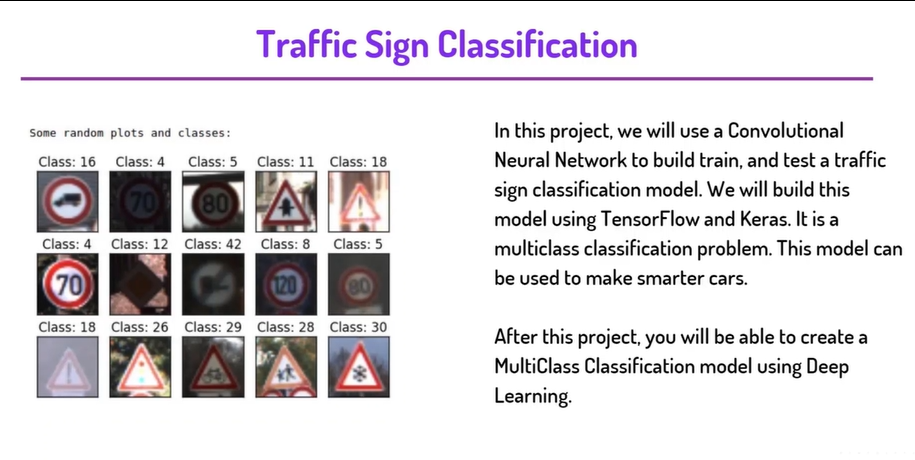
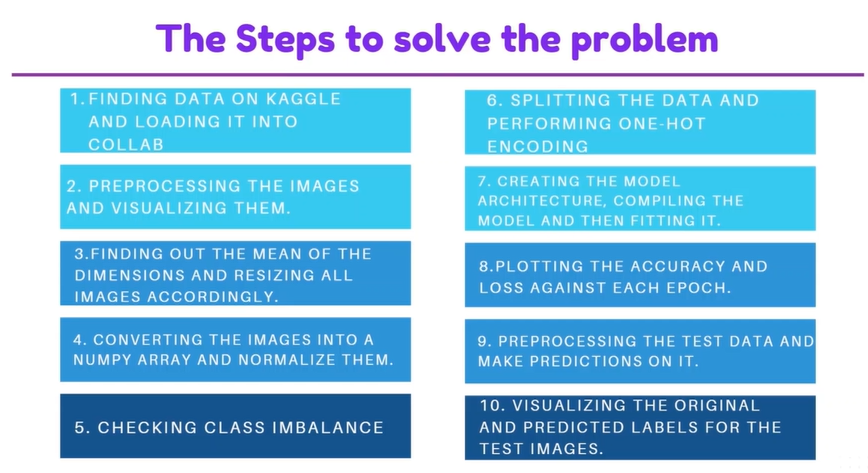
Traffic Sign Classification

