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

```
Mounted at /content/drive
```

```
from google.colab import files
files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json':
```

```
!rm -r ~/.kaggle
!mkdir ~/.kaggle
!mv ./kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
rm: cannot remove '/root/.kaggle': No such file or directory
```

```
! kaggle datasets list
```

```
ref                                                            title                                              size  lastUpdated
-------------------------------------------------------------  -------------------------------------------------  -----  --------------
carlmcbrideellis/llm-7-prompt-training-dataset                 LLM: 7 prompt training dataset                      41MB  2023-11-15 07:
thedrcat/daigt-proper-train-dataset                            DAIGT Proper Train Dataset                         119MB  2023-11-05 14:
thedrcat/daigt-v2-train-dataset                                DAIGT V2 Train Dataset                              29MB  2023-11-16 01:
joebeachcapital/30000-spotify-songs                            30000 Spotify Songs                                  3MB  2023-11-01 06:
iamsouravbanerjee/customer-shopping-trends-dataset             Customer Shopping Trends Dataset                   146KB  2023-10-05 06:
nelgiriyewithana/world-educational-data                        World Educational Data                               9KB  2023-11-04 06:
prasad22/healthcare-dataset                                    Healthcare Dataset                                 483KB  2023-10-31
ddosad/auto-sales-data                                         Automobile Sales data                               79KB  2023-11-18 12:
dillonmyrick/high-school-student-performance-and-demographics  High School Student Performance & Demographics      24KB  2023-11-10 01:
jacksondivakarr/online-shopping-dataset                        Online Shopping Dataset                              5MB  2023-11-
everydaycodings/job-opportunity-dataset                        Job Opportunities Dataset                           95KB  2023-11-20 08:
bwandowando/1-5-million-netflix-google-store-reviews           1.5 Million Netflix Google Store Reviews           114MB  2023-11-17
jdaustralia/icc-cwc23-all-innings-cleaned                      ICC Cricket World Cup CWC23 All innings             28KB  2023-11-21 04:
anshtanwar/top-200-trending-books-with-reviews                 Top 100 Bestselling Book Reviews on Amazon         422KB  2023-11-09 06:
joebeachcapital/coronavirus-covid-19-cases-daily-updates       Coronavirus (COVID-19) Cases (Daily Updates)        14MB  2023-11-23 23:
mauryansshivam/list-of-internet-products-of-top-tech-companies List of Internet Products of Top Tech Companies      9KB  2023-11-15 19:
vikramrn/icc-mens-odi-world-cup-wc-2023                        ICC mens cricket odi world cup wc 2023 - batting    16KB  2023-11-20 12:
samybaladram/databank-world-development-indicators             Global Socio-Economic & Demographic Insights         2MB  2023-11-06 05
shudhanshusingh/real-estate-properties-dataset                 Real Estate Properties Dataset                     882KB  2023-11-18 20:
samyakb/student-stress-factors                                 Student stress factors                             887B  2023-11-02 12:
```

```
# wheat = olyadgetch/wheat-leaf-dataset 2gb
#Maize = smaranjitghose/corn-or-maize-leaf-disease-dataset 169mb
#rice2 = maimunulkjisan/rice-leaf-dataset-from-mendeley-data 205mb
#sugarcane = prabhakaransoundar/sugarcane-disease-dataset 2gb
#cotton = seroshkarim/cotton-leaf-disease-dataset 190mb
dataset_name = 'prabhakaransoundar/sugarcane-disease-dataset'
zip_name = dataset_name.split('/')[-1]

!kaggle datasets download -d {dataset_name}

```

```
Downloading sugarcane-disease-dataset.zip to /content
100% 1.99G/2.00G [00:25<00:00, 146MB/s]
100% 2.00G/2.00G [00:25<00:00, 83.1MB/s]
```

```
!unzip -q ./sugarcane-disease-dataset.zip -d /content/drive/MyDrive/GROUP-4/EDI/Dataset
```

```
import numpy as np
import time

import PIL.Image as Image
import matplotlib.pylab as plt

import tensorflow as tf
import tensorflow_hub as hub

import os
```

```
import os
import shutil
from sklearn.model_selection import train_test_split
from tqdm import tqdm

import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix
import seaborn as sns

```

```
# directory_to_delete = "/content/drive/MyDrive/GROUP-4/EDI/Dataset/cotton"

#  # Use shutil.rmtree() to delete the directory and its contents
# shutil.rmtree(directory_to_delete)

# print(f"Directory '{directory_to_delete}' has been deleted.")

