```
# Import the various libraries:

from tensorflow.keras.datasets import cifar10
import numpy as np

# Load the Dataset from the in-bulit library

(X_train, y_train), (X_test, y_test) = cifar10.load_data() # this syntax fe

# Print the Dataset

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
(50000, 32, 32, 3)
(50000, 1)
(10000, 32, 32, 3)
(10000, 1)
```

```
# Normalize the Dataset (This is part of the preprocessing stage the data g

X_train_norm = X_train.astype('float32')/255.0
X_test_norm = X_test.astype('float32') / 255.0

print("The Normalize Dataset Shape for:", X_train_norm.shape)  # this synta
print("The Normalize Dataset Shape for:", X_test_norm.shape)

# Sample dataset for training and testing. Using the full dataset(50,000) c

training_values = 5000 # this syntax reflects the proportion of the dataset
testing_values = 2000


# Reduce Images

X_train_small = X_train_norm[:training_values]
X_test_small = X_test_norm[:testing_values]


# Reduce labels

y_train_small = y_train[:training_values] # the dataset in the label must a
y_test_small  = y_test[:testing_values]

```

```
# Flatten the Images for Random Forest - this ensures that the images in th

X_train_flat = X_train_small.reshape(training_values,-1)
X_test_flat = X_test_small.reshape(testing_values, -1)

print(X_train_flat.shape, y_train_small)
print(X_test_flat.shape, y_test_small.shape)
```

```
The Normalize Dataset Shape for: (50000, 32, 32, 3)
The Normalize Dataset Shape for: (10000, 32, 32, 3)
(5000, 3072) [[6]
 [9]
 [9]
 ...
 [5]
 [4]
 [6]]
(2000, 3072) (2000, 1)
```

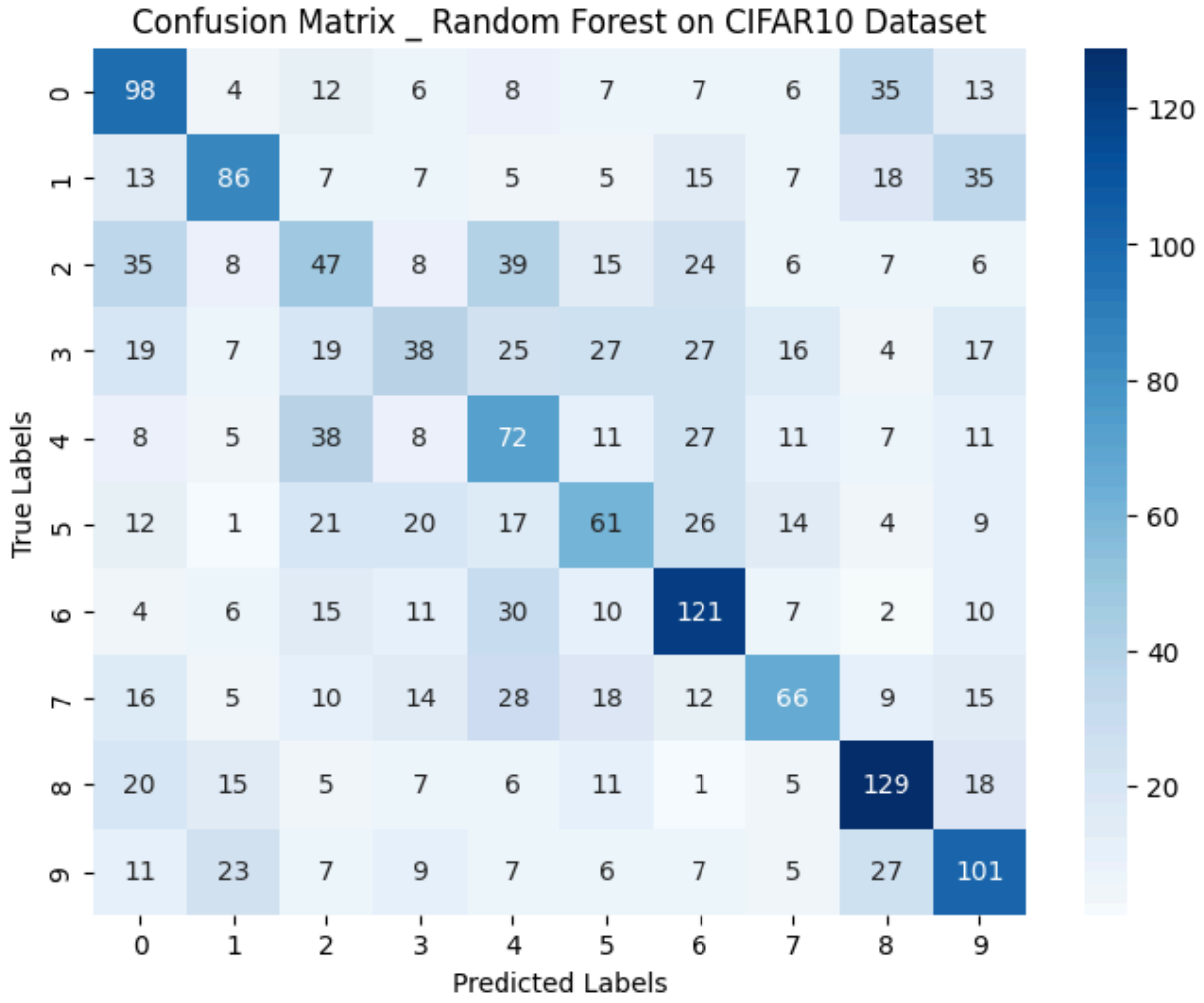
```
# The Model ( Trian the Model to ascertain if it performance on the dataset

# Import the Random Forest Model

from sklearn.ensemble import RandomForestClassifier

# this is the model used for training
random_forex_model = RandomForestClassifier(n_estimators=100, random_state=

# train the model
random_forex_model.fit(X_train_flat, y_train_small.ravel())

print("Traing Complete")
```

```
Traing Complete
```

```
# Predict the Model

y_pred = random_forex_model.predict(X_test_flat)

#Evaluate the Model

from sklearn.metrics import accuracy_score, classification_report

accuracy = accuracy_score(y_test_small, y_pred)

print("Random Forest Accuracy Score:", accuracy)
print("Classification Report:\n", classification_report(y_test_small, y_pre

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

