GROUP 1
VISUAL RECOGNITION

VISUAL RECOGNITION IN AI

Visual recognition is an evolving field in AI that focuses on enabling machines to see and understand the visual world similarly to humans.

It relies on deep learning algorithms particularly CNN’s.

These algorithms are trained on massive datasets of labeled images and videos, allowing them to extract features and patterns from visual data.

ALGORITHMS USED

Convolutional Neutral Network (CNN)

Random Forest

Support Vector Machines (SVM)

K Nearest Neighbor (KNN)

WHAT IS CNN?

CNN stands for convolutional neural network. It is a deep learning algorithm that is widely used for analyzing visual data, such as images and videos. CNNs are highly effective in tasks such as image classification, object detection, facial recognition, and image segmentation.

WHY CNN?

Convolutional neural networks, are used for their ability to automatically extract relevant features from visual data, such as images. They're really good at finding patterns in pictures, like edges or colors, and they can recognize objects no matter where they are in the picture. They learn to understand pictures by looking at different layers of details, starting with simple things like lines and shapes, and then putting those details together to understand more complex objects.

HOW CNN WORKS

They work by breaking down the image into smaller parts and looking for patterns in those parts. The network learns to recognize different features like edges or textures. It then combines these features to understand more complex shapes or objects. CNNs use layers to process the image and make predictions about what it contains. The key components of a CNN include convolutional layers, pooling layers, and fully connected layers. They are trained by comparing their predictions to the correct answers and adjusting their internal settings to improve accuracy. This allows them to learn how to recognize and classify objects in images.

IMAGE CLASSIFICATION

Image classification is the task of categorizing or labeling images into different predefined classes or categories.

The goal is to develop and train a model that can automatically analyze the features of an image and assign it to one of several predefined classes.

LIBRARIES NEEDED

```
[ ]  pip install numpy
```

```
[ ]  pip install matplotlib
```

```
[ ]  pip install tensorflow
```

```
[ ]  pip install opencv-python
```

```
[ ]  import cv2 as cv
     import numpy as np
     import matplotlib.pyplot as plt
     from tensorflow.keras import datasets, layers, models
```

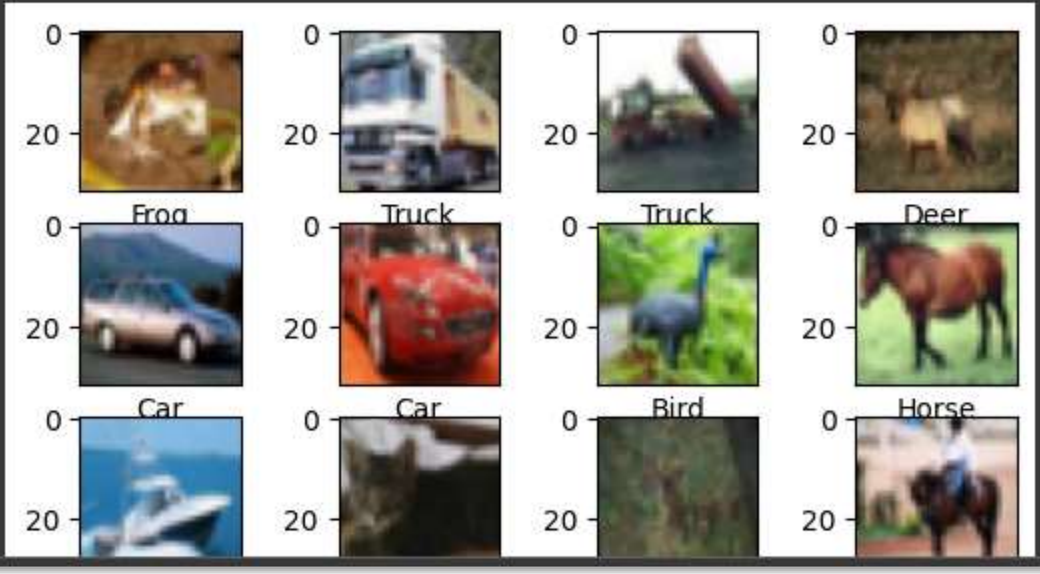
GET DATA FROM DATASET ALREADY INSIDE KERAS, CALL A LOAD FUNCTION THEN STORE THE DATA IN TRAINING AND TESTING. NEXT SCALE THE DATA DOWN SO THAT ALL VALUES ARE FROM 0-1 (MAKES IT EASIER TO WORK WITH)

```
[ ] (training_images, training_labels),(testing_images, testing_labels) = datasets.cifar10.load_data()

    training_images, testing_images = training_images / 255, testing_images / 255
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

The provided code snippet visualizes a set of training images with their respective class labels. It defines a list of class names representing different objects. The code then uses a loop to create a 4x4 grid of subplots. Within each subplot, an image from the training set is displayed using a binary color map

```
class_names = ['Plane', 'Car', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Boat', 'Truck']

for i in range(16):
  plt.subplot(4,4,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(training_images[i], cmap=plt.cm.binary)
  plt.xlabel(class_names[training_labels[i][0]])


plt.show()
```



Instead of training the neural network on the whole data, we are picking the first 100,000 training examples and the first 10,000 testing examples. Working with a smaller subset of the original dataset enables quicker experimentation or prototyping.