```

```
Directory '/content/drive/MyDrive/GROUP-4/EDI/Dataset/cotton' has been deleted.
```

```
# Define the source dataset directory
source_dataset_directory = "/content/drive/MyDrive/GROUP-4/EDI/Dataset/sugarcane RA"

# Define the output directory where the split dataset will be saved
output_directory = "/content/drive/MyDrive/GROUP-4/EDI/split_dataset/suger"

# Get the list of all classes (subdirectories) in the source directory
classes = [d for d in os.listdir(source_dataset_directory) if os.path.isdir(os.path.join(source_dataset_directory, d)) and d not in ['tr

# Create the output directory if it doesn't exist
os.makedirs(output_directory, exist_ok=True)

# Iterate over each class
for class_name in classes:
    # Define the path to the class directory in the source dataset
    class_source_directory = os.path.join(source_dataset_directory, class_name)

    # Get the list of all files (images) in the class directory
    all_files = [f for f in os.listdir(class_source_directory) if os.path.isfile(os.path.join(class_source_directory, f))]

    # Check if the dataset for the current class is not empty
    if not all_files:
        print(f"Warning: The dataset for class {class_name} is empty.")
    else:
        # Split the dataset for the current class into train and validation sets
        train_files, val_files = train_test_split(all_files, test_size=0.3, random_state=42)

        # Function to copy files from source to destination
        def copy_files(file_list, src_dir, dest_dir):
            for file in tqdm(file_list, desc=f"Copying files for class {class_name}"):
                src_path = os.path.join(src_dir, file)
                dest_path = os.path.join(dest_dir, file)
                shutil.copy(src_path, dest_path)

        # Copy train files to the train folder
        train_directory = os.path.join(output_directory, 'train', class_name)
        os.makedirs(train_directory, exist_ok=True)
        copy_files(train_files, class_source_directory, train_directory)

        # Copy validation files to the val folder
        val_directory = os.path.join(output_directory, 'val', class_name)
        os.makedirs(val_directory, exist_ok=True)
        copy_files(val_files, class_source_directory, val_directory)

```

```
Copying files for class Bacterial Blight: 100%|██████████| 70/70 [00:03<00:00, 17.88it/s]
Copying files for class Bacterial Blight: 100%|██████████| 30/30 [00:01<00:00, 21.73it/s]
Copying files for class Healthy: 100%|██████████| 70/70 [00:07<00:00,  9.44it/s]
Copying files for class Healthy: 100%|██████████| 30/30 [00:02<00:00, 13.68it/s]
Copying files for class Red Rot: 100%|██████████| 70/70 [00:06<00:00, 11.50it/s]
Copying files for class Red Rot: 100%|██████████| 30/30 [00:03<00:00,  9.03it/s]
```

```
batch_size = 32
img_height = 224
img_width = 224
data_root = '/content/drive/MyDrive/GROUP-4/EDI/split_dataset/maize/train'

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  str(data_root),
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 2560 files belonging to 4 classes.
Using 2048 files for training.
```

```
class_names = np.array(train_ds.class_names)
print('class names for predictions :',class_names)
```

```
class names for predictions : ['Blight' 'Common_Rust' 'Gray_Leaf_Spot' 'Healthy']
```

```
normalization_layer = tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
```

```
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 224, 224, 3)
(32,)
```

```
feature_extractor_model = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
feature_extractor_layer = hub.KerasLayer(
    feature_extractor_model, input_shape=(224, 224, 3), trainable=False)
```

```
num_classes = len(class_names)

model = tf.keras.Sequential([
  feature_extractor_layer,
  tf.keras.layers.Dense(num_classes)
])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer_2 (KerasLayer)  (None, 1280)              2257984

 dense_2 (Dense)             (None, 4)                 5124

=================================================================
Total params: 2263108 (8.63 MB)
Trainable params: 5124 (20.02 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