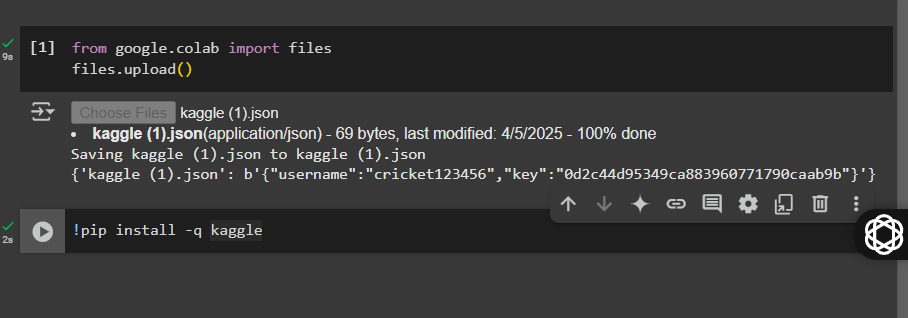




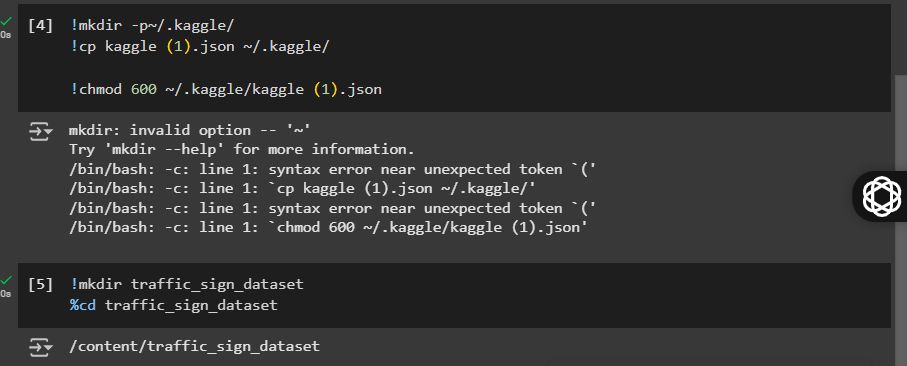


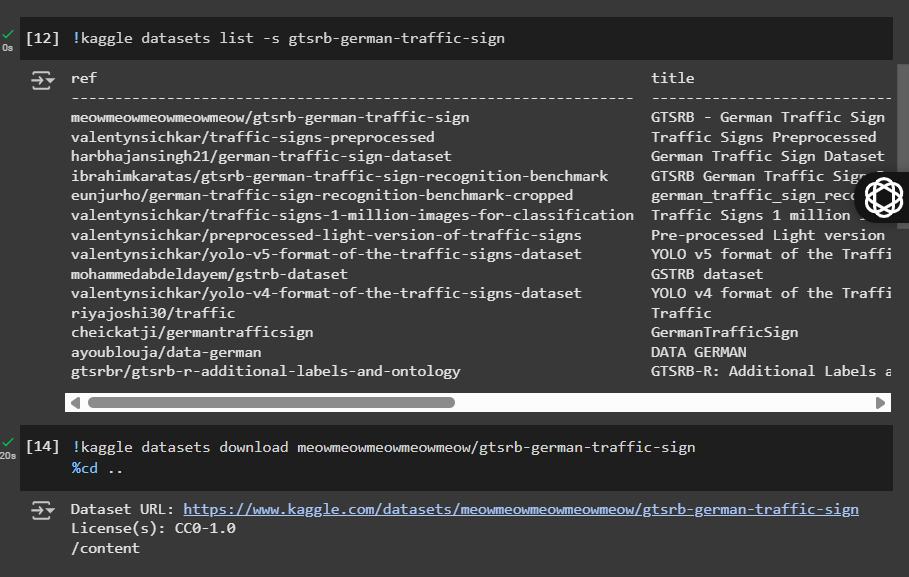
In this project, we will use Convolutional Neural Network to build train and test a traffic sign classification model. We will build this model using

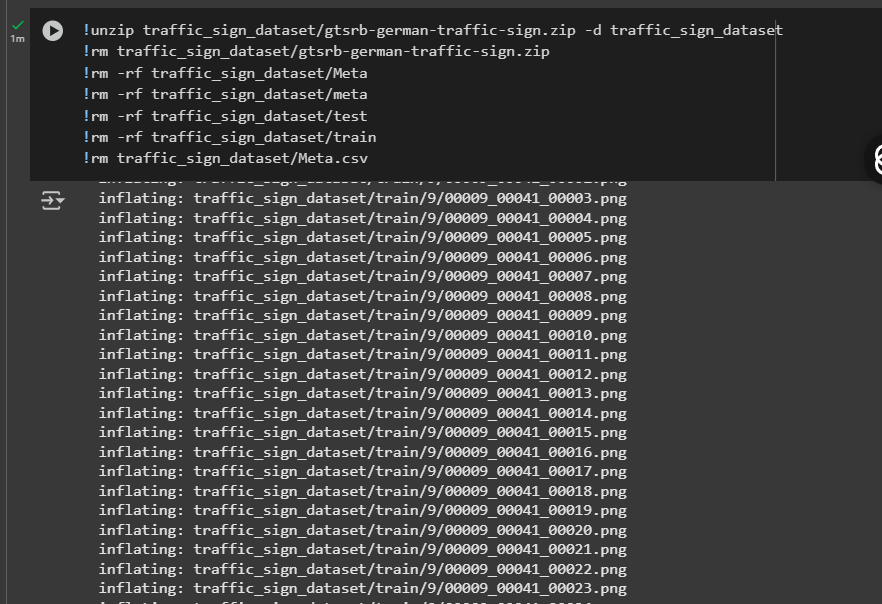
TensorFlow and Keras. It is a multiclass classification problem. This model can be used to make smarter cars.