```
[ ]  training_images= training_images[:100000]
     training_labels = training_labels[:100000]
     testing_images = testing_images[:10000]
     testing_labels = testing_labels[:10000]
```

Conv2D is an important operation used in convolutional neural networks (CNNs) to find patterns in images. It works by sliding small filters across the image and multiplying the filter values with the corresponding image pixels.

Building the neural network

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape=(32,32,3)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(10, activation = 'softmax'))

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

model.fit(training_images, training_labels, epochs = 10, validation_data = (testing_images, testing_labels))
```

ReLU, short for Rectified Linear Unit, is a popular activation function used in neural networks, particularly in deep learning models. It is a simple mathematical function that returns the input value if it is positive, and zero if the input value is negative.

Building the neural network

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape=(32,32,3)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(10, activation = 'softmax'))

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

model.fit(training_images, training_labels, epochs = 10, validation_data = (testing_images, testing_labels))
```

Evaluation is done on the model based on the testing dataset. It takes the images and their corresponding labels as input.

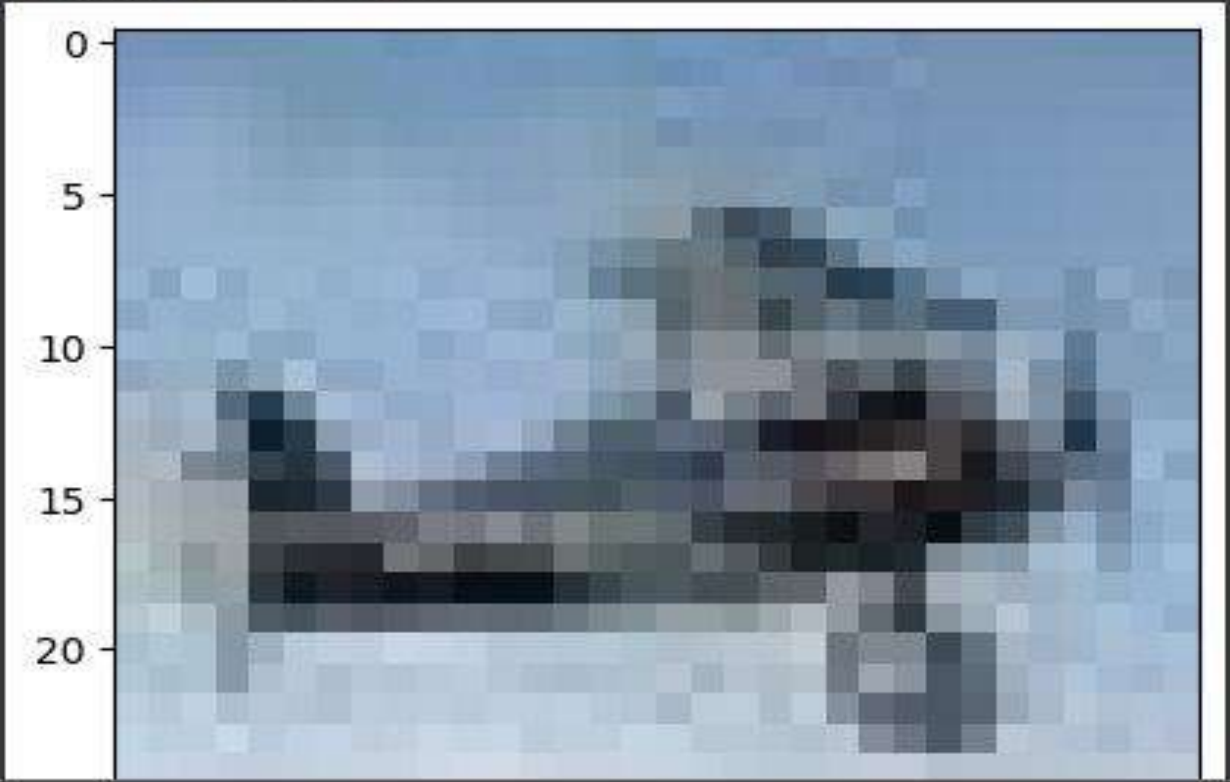
Test and Evaluate

```
[ ] loss, accuracy = model.evaluate(testing_images, testing_labels)
    print(f"Loss: {loss}")
    print(f"Accuracy: {accuracy}")

    model.save('image_classifier.model')
```

```
313/313 [==============================] - 3s 8ms/step - loss: 0.8905 - accuracy: 0.7149
Loss: 0.8904568552970886
Accuracy: 0.714900016784668
```

