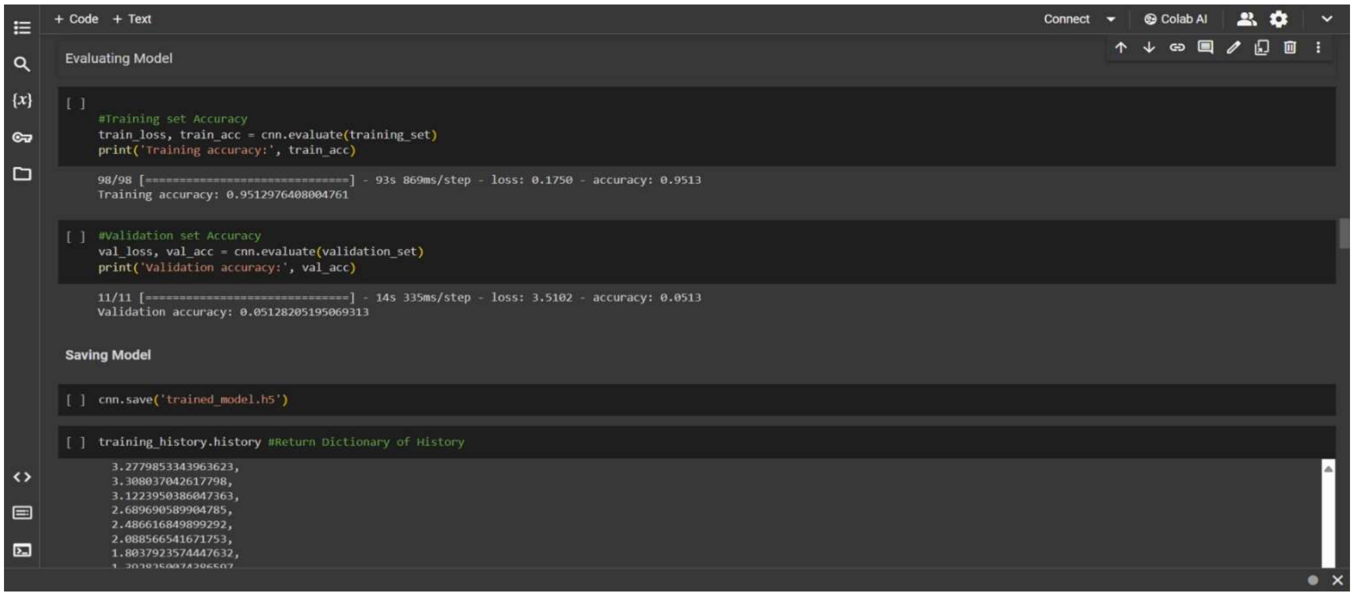


```
training_history = cnn.fit(x=training_set,validation_data=validation_set,epochs=32)
```

```
Epoch 4/32
98/98 [==============================] - 155s 1s/step - loss: 3.5298 - accuracy: 0.0474 - val_loss: 3.5316 - val_accuracy: 0.0484
Epoch 5/32
98/98 [==============================] - 149s 1s/step - loss: 3.4986 - accuracy: 0.0606 - val_loss: 3.4336 - val_accuracy: 0.0883
Epoch 6/32
98/98 [==============================] - 155s 2s/step - loss: 3.4412 - accuracy: 0.0779 - val_loss: 3.4025 - val_accuracy: 0.0883
Epoch 7/32
98/98 [==============================] - 160s 2s/step - loss: 3.4247 - accuracy: 0.0843 - val_loss: 3.2780 - val_accuracy: 0.1396
Epoch 8/32
98/98 [==============================] - 156s 1s/step - loss: 3.3737 - accuracy: 0.0948 - val_loss: 3.3080 - val_accuracy: 0.1339
Epoch 9/32
98/98 [==============================] - 155s 2s/step - loss: 3.3037 - accuracy: 0.1115 - val_loss: 3.1224 - val_accuracy: 0.2194
Epoch 10/32
98/98 [==============================] - 156s 2s/step - loss: 3.1945 - accuracy: 0.1464 - val_loss: 2.6897 - val_accuracy: 0.3504
Epoch 11/32
98/98 [==============================] - 154s 1s/step - loss: 3.0311 - accuracy: 0.1765 - val_loss: 2.4866 - val_accuracy: 0.4330
Epoch 12/32
98/98 [==============================] - 155s 1s/step - loss: 2.7785 - accuracy: 0.2499 - val_loss: 2.0886 - val_accuracy: 0.5470
Epoch 13/32
98/98 [==============================] - 151s 1s/step - loss: 2.4827 - accuracy: 0.3246 - val_loss: 1.8038 - val_accuracy: 0.6040
Epoch 14/32
98/98 [==============================] - 155s 2s/step - loss: 2.2473 - accuracy: 0.3800 - val_loss: 1.3928 - val_accuracy: 0.6866
Epoch 15/32
98/98 [==============================] - 155s 2s/step - loss: 1.9788 - accuracy: 0.4473 - val_loss: 1.0481 - val_accuracy: 0.7550
Epoch 16/32
98/98 [==============================] - 147s 1s/step - loss: 1.7156 - accuracy: 0.5200 - val_loss: 1.0317 - val_accuracy: 0.7778
Epoch 17/32
98/98 [==============================] - 156s 2s/step - loss: 1.4574 - accuracy: 0.5921 - val_loss: 0.8872 - val_accuracy: 0.8034
Epoch 18/32
98/98 [==============================] - 157s 2s/step - loss: 1.2503 - accuracy: 0.6411 - val_loss: 0.6690 - val_accuracy: 0.8575
Epoch 19/32
98/98 [==============================] - 161s 2s/step - loss: 1.1823 - accuracy: 0.6722 - val_loss: 0.6465 - val_accuracy: 0.8433
Epoch 20/32
98/98 [==============================] - 148s 1s/step - loss: 0.9917 - accuracy: 0.7225 - val_loss: 0.6151 - val_accuracy: 0.8860
```



```
Epoch 15/32
98/98 [==============================] - 155s 2s/step - loss: 1.9788 - accuracy: 0.4473 - val_loss: 1.0481 - val_accuracy: 0.7550
Epoch 16/32
98/98 [==============================] - 147s 1s/step - loss: 1.7156 - accuracy: 0.5200 - val_loss: 1.0317 - val_accuracy: 0.7778
Epoch 17/32
98/98 [==============================] - 156s 2s/step - loss: 1.4574 - accuracy: 0.5921 - val_loss: 0.8872 - val_accuracy: 0.8034
Epoch 18/32
98/98 [==============================] - 157s 2s/step - loss: 1.2503 - accuracy: 0.6411 - val_loss: 0.6690 - val_accuracy: 0.8575
Epoch 19/32
98/98 [==============================] - 161s 2s/step - loss: 1.1823 - accuracy: 0.6722 - val_loss: 0.6465 - val_accuracy: 0.8433
Epoch 20/32
98/98 [==============================] - 148s 1s/step - loss: 0.9917 - accuracy: 0.7225 - val_loss: 0.6151 - val_accuracy: 0.8860
Epoch 21/32
98/98 [==============================] - 162s 2s/step - loss: 0.8645 - accuracy: 0.7565 - val_loss: 0.5978 - val_accuracy: 0.8860
Epoch 22/32
98/98 [==============================] - 154s 1s/step - loss: 0.7602 - accuracy: 0.7856 - val_loss: 0.5045 - val_accuracy: 0.8974
Epoch 23/32
98/98 [==============================] - 156s 2s/step - loss: 0.7038 - accuracy: 0.8081 - val_loss: 0.4756 - val_accuracy: 0.8917
Epoch 24/32
98/98 [==============================] - 163s 2s/step - loss: 0.6417 - accuracy: 0.8180 - val_loss: 0.4846 - val_accuracy: 0.9003
Epoch 25/32
98/98 [==============================] - 156s 2s/step - loss: 0.5465 - accuracy: 0.8491 - val_loss: 0.4847 - val_accuracy: 0.9060
Epoch 26/32
98/98 [==============================] - 155s 1s/step - loss: 0.5736 - accuracy: 0.8462 - val_loss: 0.4923 - val_accuracy: 0.9117
Epoch 27/32
98/98 [==============================] - 160s 2s/step - loss: 0.5023 - accuracy: 0.8664 - val_loss: 0.4394 - val_accuracy: 0.9145
Epoch 28/32
98/98 [==============================] - 156s 2s/step - loss: 0.4816 - accuracy: 0.8686 - val_loss: 0.5189 - val_accuracy: 0.9117
Epoch 29/32
98/98 [==============================] - 148s 1s/step - loss: 0.5492 - accuracy: 0.8472 - val_loss: 0.4522 - val_accuracy: 0.9231
Epoch 30/32
98/98 [==============================] - 156s 2s/step - loss: 0.5055 - accuracy: 0.8699 - val_loss: 0.4027 - val_accuracy: 0.9259
Epoch 31/32
98/98 [==============================] - 148s 1s/step - loss: 0.4523 - accuracy: 0.8789 - val_loss: 0.3877 - val_accuracy: 0.9316
Epoch 32/32
98/98 [==============================] - 147s 1s/step - loss: 0.4171 - accuracy: 0.8863 - val_loss: 0.4522 - val_accuracy: 0.9316
```