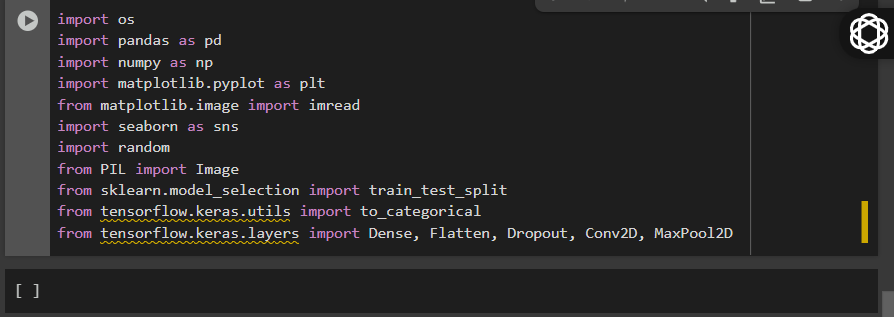
We have import the api key to the google colab for connecting and also install the api client

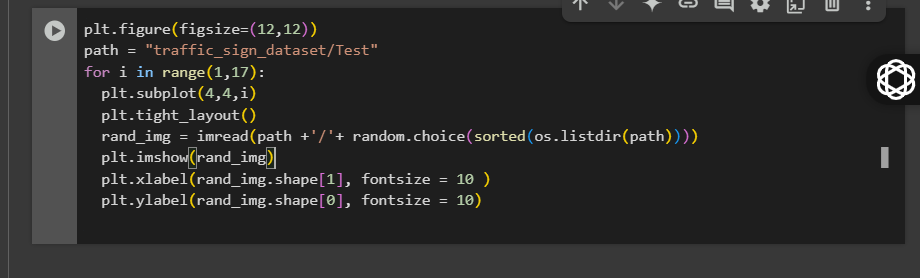
make directory to save the api and also make directory for the datasets

…..we have download the datasets



We have unzipped the data than try to remove unnecessary files

……….we have go and import the important libraries



**Full Code (for reference)**

plt.figure(figsize=(12,12))

path = "traffic\_sign\_dataset/Test"

for i in range(1,17):

plt.subplot(4,4,i)

plt.tight\_layout()

rand\_img = imread(path +'/'+ random.choice(sorted(os.listdir(path))))

plt.imshow(rand\_img)

plt.xlabel(rand\_img.shape[1], fontsize = 10 )

plt.ylabel(rand\_img.shape[0], fontsize = 10)

**Line-by-Line Explanation**

**plt.figure(figsize=(12,12))**

* **Creates a new figure** (the canvas for the plots).
* figsize=(12,12) means the figure will be **12 inches wide and 12 inches tall**.
* You want a large enough figure to fit 16 images nicely.

**path = "traffic\_sign\_dataset/Test"**

* Sets the **path to the folder** where the test images are stored.
* This is used to randomly pick images from that directory.

**for i in range(1,17):**

* A for loop that runs **16 times** (1 to 16).
* You’ll be plotting 16 random images — one per iteration.

**plt.subplot(4,4,i)**

* This creates a **subplot grid of 4 rows × 4 columns** (16 total plots).
* i is the position of the current image (1 to 16).
* Example: 1st image at position 1, 2nd at position 2, ..., 16th at position 16.

**plt.tight\_layout()**

* **Adjusts spacing** between subplots automatically.
* Prevents overlapping of images, labels, and titles.
* You can call this once **outside the loop**, but inside is fine too.

**rand\_img = imread(path + '/' + random.choice(sorted(os.listdir(path))))**

* Randomly selects an image from the Test directory:
  + os.listdir(path) → lists all file names
  + sorted(...) → optional sorting (makes it consistent)
  + random.choice(...) → randomly picks one filename
* imread(...) → loads the selected image into memory

Make sure you’ve imported:

import os, random

from matplotlib.image import imread

**plt.imshow(rand\_img)**

* Displays the selected image in the current subplot.

**plt.xlabel(rand\_img.shape[1], fontsize=10)**

* Adds a **label on the x-axis** showing the **width** (number of columns/pixels) of the image.

**plt.ylabel(rand\_img.shape[0], fontsize=10)**

* Adds a **label on the y-axis** showing the **height** (number of rows/pixels) of the image.