```
[ ] img = cv.imread('plane(1).jpg')
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

    plt.imshow(img, cmap=plt.cm.binary)

    prediction = model.predict(np.array([img]) / 255)
    index = np.argmax(prediction)
    print(f'prediction is {class_names[index]}')
```

```
1/1 [==============================] - 0s 27ms/step
prediction is Plane
```



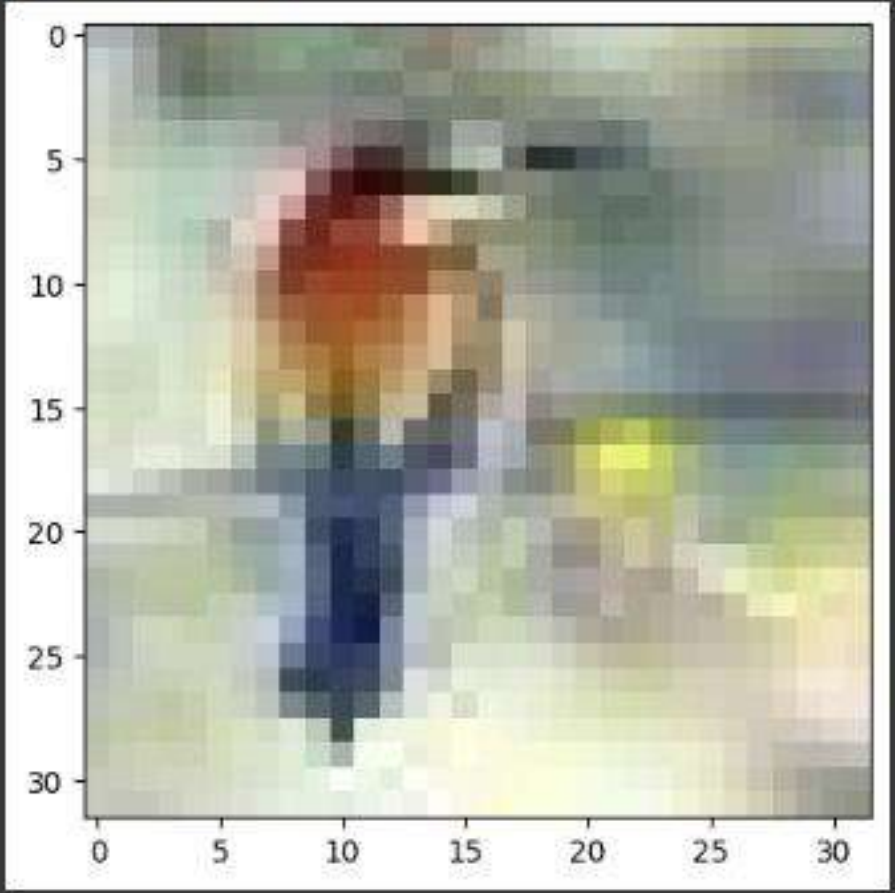
1s completed at 10:30 AM

cv.imread is a function in the OpenCV library used to read an image and store it in the variable “img”.

Converting an image from BGR to RGB is important for compatibility, accurate display, and proper interpretation of colors in the image because most deep-learning models use the RGB format.

```python
img = cv.imread('bird2.jpg')
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

plt.imshow(img, cmap=plt.cm.binary)