```
image_batch.shape
```

```
TensorShape([32, 224, 224, 3])
```

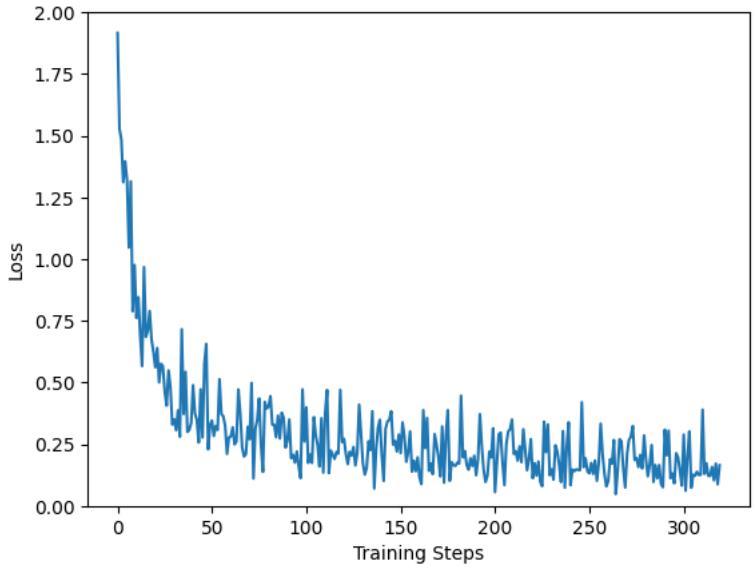
```
model.compile(
  optimizer=tf.keras.optimizers.Adam(),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['acc'])
```

```
class CollectBatchStats(tf.keras.callbacks.Callback):
  def __init__(self):
    self.batch_losses = []
    self.batch_acc = []

  def on_train_batch_end(self, batch, logs=None):
    self.batch_losses.append(logs['loss'])
    self.batch_acc.append(logs['acc'])
    self.model.reset_metrics()

batch_stats_callback = CollectBatchStats()

history = model.fit(train_ds, epochs=5,
                    callbacks=[batch_stats_callback])
```

```
Epoch 1/5
64/64 [==============================] - 176s 3s/step - loss: 0.0000e+00 - acc: 0.0000e+00
Epoch 2/5
64/64 [==============================] - 3s 43ms/step - loss: 0.0000e+00 - acc: 0.0000e+00
Epoch 3/5
64/64 [==============================] - 3s 41ms/step - loss: 0.0000e+00 - acc: 0.0000e+00
Epoch 4/5
64/64 [==============================] - 3s 41ms/step - loss: 0.0000e+00 - acc: 0.0000e+00
Epoch 5/5
64/64 [==============================] - 3s 41ms/step - loss: 0.0000e+00 - acc: 0.0000e+00
```

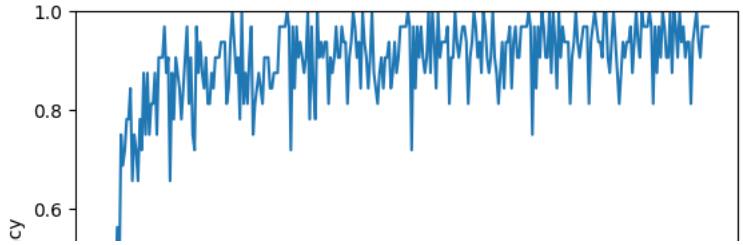
```
plt.figure()
plt.ylabel("Loss")
plt.xlabel("Training Steps")
plt.ylim([0,2])
plt.plot(batch_stats_callback.batch_losses)
```

```
[<matplotlib.lines.Line2D at 0x7c4b0bc6ea40>]
```

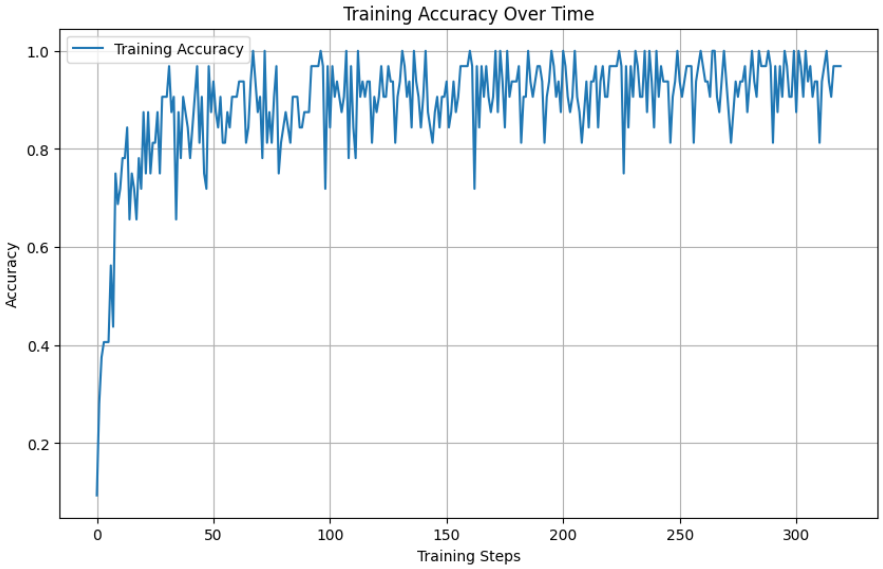