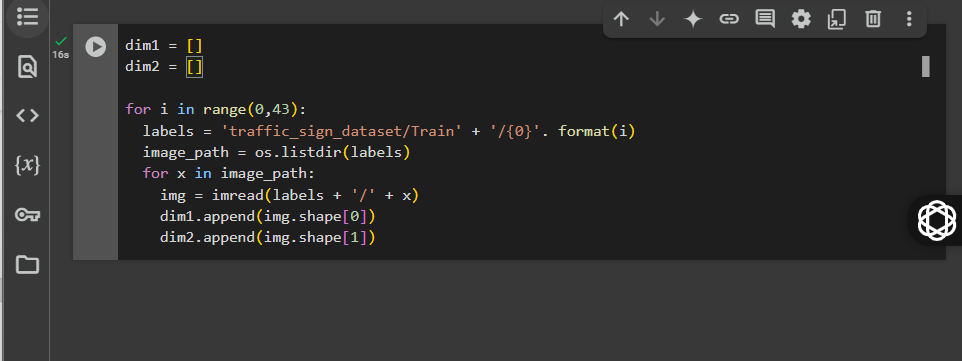
**Output:**

* A **4×4 grid** showing **16 random traffic sign images**
* Each image shows its **width** and **height** on the axes

**Optional Improvements:**

* Add plt.title(...) to show filename or class
* Add error handling in case the folder contains non-image files





**Purpose of the Code**

This script loops through each class folder (from 0 to 42) in the training dataset, reads every image inside, and records the image dimensions—specifically the height and width of each image.

**Line-by-Line Explanation**

dim1 = []

dim2 = []

Initializes two empty lists:

* dim1 will store the height of each image
* dim2 will store the width of each image

for i in range(0, 43):

Loops over 43 class folders labeled 0 to 42.

labels = 'traffic\_sign\_dataset/Train' + '/{0}'.format(i)

Constructs the path to each class folder.  
For example:

* When i = 0, the path becomes traffic\_sign\_dataset/Train/0

image\_path = os.listdir(labels)

Lists all the image filenames in the current class folder.

for x in image\_path:

Iterates through each image file in the folder.

img = imread(labels + '/' + x)

Reads the image using imread, which loads the image as a NumPy array.

dim1.append(img.shape[0])

dim2.append(img.shape[1])

* img.shape[0] gives the image height (number of rows)
* img.shape[1] gives the image width (number of columns)  
  Both values are appended to the respective lists.

**What You Can Do Next**

Now that you have the height and width of all images stored in dim1 and dim2, you can perform analysis or visualizations. Here are a few examples.

**View the most common image sizes**

import pandas as pd

df\_dims = pd.DataFrame({'Height': dim1, 'Width': dim2})

print(df\_dims.value\_counts().head())

**Visualize the distribution of heights and widths**

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))

sns.histplot(dim1, bins=20, kde=True, color='blue', label='Height')

sns.histplot(dim2, bins=20, kde=True, color='green', label='Width')

plt.legend()

plt.title("Distribution of Image Dimensions")

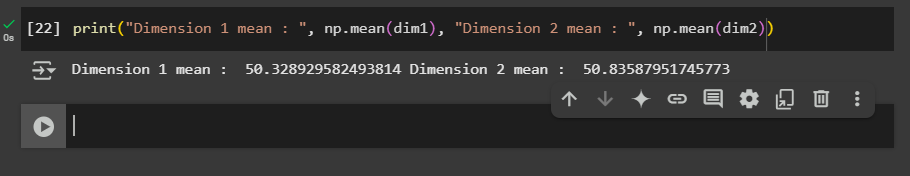
plt.xlabel("Pixels")

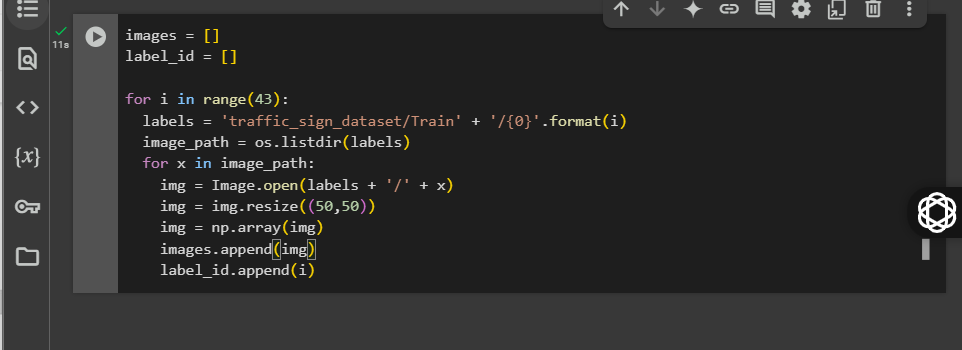
plt.ylabel("Frequency")

plt.show()

**Why This is Important**

* Helps you understand the consistency or variation in image sizes.
* Essential for deciding whether to resize images to a standard dimension before training a machine learning model.
* Consistent input sizes are required for neural network models such as CNNs.