prediction = model.predict(np.array([img]) / 255)
index = np.argmax(prediction)
print(f'prediction is {class_names[index]}')
```

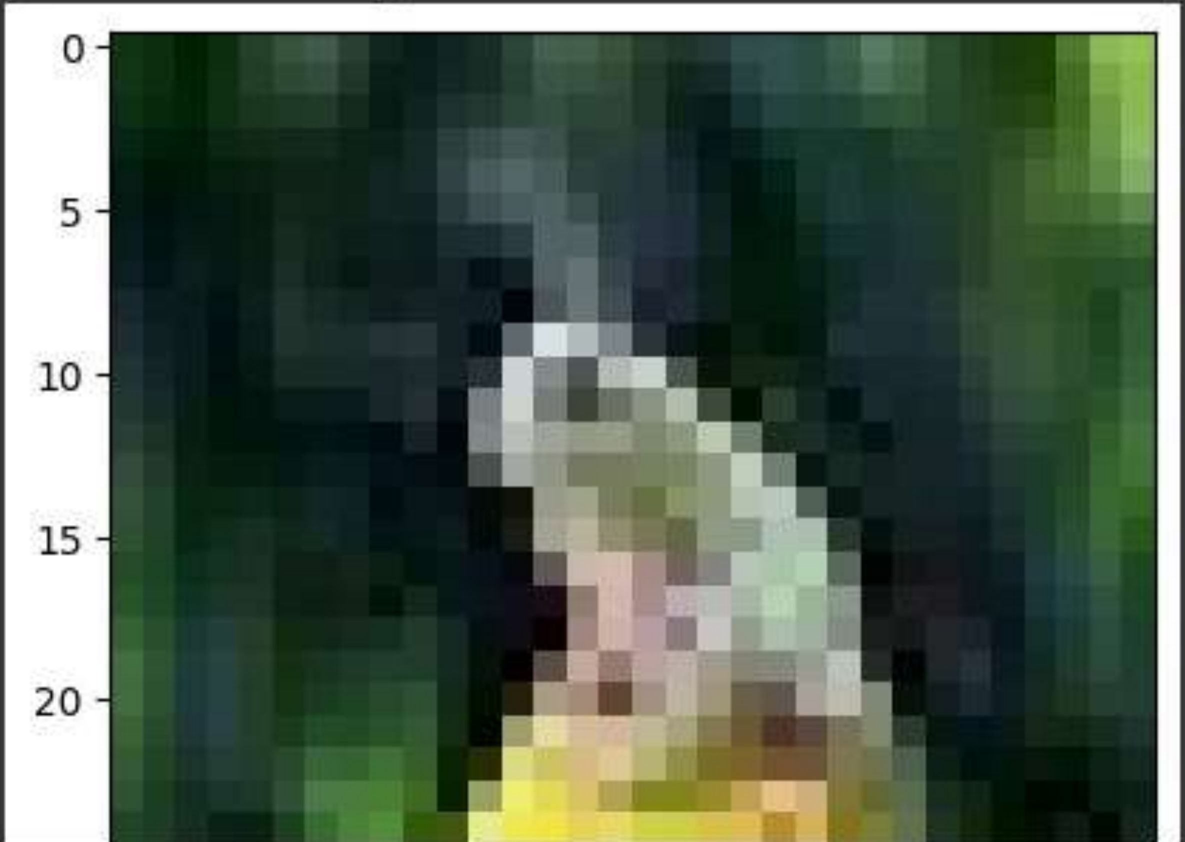
```
1/1 [==============================] - 0s 43ms/step
prediction is Bird
```



3s completed at 11:14 AM

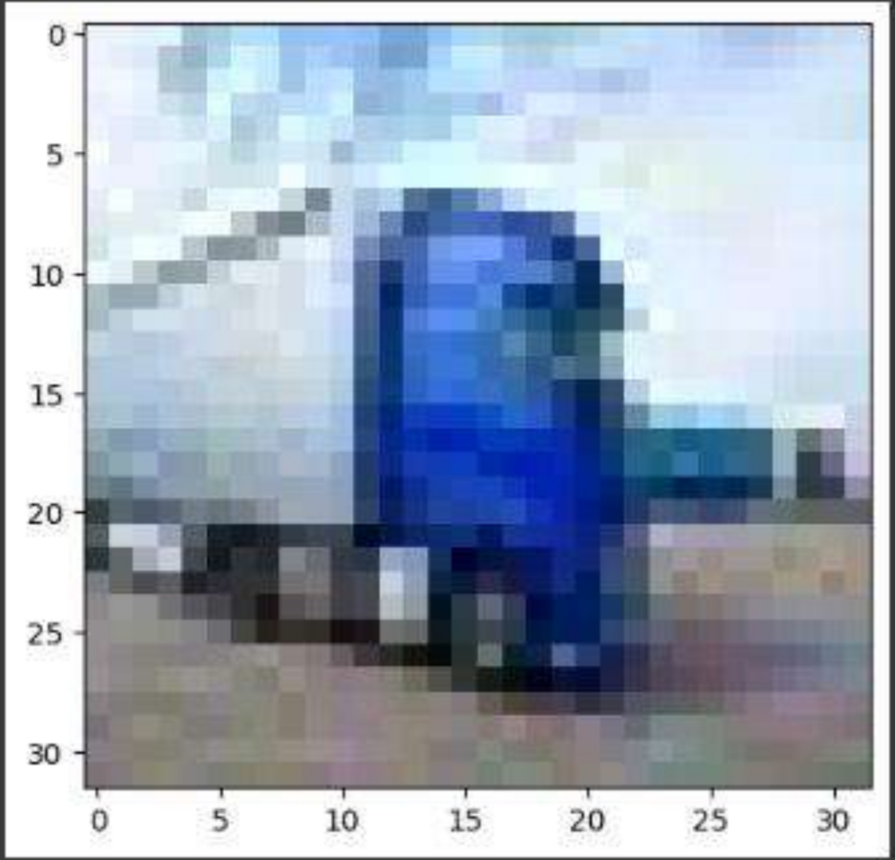
```
img = cv.imread('fr.jpg')
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

plt.imshow(img, cmap=plt.cm.binary)

prediction = model.predict(np.array([img]) / 255)
index = np.argmax(prediction)
print(f'prediction is {class_names[index]}')
```