```
plt.figure()
plt.ylabel("Accuracy")
plt.xlabel("Training Steps")
plt.ylim([0,1])
plt.plot(batch_stats_callback.batch_acc)
```

```
[<matplotlib.lines.Line2D at 0x7c4b1df02e30>]
```




```
import matplotlib.pyplot as plt

# Access accuracy values collected during training
batch_acc = batch_stats_callback.batch_acc

# Plot the accuracy matrix
plt.figure(figsize=(10, 6))
plt.plot(batch_acc, label='Training Accuracy')
plt.xlabel('Training Steps')
plt.ylabel('Accuracy')
plt.title('Training Accuracy Over Time')
plt.legend()
plt.grid(True)
plt.show()

```



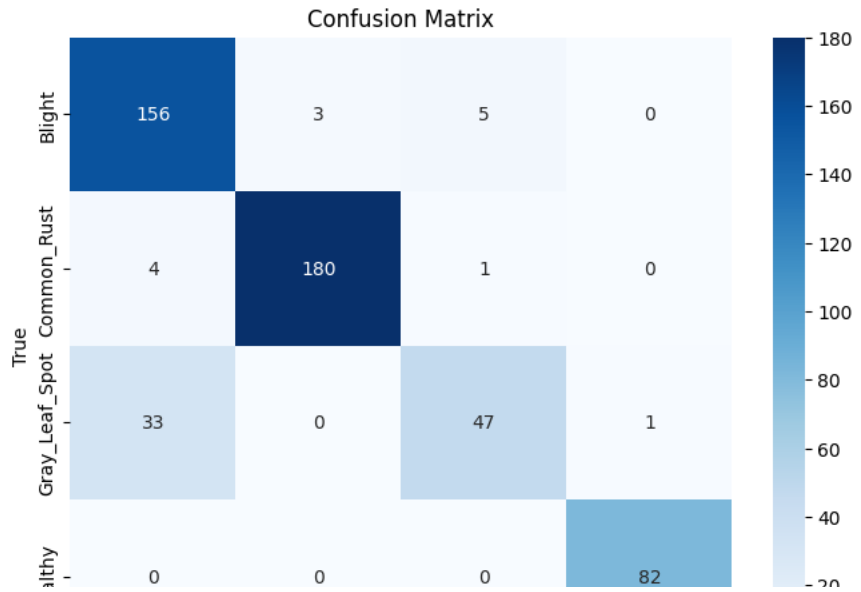
```
# Load and preprocess validation dataset
data_root_val = '/content/drive/MyDrive/GROUP-4/EDI/split_dataset/maize/val'

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  str(data_root),
  validation_split=0.2,
  subset="validation",  # Use validation subset
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# Evaluate the model on the validation dataset
val_loss, val_accuracy = model.evaluate(val_ds)

print(f"Validation Loss: {val_loss * 100:.2f}%")
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")

```

```
Found 2560 files belonging to 4 classes.
Using 512 files for validation.
16/16 [==============================] - 45s 1s/step - loss: 0.2234 - acc: 0.9082
Validation Loss: 22.34%
Validation Accuracy: 90.82%
```

```
true_labels = []
predicted_labels = []

for images, labels in val_ds:
    true_labels.extend(labels.numpy())  # Collect true labels
    predictions = model.predict(images)
    predicted_labels.extend(tf.argmax(predictions, axis=1).numpy())  # Collect predicted labels

```