```
17
18 confix_matrix = confusion_matrix(y_test_small, y_pred)
19
20 plt.figure(figsize=(8,6))
21 sns.heatmap(confix_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=r
22 plt.xlabel("Predicted Labels")
23 plt.ylabel("True Labels")
24 plt.title("Confusion Matrix _ Random Forest on CIFAR10 Dataset")
25 plt.show()
```

```
Random Forest Accuracy Score: 0.4095
Classification Report:
              precision    recall  f1-score   support

           0       0.42      0.50      0.45       196
           1       0.54      0.43      0.48       198
           2       0.26      0.24      0.25       195
           3       0.30      0.19      0.23       199
           4       0.30      0.36      0.33       198
           5       0.36      0.33      0.34       185
           6       0.45      0.56      0.50       216
           7       0.46      0.34      0.39       193
           8       0.53      0.59      0.56       217
           9       0.43      0.50      0.46       203

    accuracy                           0.41      2000
   macro avg       0.40      0.41      0.40      2000
weighted avg       0.41      0.41      0.40      2000
```




```
1 from sklearn.model_selection import GridSearchCV
2
3 #Fining the Paremeters
```

```
param_grid = {
    'n_estimators': [50, 100,150],        # I used a small range of trees h
    'max_depth': [None, 10, 20, 30],      # shallow vs deeper
    'min_samples_split': [2, 5, 10],      # small options
    'min_samples_leaf': [1]               # fixed to reduce combination
}

# Set - up the Base
random_forex_model = RandomForestClassifier(random_state=42, n_jobs=-1)

grid_search = GridSearchCV(           # this is integration of the GridSear
    estimator=random_forex_model,
    param_grid=param_grid,
    cv=3,                             # 3 fold cross validation
    n_jobs=-1,                        # use all the cpu cores
    verbose=2)                        # show progress

grid_search.fit(X_train_flat, y_train_small.ravel())