Evaluating Model

```
[ ]
    #Training set Accuracy
    train_loss, train_acc = cnn.evaluate(training_set)
    print('Training accuracy:', train_acc)
```

```
98/98 [==============================] - 93s 869ms/step - loss: 0.1750 - accuracy: 0.9513
Training accuracy: 0.9512976408004761
```

```
[ ] #Validation set Accuracy
    val_loss, val_acc = cnn.evaluate(validation_set)
    print('Validation accuracy:', val_acc)
```

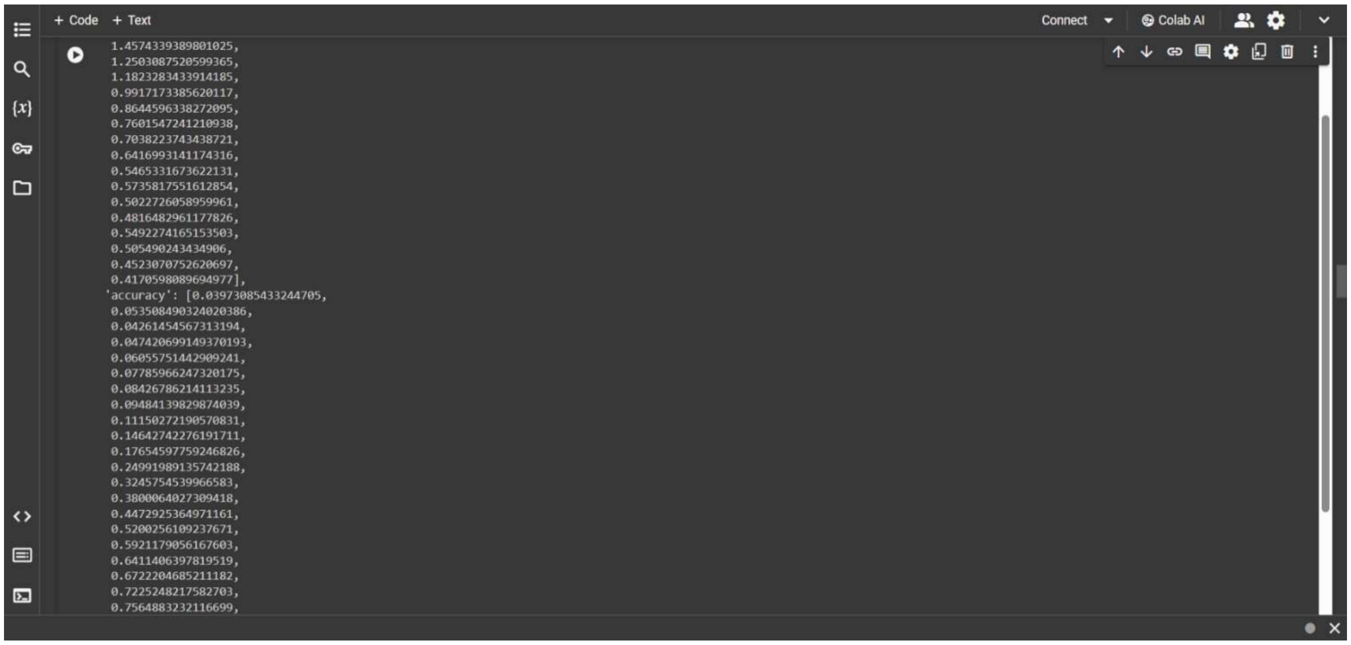
```
11/11 [==============================] - 14s 335ms/step - loss: 3.5102 - accuracy: 0.0513
Validation accuracy: 0.05128205195069313
```