…now we are going find the mean of the dim 1,2



**Purpose**

This code:

* Loads all images from the training dataset (Train/0, Train/1, ..., Train/42)
* Resizes each image to a fixed size of 50x50 pixels
* Converts each image into a NumPy array
* Appends the image data and corresponding label to separate lists

This is often the first step in preparing a dataset for training a machine learning model.

**Line-by-Line Explanation**

images = []

label\_id = []

Initializes two empty lists:

* images: to store the image data as NumPy arrays
* label\_id: to store the numeric label (class index) of each image

for i in range(43):

Loops over each class folder from 0 to 42 (total 43 traffic sign categories).

labels = 'traffic\_sign\_dataset/Train' + '/{0}'.format(i)

Constructs the path to the current class folder, for example:

* traffic\_sign\_dataset/Train/0
* traffic\_sign\_dataset/Train/1 and so on.

image\_path = os.listdir(labels)

Lists all the filenames inside the current class folder.

for x in image\_path:

Loops through each image file in the class folder.

img = Image.open(labels + '/' + x)

Opens the image file using the PIL.Image module.

Make sure to include:

from PIL import Image

import numpy as np

img = img.resize((50, 50))

Resizes the image to 50x50 pixels to ensure uniformity (required for input to a model like CNN).

img = np.array(img)

Converts the image from a PIL object to a NumPy array so it can be used as numerical data for machine learning.

images.append(img)

Adds the processed image array to the images list.

label\_id.append(i)

Adds the corresponding label (i.e., folder index i) to the label\_id list.

**What You Have After This**

* images: a list of all resized images as NumPy arrays (each of shape 50x50x3 if RGB)
* label\_id: a list of class labels (integers 0 to 42)

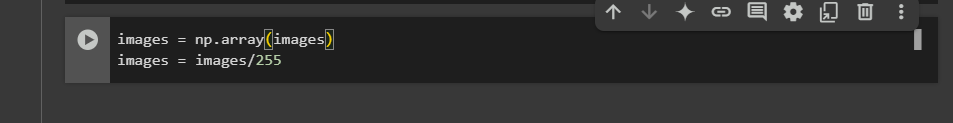
**Optional: Convert to NumPy Arrays**

You can now convert both lists into arrays for modeling:

images = np.array(images)

label\_id = np.array(label\_id)

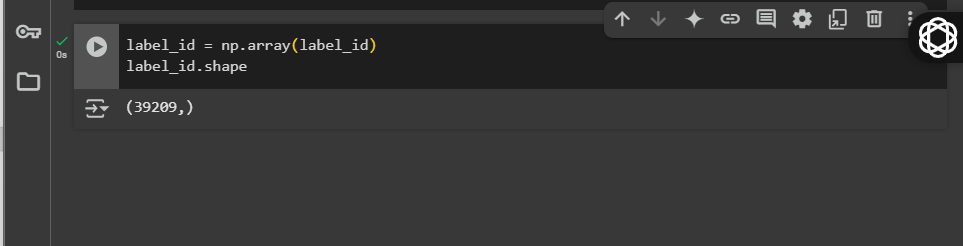
Now images has shape (total\_images, 50, 50, 3)  
And label\_id has shape (total\_images,)