#Print the best GridSearch Result:

print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
Best Parameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 1
Best Score: 0.40140303271878636
```

```
# Train the Random Forest Model based on the GirdSearchCV Results.
## GridSearchCV does not train my model, instead it observed the model
## based on the paremters that were established and provided me with the be

# Retrive the best paremeters from the GridSearchCV

best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Train the final model

final_model = RandomForestClassifier(
    n_estimators = best_params['n_estimators'],
    max_depth= best_params['max_depth'],
    min_samples_split = best_params['min_samples_split'],
    min_samples_leaf=best_params['min_samples_leaf'],
    random_state =42,
    n_jobs = -1,
    verbose = 0 # the output from my RF indicated the logs of the training
 )

```

```
22
23 # # Fit the final model on the full reduce text/dataset
24
25 final_model.fit(X_train_flat, y_train_small.ravel())
26
27 print("Final Model Training Complete")
28
```

```
Best Parameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 1
Final Model Training Complete
```

```
1 # Evaluate the Model
2
3 y_pred_final = final_model.predict(X_test_flat)
4
5 accuracy = accuracy_score(y_test_small, y_pred_final)
6
7 print("Random Forest Accuracy Score:", accuracy)
8 print("Classification Report:\n", classification_report(y_test_small, y_pre
```

```
Random Forest Accuracy Score: 0.403
Classification Report:
              precision    recall  f1-score   support

           0       0.39      0.46      0.42       196
           1       0.56      0.41      0.48       198
           2       0.24      0.21      0.22       195
           3       0.35      0.18      0.24       199
           4       0.30      0.34      0.32       198
           5       0.30      0.31      0.31       185
           6       0.44      0.53      0.48       216
           7       0.45      0.35      0.39       193
           8       0.53      0.62      0.57       217
           9       0.44      0.56      0.49       203

    accuracy                           0.40      2000
   macro avg       0.40      0.40      0.39      2000
weighted avg       0.40      0.40      0.40      2000
```

This test is based on the above evaluation that was ran for the final RF Model

Results & Model Interpretation:

The final tuned Random Forest model achieved an accuracy of approximately 40.3% on the CIFAR-10 test set. This level of performance is expected for a classical machine learning algorithm applied to image data. CIFAR-10 contains complex, high-dimensional images (32×32×3 pixels), and Random Forests treat each pixel as an independent feature

without understanding spatial patterns or textures. As a result, the model is limited in its ability to capture visual structure compared to convolutional neural networks (CNNs).

Across individual classes, the model performed better on objects with more distinct shapes and colors, such as automobiles, trucks, ships, and frogs. These classes achieved relatively higher precision and recall. In contrast, classes such as cats, dogs, deer, and birds had lower performance because they share similar shapes and color patterns, making them harder for a pixel-based model to separate.

Overall, the results demonstrate that while Random Forests can provide some classification ability, they are not well-suited for complex image tasks. The experiment highlights the importance of using specialized deep learning architectures for image recognition. Nevertheless, the model behaves consistently and validates the effectiveness of preprocessing, parameter tuning, and evaluation steps in the machine learning workflow.

```
 1
 2 from PIL import Image
 3
 4 # Prediction on new image
 5
 6 cifar10_classes = [
 7     "airplane", "automobile", "bird", "cat", "deer",
 8     "dog", "frog", "horse", "ship", "truck"
 9 ]
10
11 def predict_new_image(image_path, model):
12     img = Image.open(image_path).convert('RGB') # load the image and conver
13     img = img.resize((32, 32)) # this syntax resize the image as per the re
14     img_array = np.array(img) # convert the image to an array
15     img_array = img_array.astype('float32') / 255.0 # normalize the image
16     img_flat = img_array.reshape(1, -1) # flatten the image leads to equal
17     predicted_model = model.predict(img_flat) # predict the image
18     predicted_class = cifar10_classes[predicted_model[0]] # convert predict
19     return predicted_class
```