```
1/1 [==============================] - 0s 59ms/step
prediction is Frog
```



```
img = cv.imread('tru.jpg')
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

plt.imshow(img, cmap=plt.cm.binary)

prediction = model.predict(np.array([img]) / 255)
index = np.argmax(prediction)
print(f'prediction is {class_names[index]}')
```

```
1/1 [==============================] - 0s 63ms/step
prediction is Truck
```



3s completed at 11:14 AM

RANDOM FOREST ALGORITHM

A random forest is an ensemble learning method, primarily used for classification and regression. It works by constructing multiple decision trees during training and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

WHY RANDOM FOREST IN VISUAL RECOGNITION?

Random forests are utilized for their robustness against overfitting, as they average out biases by combining the results of various trees. In the context of visual recognition, they are effective because they can handle high-dimensional data and identify the most informative features from a large dataset, such as pixels in images.

HOW DOES RANDOM FOREST WORK?

For visual recognition, random forest first trains on subsets of the image dataset with a random selection of features at each tree node, creating a "forest" of decision trees. During the recognition phase, each tree votes for a class, and the highest voted class is chosen. The model's ensemble approach enhances accuracy, as it reduces the risk of errors from individual trees.

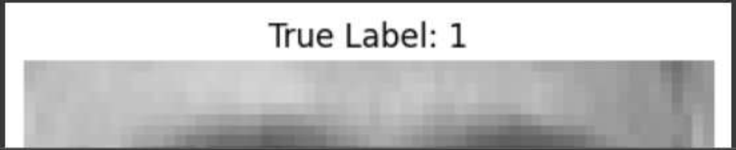
```
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load Olivetti Faces dataset
dataset = fetch_olivetti_faces()

# Extract features (X) and labels (y)
X = dataset.data
y = dataset.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Choose a random index from the test set
random_index = 21  # You can change this to any index you want to visualize

# Display the image and true label
plt.imshow(X_test[random_index].reshape((64, 64)), cmap='gray')
plt.title(f"True Label: {y_test[random_index]}")
plt.axis('off')
plt.show()
```

```python
# Display the image and true label
plt.imshow(X_test[random_index].reshape((64, 64)), cmap='gray')
plt.title(f"True Label: {y_test[random_index]}")
plt.axis('off')
plt.show()

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the selected image
predicted_label = rf_classifier.predict([X_test[random_index]])
print(f"Predicted Label: {predicted_label[0]}")

# Evaluate the performance
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Untitled0.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
plt.show()

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the selected image
predicted_label = rf_classifier.predict([X_test[random_index]])
print(f"Predicted Label: {predicted_label[0]}")

# Evaluate the performance
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```



Connected to Python 3 Google Compute Engine backend


```
Predicted Label: 1
Accuracy: 0.94
Classification Report:
              precision    recall  f1-score   support
```

4s

```
Predicted Label: 1
Accuracy: 0.94
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         1
           2       0.67      1.00      0.80         2
           3       1.00      1.00      1.00         4
           4       1.00      1.00      1.00         3
           5       0.75      1.00      0.86         3
           7       1.00      0.50      0.67         6
           8       1.00      1.00      1.00         2
           9       1.00      1.00      1.00         2
          10       0.67      1.00      0.80         2
          11       1.00      1.00      1.00         3
          12       1.00      1.00      1.00         2
          13       1.00      1.00      1.00         1
          14       1.00      1.00      1.00         3
          15       1.00      1.00      1.00         2
          17       0.75      1.00      0.86         3
          18       1.00      1.00      1.00         1
          19       1.00      1.00      1.00         1
          20       1.00      1.00      1.00         1
          21       1.00      1.00      1.00         1
          22       1.00      1.00      1.00         3
```

Bard | Search results - Google Drive | Untitled0.ipynb - Colaboratory | Search results - Google Drive | ChatGPT

colab.research.google.com/drive/1hCTYqj2JvZt6-Uqb1M2S-o4EooBBkru3#scrollTo=nksIU5-BWrgM

Gmail | YouTube | Maps | Download up4pc com... | How to Get Started W... | Document | favicon.ico Generator | Automate the Boring... | Free Online Video Do... | All Bookmarks

Untitled0.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Colab AI

4s

0	1.00	1.00	1.00	3
1	1.00	1.00	1.00	1
2	0.67	1.00	0.80	2
3	1.00	1.00	1.00	4
4	1.00	1.00	1.00	3
5	0.75	1.00	0.86	3
7	1.00	0.50	0.67	6
8	1.00	1.00	1.00	2
9	1.00	1.00	1.00	2
10	0.67	1.00	0.80	2
11	1.00	1.00	1.00	3
12	1.00	1.00	1.00	2
13	1.00	1.00	1.00	1
14	1.00	1.00	1.00	3
15	1.00	1.00	1.00	2
17	0.75	1.00	0.86	3
18	1.00	1.00	1.00	1
19	1.00	1.00	1.00	1
20	1.00	1.00	1.00	1
21	1.00	1.00	1.00	1
22	1.00	1.00	1.00	3
23	1.00	1.00	1.00	2
24	1.00	1.00	1.00	1
25	0.00	0.00	0.00	1
26	1.00	1.00	1.00	4
27	1.00	1.00	1.00	2
28	1.00	1.00	1.00	2

Connected to Python 3 Google Compute Engine backend

Search

10:25 AM
1/24/2024

VISUAL RECOGNITION SPECIFICS

In visual tasks, trees in the random forest decide on pixel values, colors, textures, or other image features. The random selection of features at each node is crucial because it forces the model to explore diverse feature combinations, enhancing the forest's capability to generalize from visual patterns.