Saving Model

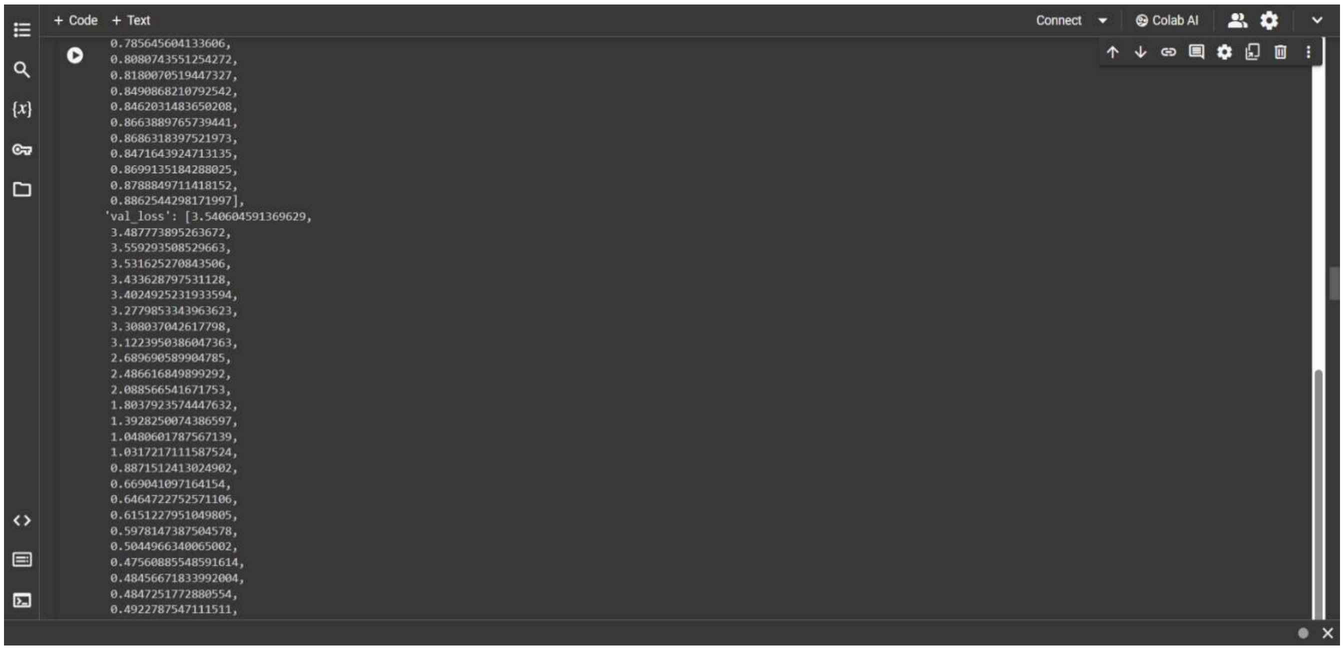
```
[ ] cnn.save('trained_model.h5')
```

```
[ ] training_history.history #Return Dictionary of History
```

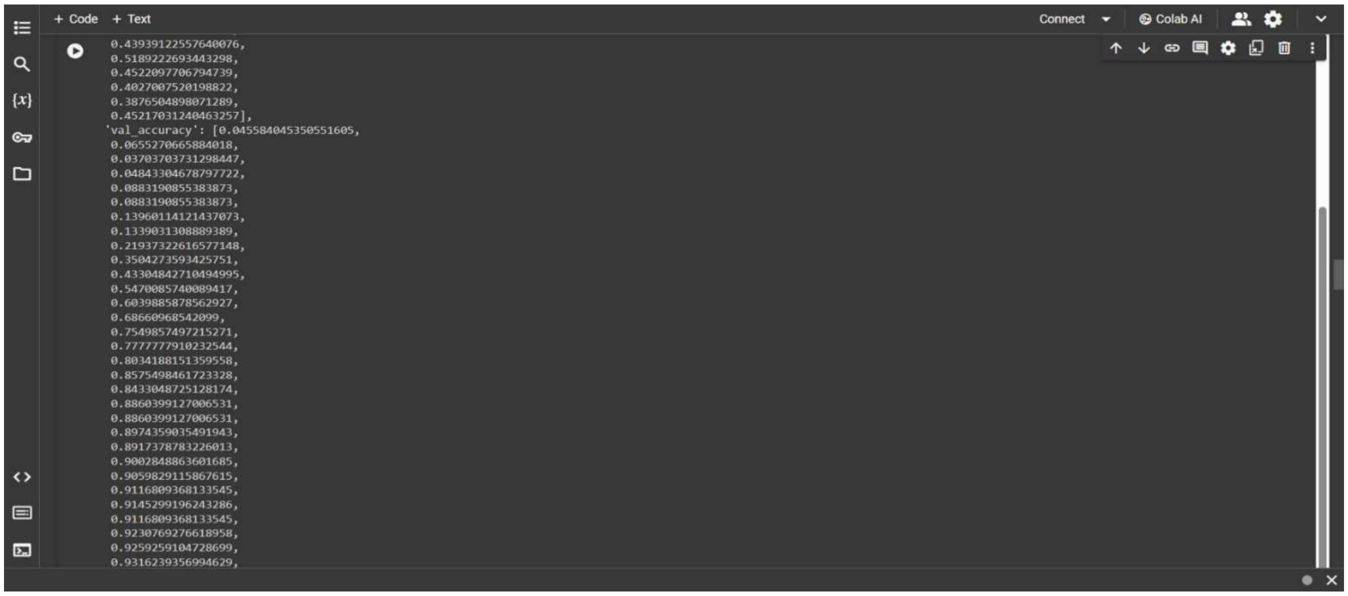
```
  3.2779853343963623,
  3.308037042617798,
  3.1223950386047363,
  2.689690589904785,
  2.486616849899292,
  2.088566541671753,
  1.8037923574447632,
```