```
1 from google.colab import files # this syntax allows me to upload a picture
2 uploaded = files.upload()
3
4 filename = list(uploaded.keys())[0]
5 image_path = filename
6
7 result = predict_new_image(image_path, final_model)
8 print("Predicted Class:", result)
9
```

```
Choose Files s4vstarget.png
s4vstarget.png(image/png) - 183335 bytes, last modified: 8/4/2025 - 100% done
Saving s4vstarget.png to s4vstarget.png
Predicted Class: airplane
```

```
from sklearn.svm import SVC # this library is call the Support Vector Machi

svm_model = SVC(kernel='linear')

print("Training Model SVC")
svm_model.fit(X_train_flat, y_train_small.ravel())
print("Training Complete")
```

```
Training Model SVC
Training Complete
```

```
# Predict the Model

y_pred_svm = svm_model.predict(X_test_flat)

# Print the Reports - Accuracy, SVM & Classification Reports

svm_model_accuracy = accuracy_score(y_test_small, y_pred_svm)

print("SVM Accuracy Score:", svm_model_accuracy)
print("Classification Report:\n", classification_report(y_test_small, y_pre

print("Random Forest Accuracy Score:", accuracy)
print("Classification Report:\n", classification_report(y_test_small, y_pre
```

```
SVM Accuracy Score: 0.3
Classification Report:
              precision    recall  f1-score   support

           0       0.27      0.39      0.32       196
           1       0.39      0.38      0.39       198
           2       0.24      0.31      0.27       195
           3       0.20      0.21      0.20       199
           4       0.25      0.26      0.25       198
           5       0.16      0.15      0.15       185
           6       0.36      0.27      0.31       216
           7       0.32      0.27      0.29       193
           8       0.45      0.46      0.45       217
           9       0.40      0.28      0.33       203

    accuracy                           0.30      2000
   macro avg       0.30      0.30      0.30      2000
weighted avg       0.31      0.30      0.30      2000

Random Forest Accuracy Score: 0.403
Classification Report:
              precision    recall  f1-score   support
```

```
              0       0.27      0.39      0.32       196
              1       0.39      0.38      0.39       198
              2       0.24      0.31      0.27       195
              3       0.20      0.21      0.20       199
              4       0.25      0.26      0.25       198
              5       0.16      0.15      0.15       185
              6       0.36      0.27      0.31       216
              7       0.32      0.27      0.29       193
              8       0.45      0.46      0.45       217
              9       0.40      0.28      0.33       203

       accuracy                           0.30      2000
      macro avg       0.30      0.30      0.30      2000
   weighted avg       0.31      0.30      0.30      2000
```

Deployment Strategy:

Deploying an image classification model involves preparing the trained system so it can be used reliably in real-world applications. Although this project uses a classical machine learning approach, the deployment workflow follows the same principles used in modern ML systems. The strategy below outlines how the Random Forest model can be operationalized in a production environment.

1. Model Packaging

After training, the final Random Forest model should be saved using a serialization format such as pickle or joblib. This allows the model to be reloaded without retraining. The preprocessing steps—image resizing, normalization, and flattening—must also be packaged with the model so the same transformations are applied consistently during inference.

2. Building an Inference Pipeline

A lightweight prediction pipeline is needed to accept new images and return output: Receive the uploaded image (e.g., JPG/PNG). Convert it to RGB, resize it to 32×32, and normalize pixel values. Flatten the image into a 1D array of 3,072 features. Pass the processed array to the saved model for prediction. Map the numeric prediction to the corresponding CIFAR-10 class label. This ensures the model handles new images in the exact same manner as during training.

3. Deployment Options

There are multiple ways the model could be deployed depending on system requirements: Web API Deployment : The model can be hosted behind a simple REST API using tools such as: • Flask • FastAPI • Django REST Framework Clients can send images to the API endpoint, and the server returns the predicted class. Cloud Deployment: The

model can be deployed using cloud services such as: • AWS Lambda • Google Cloud Functions • Azure App Service These services are automatically scaled based on user demand and simplify maintenance. Local or Desktop Application: For offline use, the model can be integrated into a: • desktop application • mobile app • command-line tool