SUPPORT VECTOR MACHINE ALGORITHM

Introduction to support vector machine in visual recognition: In machine learning, Support Vector Machine stands out as a powerful tool, particularly when it comes to tasks like image classification. Support vector machine learns to draw boundaries between different classes in a dataset. It finds the best way to draw a hyperplane ensuring the most accurate classification.

WHY SVM IN VISUAL RECOGNITION?

Effective in high dimensions: Images are often represented as high-dimensional data, with each pixel acting as a dimension. SVM excels in such scenarios, making it a perfect fit for visual data.

Versatility: SVM proves to be versatile. Its ability to handle complex datasets makes it indispensable in various applications.

Optimal Decision Boundaries: SVM finds the optimal decision boundaries, leading to precise and accurate classifications.

To demonstrate the capabilities of a support vector machine using an image classification model here's how the model works:

CATEGORY SELECTION

Image Classification Using SVM.ipynb

```python
Categories=['Cars','Ice cream cone','Cricket ball']
print("Type y to give categories or type n to go with classification of Cars,Ice Cream cone and Cricket ball");

while(True):
  check=input()
  if(check=='n' or check=='y'):
    break
  print("Please give a valid input (y/n)")
if(check=='y'):
  print("Enter How Many types of Images do you want to classify")
  n=int(input())
  Categories=[]
  print(f'please enter {n} names')
  for i in range(n):
    name=input()
    Categories.append(name)
  print(f"If not drive Please upload all the {n} category Images in google collab with the same names as given in categories")
```

```
Type y to give categories or type n to go with classification of Cars,Ice Cream cone and Cricket ball
y
Enter How Many types of Images do you want to classify
3
please enter 3 names
Cars
Ice cream cone
Cricket ball
If not drive Please upload all the 3 category images in google collab with the same names as given in categories
```

GIVE PATH OF THE FOLDER CONTAINING ALL CATEGORY IMAGES AND LOAD THE DATA.

Image Classification Using SVM.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text Reconnect Colab AI

```
flat_data_arr=[]
target_arr=[]
datadir = '/content/drive/MyDrive/Image Classification Using SVM'
from matplotlib.image import imread
from skimage.transform import resize

for i in Categories:
  print(f'loading... category : {i}')
  path=os.path.join(datadir,i)
  for img in os.listdir(path):
    img_array=imread(os.path.join(path,img))
    img_resized=resize(img_array,(150,150,3))
    flat_data_arr.append(img_resized.flatten())
    target_arr.append(Categories.index(i))
  print(f'loaded category:{i} successfully')
flat_data=np.array(flat_data_arr)
target=np.array(target_arr)
df=pd.DataFrame(flat_data)
df['Target']=target
df
```

```
loading... category : Cars
loaded category:Cars successfully
loading... category : Cricket ball
loaded category:Cricket ball successfully
loading... category : Ice cream cone
loaded category:Ice cream cone successfully
```