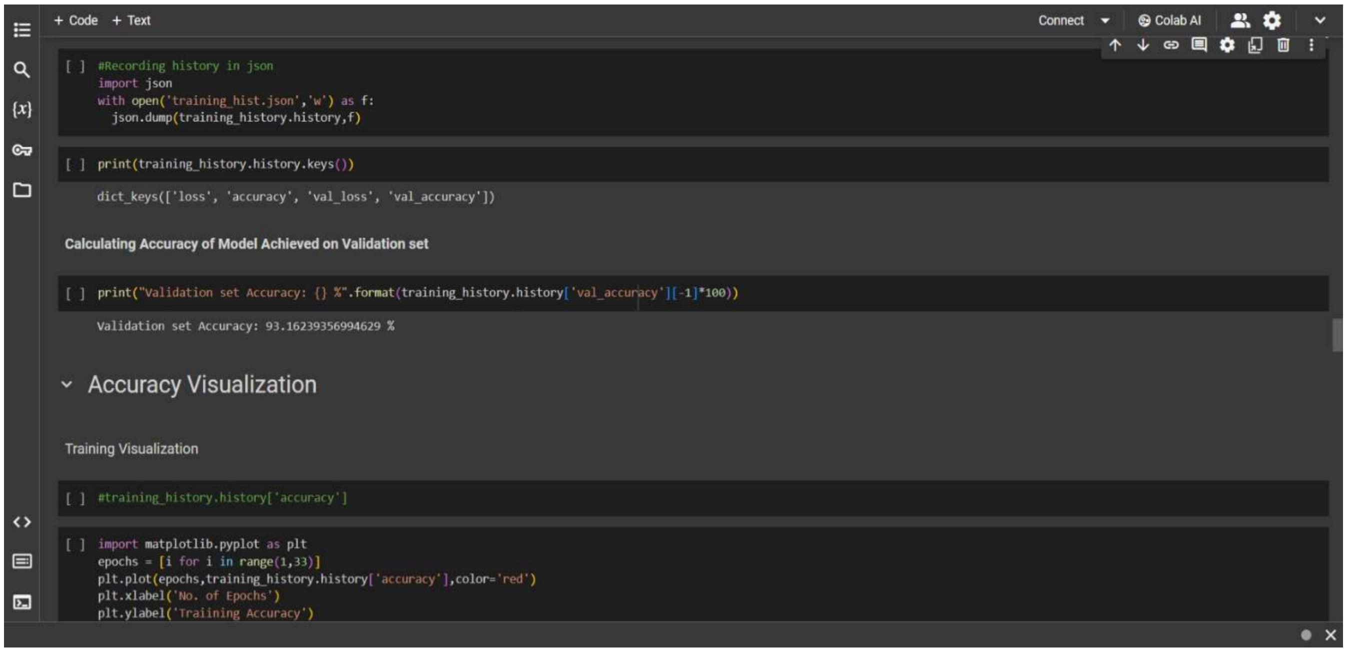
```
  1.4574339389801025,
  1.2503087520599365,
  1.1823283433914185,
  0.9917173385620117,
  0.8644596338272095,
  0.7601547241210938,
  0.7038223743438721,
  0.6416993141174316,
  0.5465331673622131,
  0.5735817551612854,
  0.5022726058959961,
  0.4816482961177826,
  0.5492274165153503,
  0.505490243434906,
  0.4523070752620697,
  0.4170598089694977],
 'accuracy': [0.03973085433244705,
  0.053508490324020386,
  0.04261454567313194,
  0.047420699149370193,
  0.06055751442909241,
  0.07785966247320175,
  0.08426786214113235,
  0.09484139829874039,
  0.11150272190570831,
  0.14642742276191711,
  0.17654597759246826,
  0.24991989135742188,
  0.3245754539966583,
  0.3800064027309418,
  0.4472925364971161,
  0.5200256109237671,
  0.5921179056167603,
  0.6411406397819519,
  0.6722204685211182,
  0.7225248217582703,
  0.7564883232116699,
```



```
 0.785645604133606,
 0.8080743551254272,
 0.8180070519447327,
 0.8490868210792542,
 0.8462031483650208,
 0.8663889765739441,
 0.8686318397521973,
 0.8471643924713135,
 0.8699135184288025,
 0.8788849711418152,
 0.8862544298171997],
'val_loss': [3.540604591369629,
 3.487773895263672,
 3.559293508529663,
 3.531625270843506,
 3.433628797531128,
 3.4024925231933594,
 3.2779853343963623,
 3.308037042617798,
 3.1223950386047363,
 2.689690589904785,
 2.486616849899292,
 2.088566541671753,
 1.8037923574447632,
 1.3928250074386597,
 1.0480601787567139,
 1.0317217111587524,
 0.8871512413024902,
 0.669041097164154,
 0.6464722752571106,
 0.6151227951049805,
 0.5978147387504578,
 0.5044966340065002,
 0.47560885548591614,
 0.48456671833992004,
 0.4847251772880554,
 0.4922787547111511,
```



```
 0.43939122557640076,
 0.5189222693443298,
 0.4522097706794739,
 0.4027007520198822,
 0.3876504898071289,
 0.45217031240463257],
'val_accuracy': [0.045584045350551605,
 0.0655270665884018,
 0.037037037312984467,
 0.048433046787977222,
 0.0883190855383873,
 0.0883190855383873,
 0.13960114121437073,
 0.1339031308889389,
 0.21937322616577148,
 0.3504273593425751,
 0.43304842710494995,
 0.5470085740089417,
 0.6039885878562927,
 0.68660968542099,
 0.7549857497215271,
 0.7777777910232544,
 0.8034188151359558,
 0.8575498461723328,
 0.8433048725128174,
 0.8860399127006531,
 0.8860399127006531,
 0.8974359035491943,
 0.8917378783226013,
 0.9002848863601685,
 0.9059829115867615,
 0.9116809368133545,
 0.9145299196243286,
 0.9116809368133545,
 0.9230769276618958,
 0.9259259104728699,
 0.9316239356994629,
```



```
[ ] #Recording history in json
    import json
    with open('training_hist.json','w') as f:
      json.dump(training_history.history,f)
```

```
[ ] print(training_history.history.keys())

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

Calculating Accuracy of Model Achieved on Validation set

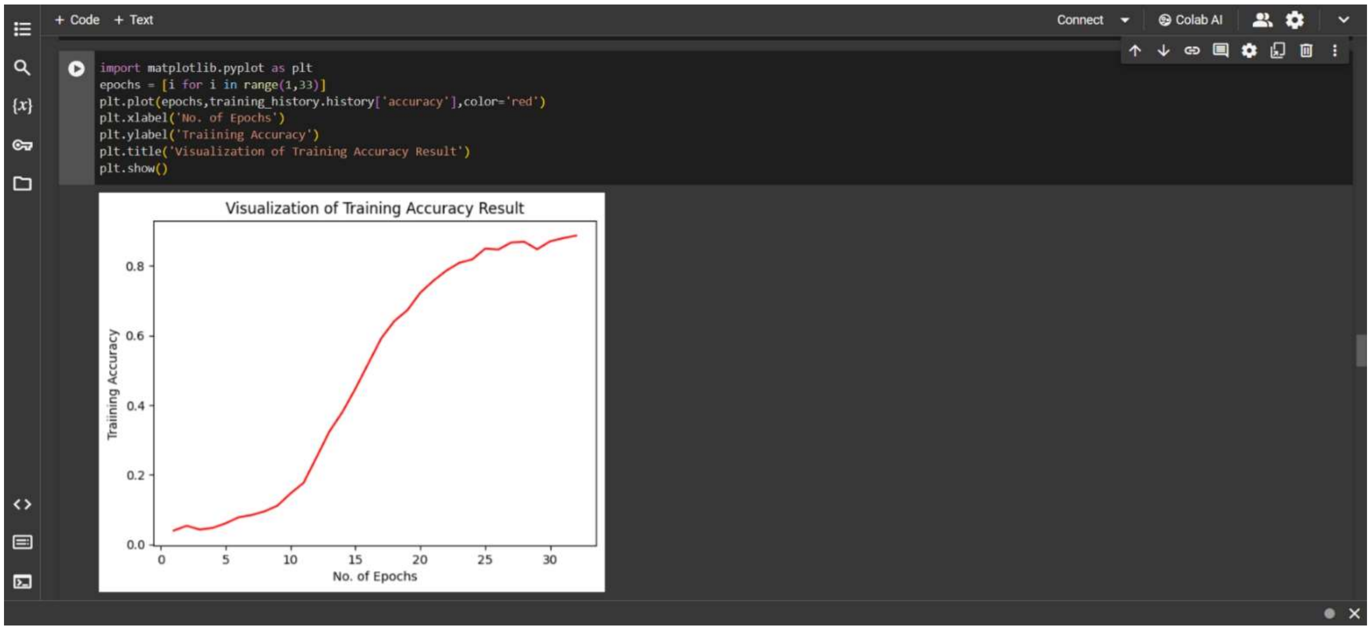
```
[ ] print("Validation set Accuracy: {} %".format(training_history.history['val_accuracy'][-1]*100))