```
WARNING:tensorflow:5 out of the last 11 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7c4b1bad0d30> tri
1/1 [==============================] - 1s 761ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 35ms/step
```

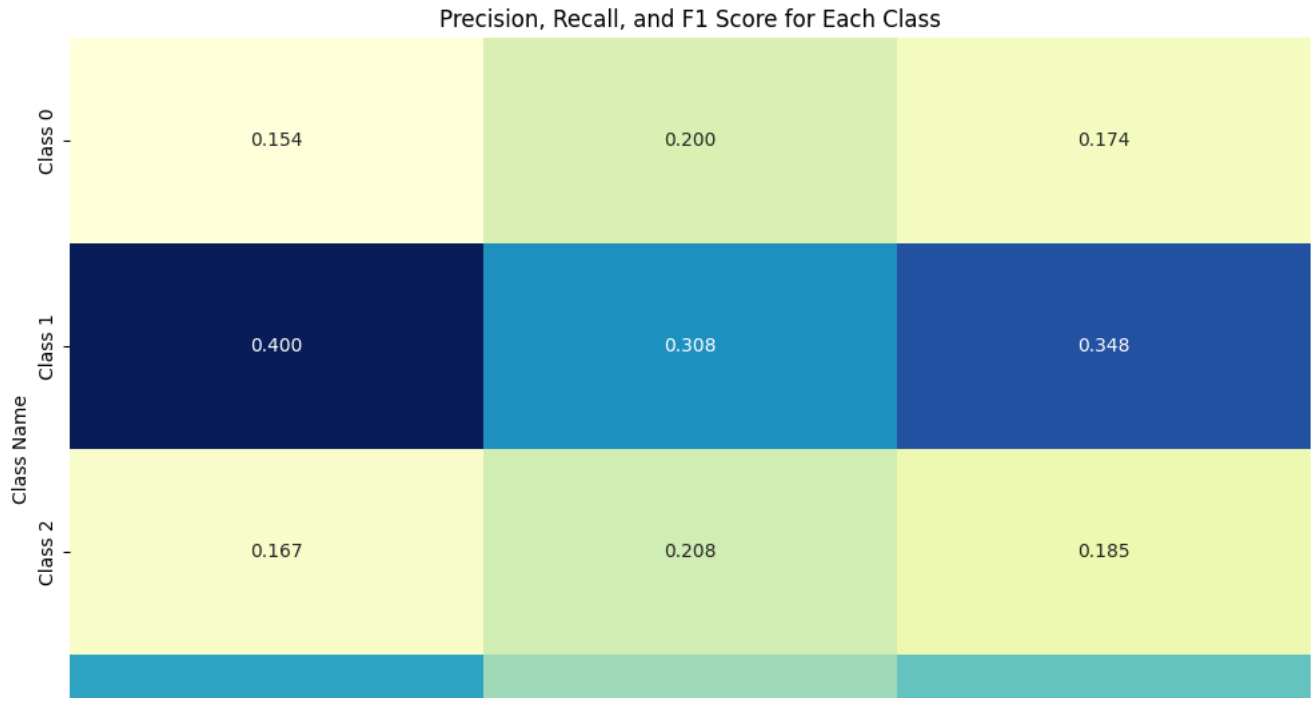
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Calculate confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

```




```
# Specify the path for saving the model in Google Drive
h5_export_path = "/content/drive/MyDrive/maize_model.h5"

# Save the model as an HDF5 file
model.save(h5_export_path)

print(f"Model saved as {h5_export_path}")

```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `
  saving_api.save_model(
Model saved as /content/drive/MyDrive/maize_model.h5
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
import seaborn as sns
import pandas as pd  # Import pandas library

# Generate example data (replace this with your actual data)
np.random.seed(42)
true_labels = np.random.randint(0, 4, 100)
predicted_labels = np.random.randint(0, 4, 100)
class_names = ['Class 0', 'Class 1', 'Class 2','class 3']

# Calculate confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Calculate precision, recall, and f1 score for each class
precision = precision_score(true_labels, predicted_labels, average=None)
recall = recall_score(true_labels, predicted_labels, average=None)
f1 = f1_score(true_labels, predicted_labels, average=None)

# Plot precision, recall, and f1 score in a table
plt.figure(figsize=(12, 8))
table_data = {'Class Name': class_names, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}
sns.heatmap(pd.DataFrame(table_data).set_index('Class Name'), annot=True, cmap="YlGnBu", fmt=".3f", cbar=False)
plt.xlabel('Metrics')
plt.title('Precision, Recall, and F1 Score for Each Class')
plt.show()

```




```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, precision_score, recall_score
import pandas as pd

# Calculate confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Calculate precision and recall for each class as float values
precision = precision_score(true_labels, predicted_labels, average=None)
recall = recall_score(true_labels, predicted_labels, average=None)

# Multiply precision and recall by 100 and format to two decimal places
precision = ["{:.2f}".format(p * 100) for p in precision]
recall = ["{:.2f}".format(r * 100) for r in recall]

# Create a table with reduced width
plt.figure(figsize=(4, 4))  # Smaller table width
table_data = {'Class Name': class_names, 'Precision': precision, 'Recall': recall}
ax = plt.subplot(111, frame_on=False)  # Remove frame around table
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)

# Convert the column labels to a list
col_labels = list(table_data.keys())

# Use col_labels when creating the table
ax.table(cellText=pd.DataFrame(table_data).values, colLabels=col_labels, cellLoc='center', loc='center')

plt.title('Precision and Recall for Each Class')
plt.show()

```