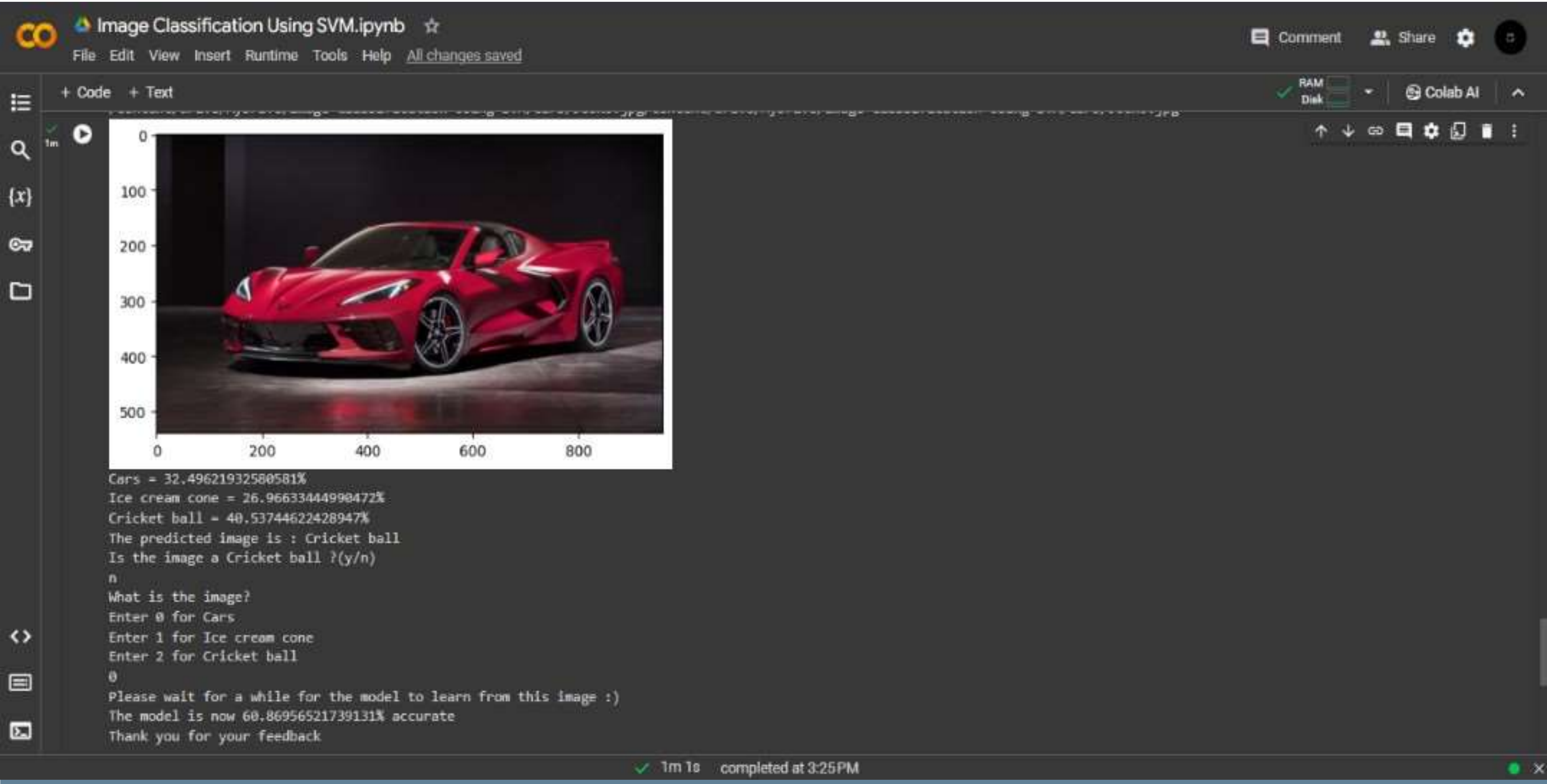
TRAINING THE DATA USING SUPPORT VECTOR CLASSIFICATION MODEL

Image Classification Using SVM.ipynb

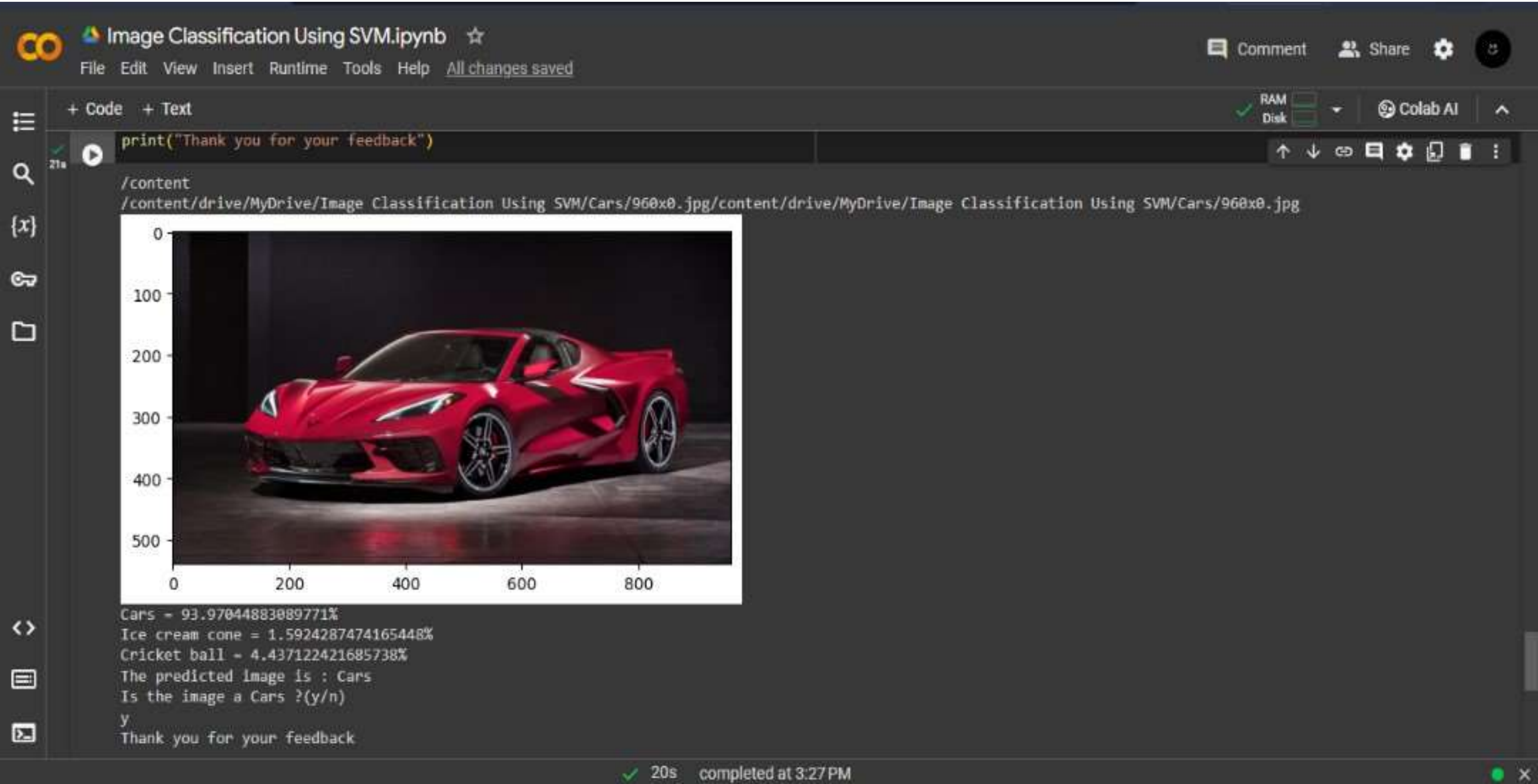
```
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn import svm

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(flat_data, target, test_size=0.2, random_state=42)

# Feature Scaling
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Define the parameter grid
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'poly', 'linear']}

# Create the SVM model
svc = svm.SVC(probability=True)

# Create the GridSearchCV model with StratifiedKFold
cv = StratifiedKFold(n_splits=5)
model = GridSearchCV(svc, param_grid, cv=cv)

# Train the model
print("The training of the model is started, please wait for a while as it may take a few minutes to complete")
model.fit(x_train_scaled, y_train)

# Print the results
print('The Model is trained well with the given images')
print('Best parameters:', model.best_params_)
```

```
The training of the model is started, please wait for a while as it may take a few minutes to complete
The Model is trained well with the given images
Best parameters: {'C': 0.1, 'gamma': 1, 'kernel': 'linear'}
```

TESTING THE MODEL



TESTING THE MODEL 2



K NEAREST NEIGHBOUR

What Is KNN algorithm: the k-nearest neighbors algorithm, also known as KNN or k-nn, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

HOW KNN ALGORITHM CAN BE USED IN VISUAL RECOGNITION

K-nearest neighbours (KNN) is one of the popular algorithms used in face recognition. It's a supervised learning approach that is used for classification purposes. It operates by locating the nearest neighbours of a data point and classifying it according to the majority of its neighbours.

WHY KNN IS USED

The KNN algorithm can compete with the most accurate models because it makes highly accurate predictions. Therefore, you can use the KNN algorithm for applications that require high accuracy but that do not require a human-readable model.

CHALLENGES OF USING KNN