    Validation set Accuracy: 93.16239356994629 %
```

Accuracy Visualization

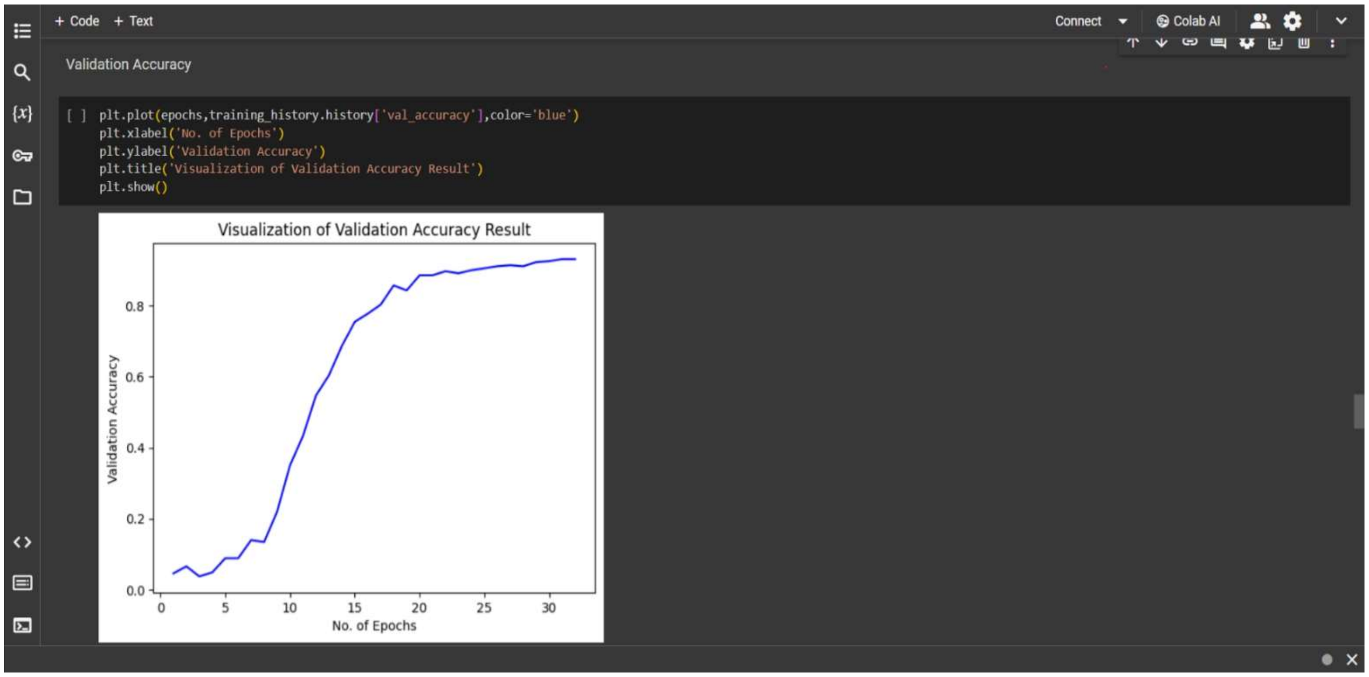
Training Visualization

```
[ ] #training_history.history['accuracy']
```

```
[ ] import matplotlib.pyplot as plt
    epochs = [i for i in range(1,33)]
    plt.plot(epochs,training_history.history['accuracy'],color='red')
    plt.xlabel('No. of Epochs')
    plt.ylabel('Traiining Accuracy')
```

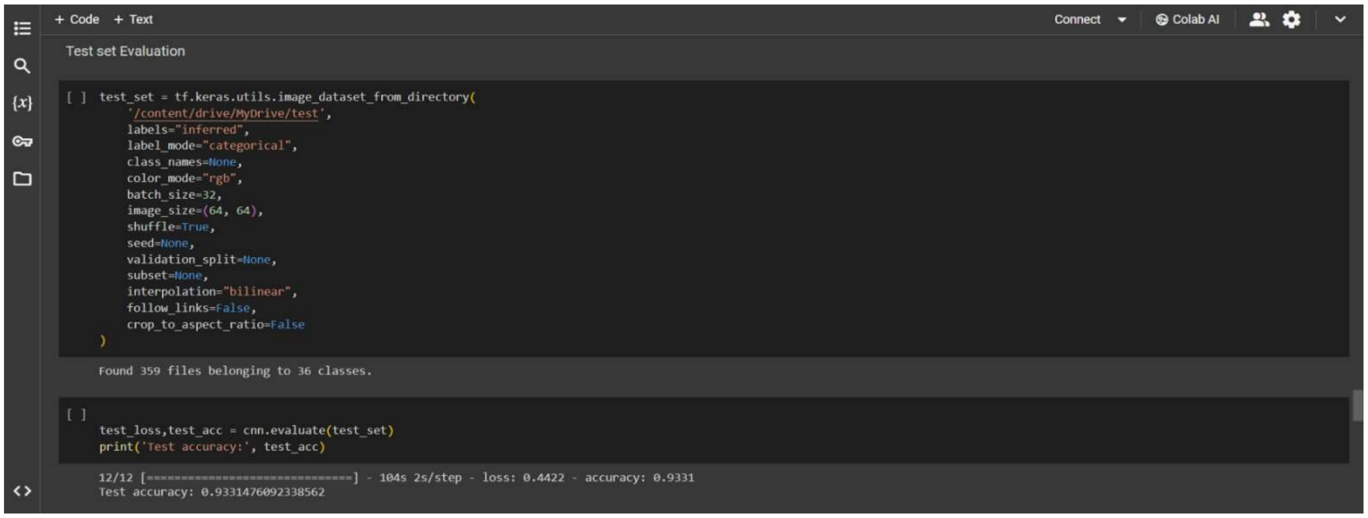