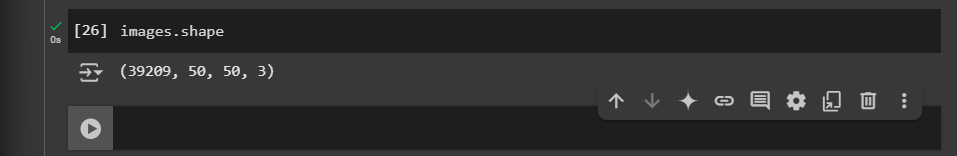


Converting images into numpy array

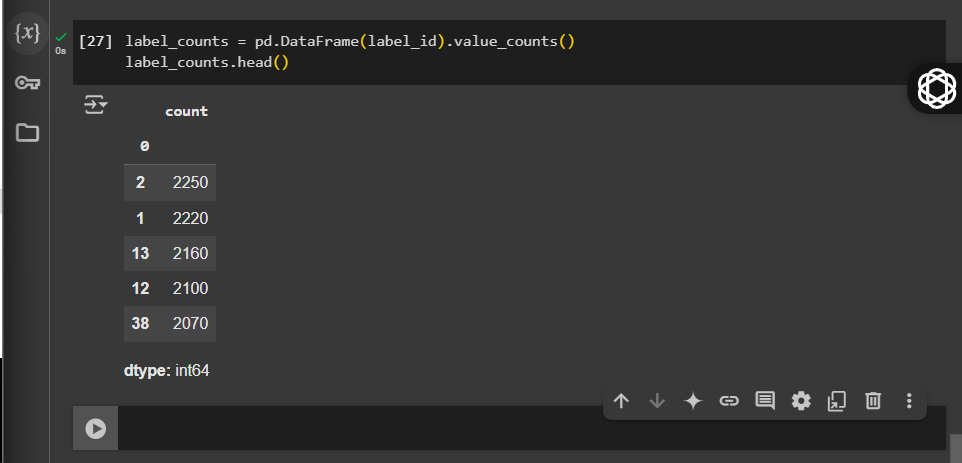
The pixel value of each image ranges between 0 to 255

Dividing each image by 255 will scale the values between 0 to 1 this is also known as normalization

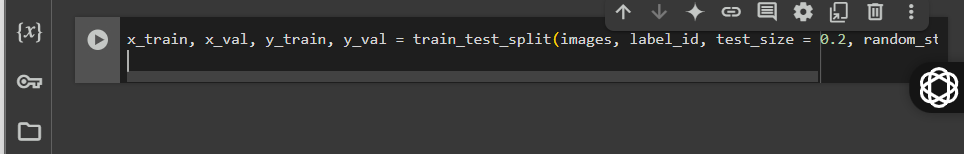
  
storing the label ids into numpy array and printing the shape here we can observe that there are 39209 label ids

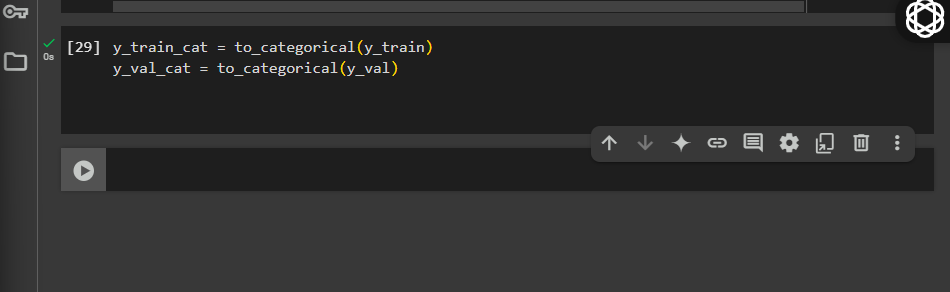


Checking the shape of the images here we can see that there are 39209 Images with the shape of 50,50,3.

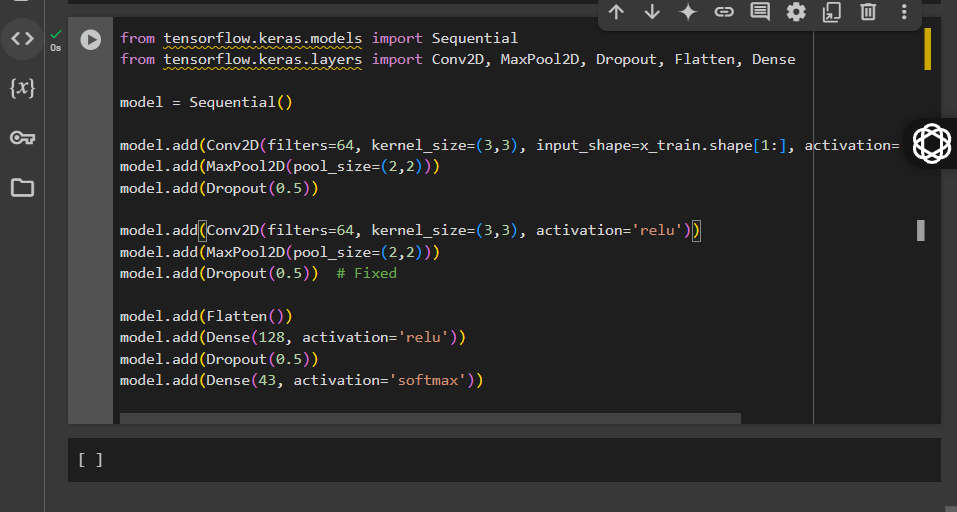


Now we will observe the images per class for checking whether the data is balanced or not from the result we can say that data is balanced.