KNN has some drawbacks and challenges, such as computational expense, slow speed, memory and storage distance metric, and susceptibility to the curve of dimensionality.

```python
import numpy as np #library for numerical operations.
from collections import Counter


def euclidean_distance(x1, x2): #closeness between x1 & x2
    distance = np.sqrt(np.sum((x1 - x2) ** 2))
    return distance


class KNN:
    def __init__(self, k=5): #classes of the first 5 closest points
        self.k = k

    def fit(self, X, y): #A method to store the training data
        self.X_train = X
        self.y_train = y

    def predict(self, X): # A method to make predictions for a given set of input data X.
        predictions = [self._predict(x) for x in X]
        return predictions

    def _predict(self, x): #A helper method that calculates distances,
        # finds the k-nearest neighbors, and performs majority voting to make a prediction.
        # compute the distance
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]

        # get the closest k
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
```

```python
class KNN:
    def __init__(self, k=5): #classes of the first 5 closest points
        self.k = k

    def fit(self, X, y): #A method to store the training data
        self.X_train = X
        self.y_train = y

    def predict(self, X): # A method to make predictions for a given set of input data X.
        predictions = [self._predict(x) for x in X]
        return predictions

    def _predict(self, x): #A helper method that calculates distances,
        # finds the k-nearest neighbors, and performs majority voting to make a prediction.
        # compute the distance
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]

        # get the closest k
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        # majority voye
        most_common = Counter(k_nearest_labels).most_common()
        return most_common[0][0]
```


X2
Category A:3 neighbors
Category B:2 neighbors
Category B
New Data
point
Category A
X1