```
import matplotlib.pyplot as plt
epochs = [i for i in range(1,33)]
plt.plot(epochs,training_history.history['accuracy'],color='red')
plt.xlabel('No. of Epochs')
plt.ylabel('Traiining Accuracy')
plt.title('Visualization of Training Accuracy Result')
plt.show()
```



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()
```



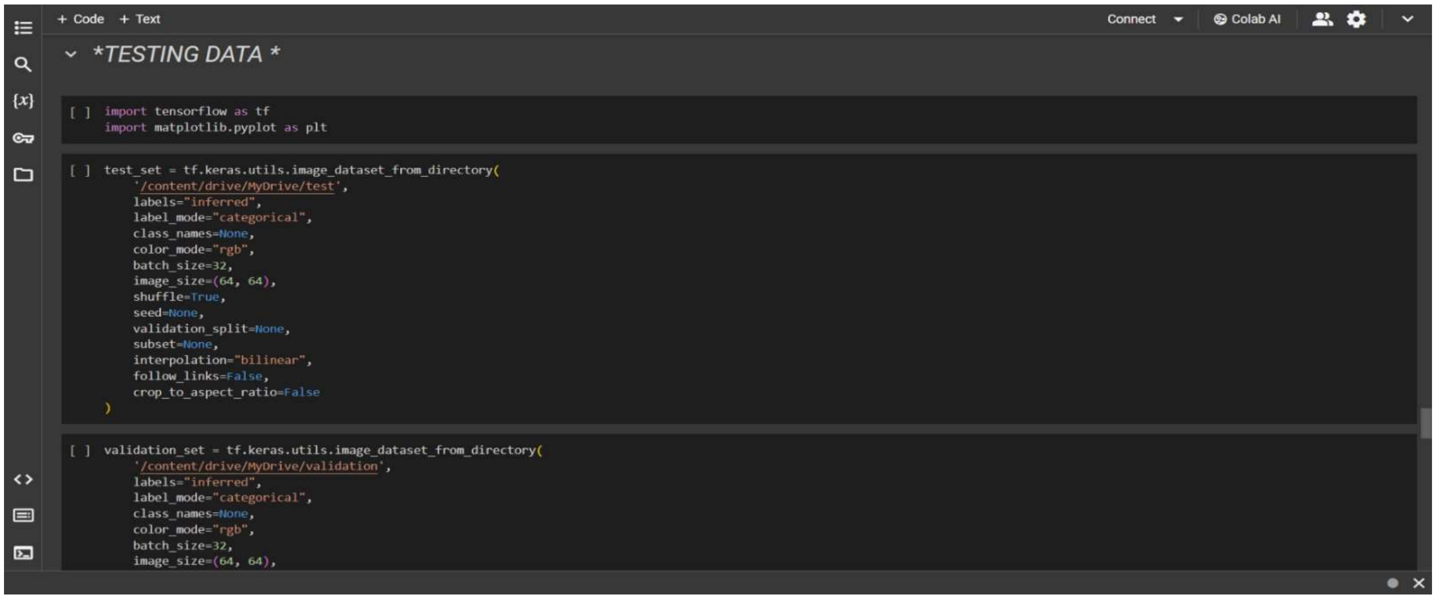
Test set Evaluation

```
[ ] test_set = tf.keras.utils.image_dataset_from_directory(
        '/content/drive/MyDrive/test',
        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 359 files belonging to 36 classes.
```

```
[ ]
    test_loss,test_acc = cnn.evaluate(test_set)
    print('Test accuracy:', test_acc)
```

```
12/12 [==============================] - 104s 2s/step - loss: 0.4422 - accuracy: 0.9331
Test accuracy: 0.9331476092338562
```

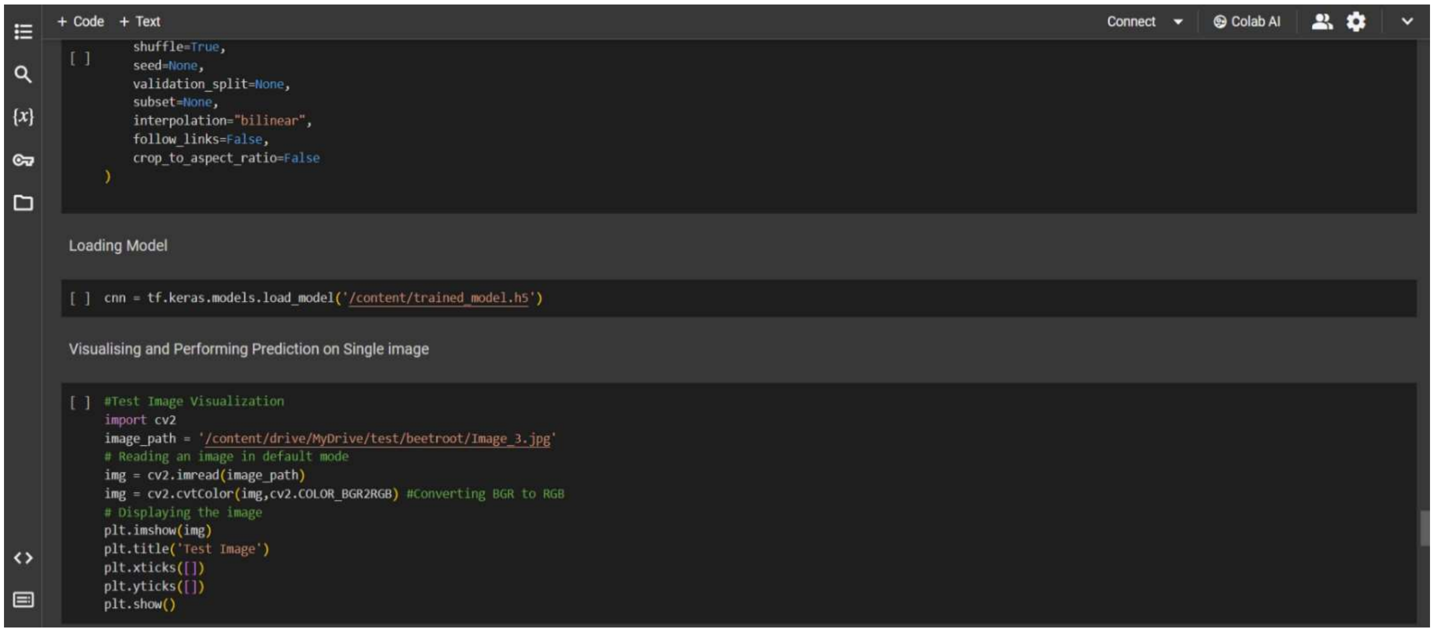


*TESTING DATA *

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

```
test_set = tf.keras.utils.image_dataset_from_directory(
    '/content/drive/MyDrive/test',
    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
)
```