.now we are going to split the data

keras has a built in function for one-hot encoding

That is converting the classes column into categorical using to categorical( ) function



**Code Breakdown – Line by Line**

**Step 1: Import Required Layers and Model**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPool2D, Dropout, Flatten, Dense

* Sequential: A linear stack of layers, ideal for a straightforward feed-forward network.
* Conv2D: Performs 2D convolution, essential for extracting features from images.
* MaxPool2D: Reduces the spatial size of the feature maps (downsampling).
* Dropout: Prevents overfitting by randomly disabling neurons during training.
* Flatten: Converts the 2D output of the convolutional layers to a 1D vector for the dense layers.
* Dense: Fully connected (feedforward) layer.

**Step 2: Initialize the Model**

model = Sequential()

Creates a new sequential model — layers will be added to it one by one.

**Step 3: First Convolutional Block**

model.add(Conv2D(filters=64, kernel\_size=(3,3), input\_shape=x\_train.shape[1:], activation='relu', padding='same'))

* Applies 64 convolutional filters of size 3x3.
* input\_shape=x\_train.shape[1:] gives the input size as (height, width, channels).
* activation='relu' introduces non-linearity.
* padding='same' ensures output size is the same as input.

model.add(MaxPool2D(pool\_size=(2,2)))

* Downsamples the feature maps by taking the max value in each 2x2 window.

model.add(Dropout(0.5))

* Randomly sets 50% of neurons to zero during training to prevent overfitting.

**Step 4: Second Convolutional Block**

model.add(Conv2D(filters=64, kernel\_size=(3,3), activation='relu'))

* Another convolutional layer with 64 filters and 3x3 kernels.
* No input\_shape needed — it’s automatically inferred.

model.add(MaxPool2D(pool\_size=(2,2)))

* Further reduces the size of the feature maps.

model.add(Dropout(0.5))

* Again, adds regularization by dropping 50% of the neurons.

**Step 5: Flatten and Dense Layers**

model.add(Flatten())

* Converts the 2D matrix (feature maps) into a 1D vector, required before Dense layers.

model.add(Dense(128, activation='relu'))

* A fully connected layer with 128 neurons and ReLU activation.

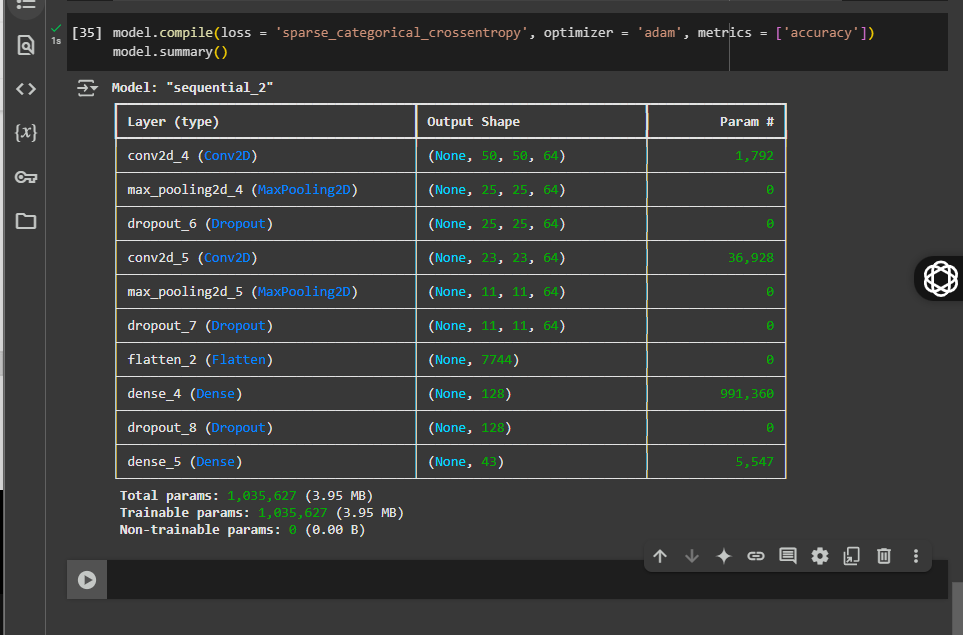
model.add(Dropout(0.5))