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



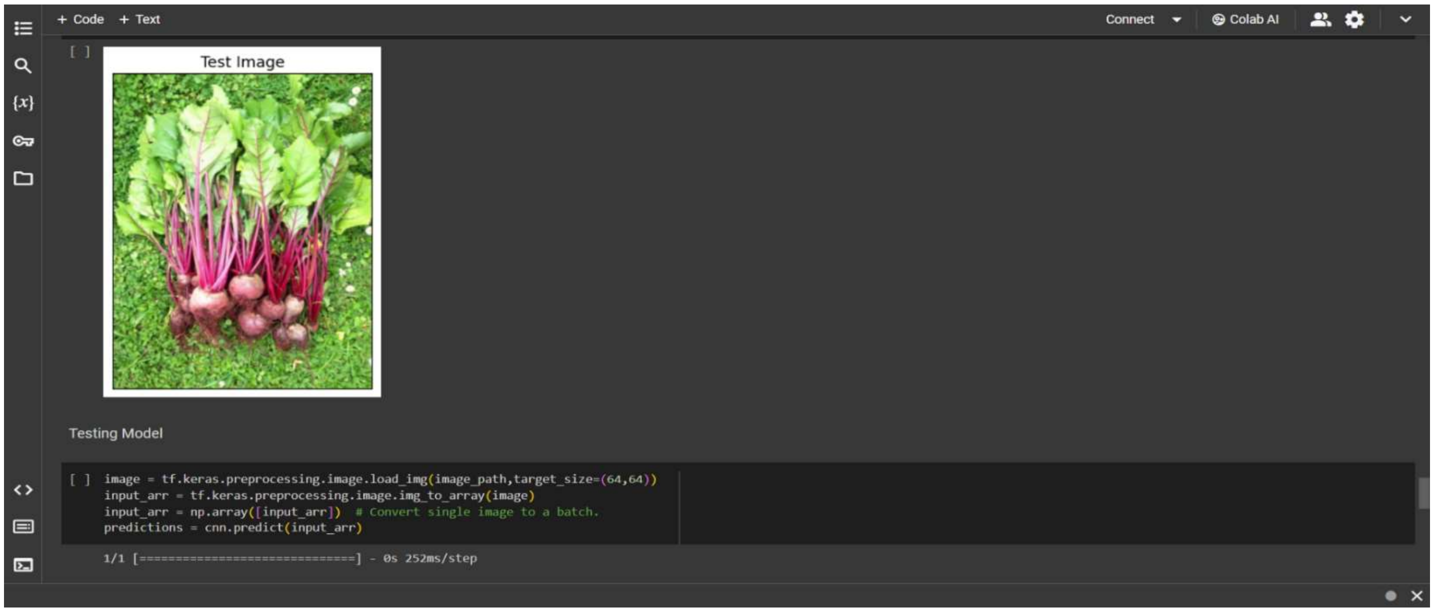
```
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

Loading Model

```
cnn = tf.keras.models.load_model('/content/trained_model.h5')
```

Visualising and Performing Prediction on Single image

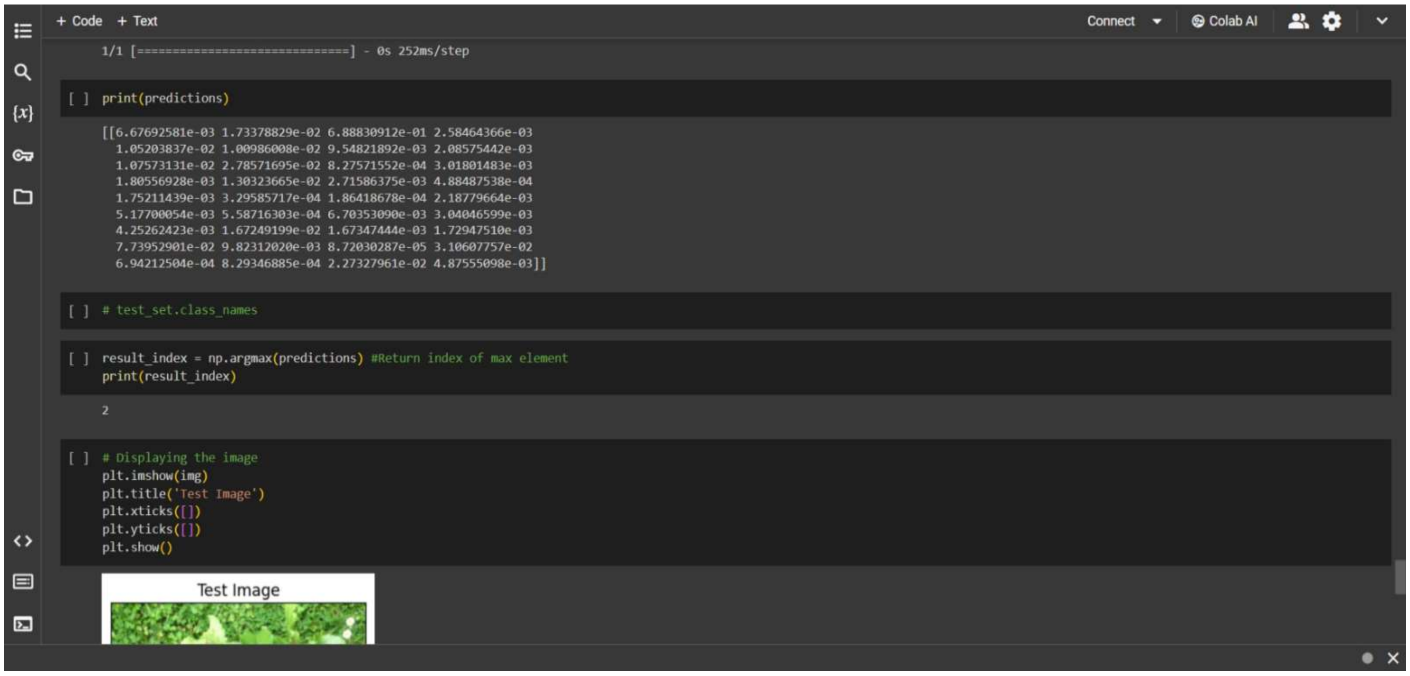
```
#Test Image Visualization
import cv2
image_path = '/content/drive/MyDrive/test/beetroot/Image_3.jpg'
# Reading an image in default mode
img = cv2.imread(image_path)
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB) #Converting BGR to RGB
# Displaying the image
plt.imshow(img)
plt.title('Test Image')
plt.xticks([])
plt.yticks([])
plt.show()
```



Testing Model

```
image = tf.keras.preprocessing.image.load_img(image_path,target_size=(64,64))
input_arr = tf.keras.preprocessing.image.img_to_array(image)
input_arr = np.array([input_arr])  # Convert single image to a batch.
predictions = cnn.predict(input_arr)
```

```
1/1 [==============================] - 0s 252ms/step
```



```
1/1 [==============================] - 0s 252ms/step
```

```
[ ] print(predictions)
```

```
[[6.67692581e-03 1.73378829e-02 6.88830912e-01 2.58464366e-03
  1.05203837e-02 1.00986008e-02 9.54821892e-03 2.08575442e-03
  1.07573131e-02 2.78571695e-02 8.27571552e-04 3.01801483e-03
  1.80556928e-03 1.30323665e-02 2.71586375e-03 4.88487538e-04
  1.75211439e-03 3.29585717e-04 1.86418678e-04 2.18779664e-03
  5.17700054e-03 5.58716303e-04 6.70353090e-03 3.04046599e-03
  4.25262423e-03 1.67249199e-02 1.67347444e-03 1.72947510e-03
  7.73952901e-02 9.82312020e-03 8.72030287e-05 3.10607757e-02
  6.94212504e-04 8.29346885e-04 2.27327961e-02 4.87555098e-03]]
```

```
[ ] # test_set.class_names
```

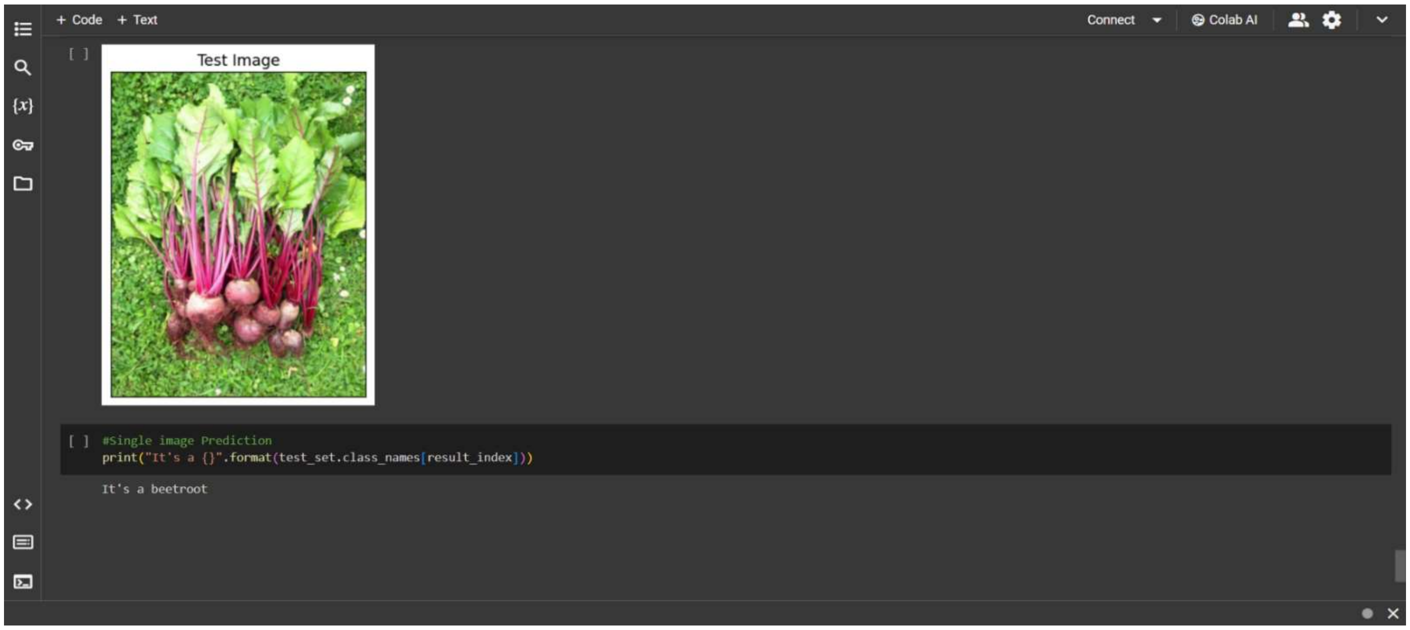
```
[ ] result_index = np.argmax(predictions) #Return index of max element
    print(result_index)
```

```
2
```

```
[ ] # Displaying the image
    plt.imshow(img)
    plt.title('Test Image')
    plt.xticks([])
    plt.yticks([])
    plt.show()
```

CHAPTER 7

RESULT AND DISCUSSION



```
#Single image Prediction
print("It's a {}".format(test_set.class_names[result_index]))
```

```
It's a beetroot
```

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

8.1 Conclusion

Given a picture of food as input to the system, it will quickly recognise the food item/items in the image with it's calorie value as output. Today about 30% of the entire human population is obese and overight. Obesity has been directly linked with various diseases such as diabetes, high blood pressure and even cancer. Majority of similar applications come with premium packages, hover our application would is free. On a social level, it will help bring awareness among people with respect to the food items they consume as ll as the amount of calorie intake. This would in turn lead to a fall in the fraction of population suffering from obesity.

8.2 Future Scope

There is , hover one part of the project that can be worked upon. It is the calorie estimation part. With the given time frame, I was only able to finish up a food item detection model. A calorie estimation model, consdering the wide aspects of it, would require a longer time frame. Other than this, one could work upon creating this aplication as a non-volative user food tracking system which my application is not,

REFERENCES

- Dataset - https://www.kaggle.com/datasets/kritikseth/fruit-and-vegetable-image-recognition

- https://youtube.com/playlist?list=PLvz5lCwTgdXByZ_z-LFo4vJbbFIMPhkkM&si=uvTGHf3tzAwex8MA

- Liu, Y.-Y.; Huang- Deep Learning Model for Computer-Aided Diagnosis of Urolithiasis Detection from Kidney–Ureter–Bladder Images. Bioengineering 2022, 9, 811. https://doi.org/10.3390/bioengineering9120811
- Zhang, H., Botler, M., & Kooman, J. P. (2023). Deep Learning for Image Analysis in Kidney Care. Advances in Kidney Disease and Health, 30(1), 25-32. https://doi.org/10.1053/j.akdh.2022.11.003
- Uhm, KH., Jung, SW., Choi, M.H. et al. Deep learning for end-to-end kidney cancer diagnosis on multi-phase abdominal
- computed tomography. npj Precis. Onc. 5, 54 (2021). https://doi.org/10.1038/s41698-021-00195-y
- George, M., Anita, H.B. (2022). Analysis of Kidney Ultrasound Images Using Deep Learning and Machine Learning Techniques, Singapore. https://doi.org/10.1007/978-981-16-5640- 8_15