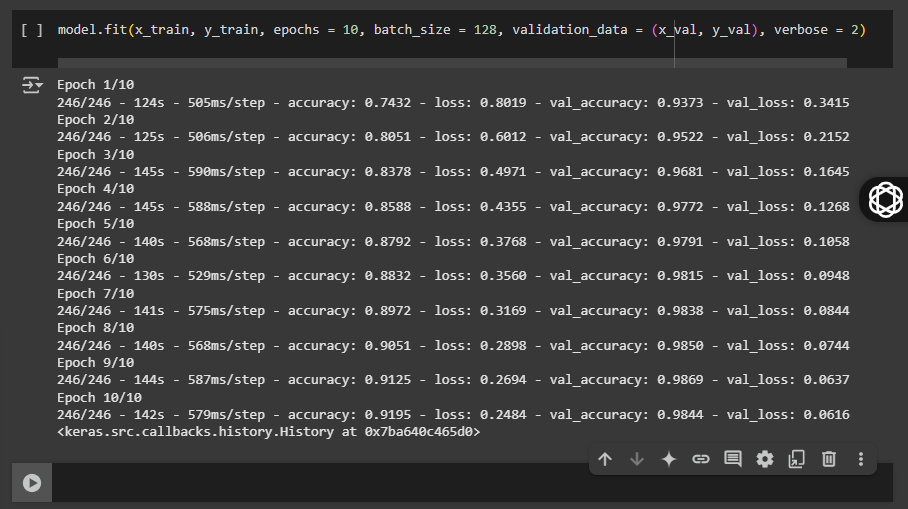
* Adds dropout to this dense layer for further regularization.

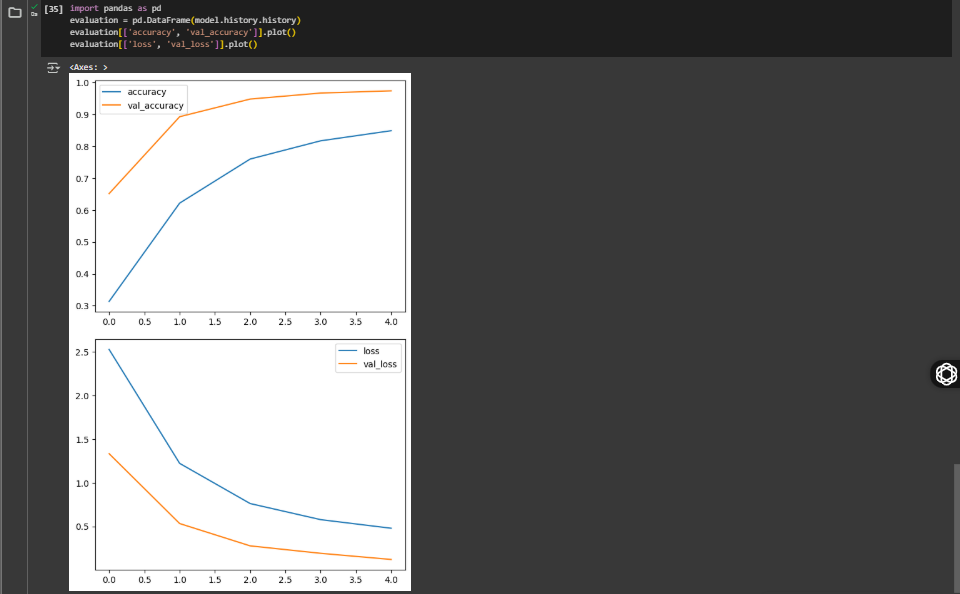
**Step 6: Output Layer**

model.add(Dense(43, activation='softmax'))

* Final layer with **43 neurons** (for 43 traffic sign classes).
* softmax activation outputs a probability distribution over all classes.

……this it he summary of the model

now we will fit the model and observe how our is getting trained on each epoch

….show the accuracy plotting using pandas