import os

import sys

from tempfile import NamedTemporaryFile

from urllib.request import urlopen

from urllib.parse import unquote, urlparse

from urllib.error import HTTPError

from zipfile import ZipFile

import tarfile

import shutil

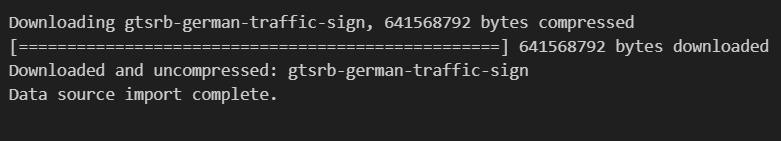
from google.colab import files

uploaded = files.upload()

import zipfile

with zipfile.ZipFile('german\_traffic.zip', 'r') as zip\_ref:

zip\_ref.extractall('data') # Extract to 'data' directory



import numpy as np

from PIL import Image

from sklearn.model\_selection import train\_test\_split

from keras.utils import to\_categorical

from keras.models import Sequential

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

data = []

labels = []

classes = 43

cur\_path = os.getcwd()

for i in range(classes):

path = os.path.join(cur\_path, 'train', str(i))

images = os.listdir(path)

for a in images:

try:

image = Image.open(os.path.join(path, a))

image = image.resize((30, 30))

image = np.array(image)

data.append(image)

labels.append(i)

except:

print("Error loading image")

data = np.array(data)

labels = np.array(labels)

# Display dataset statistics

num\_classes = len(np.unique(image\_labels))

print(f"Number of classes: {num\_classes}")

# Plot distribution of images per class

class\_counts = np.bincount(image\_labels)

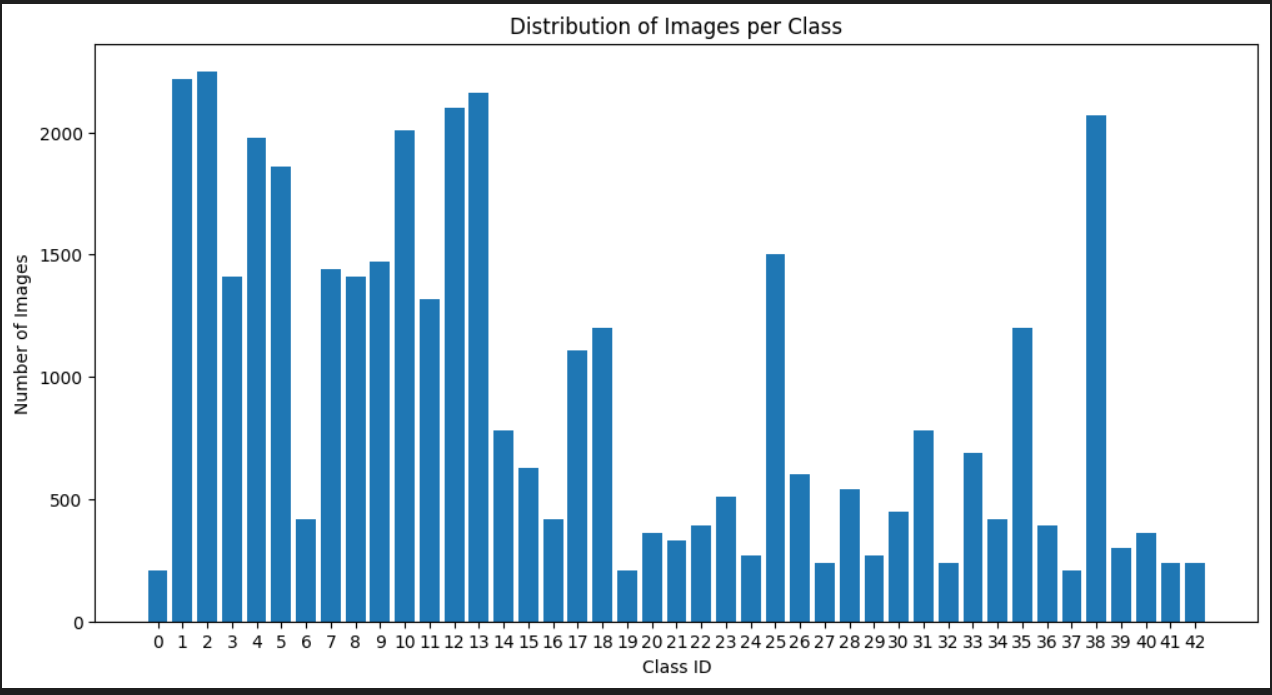
plt.figure(figsize=(12, 6))

plt.bar(range(num\_classes), class\_counts, tick\_label=range(num\_classes))

plt.xlabel("Class ID")

plt.ylabel("Number of Images")

plt.title("Distribution of Images per Class")

plt.show()

# Split the dataset into training and validation sets

X\_train, X\_val, y\_train, y\_val = train\_test\_split(image\_data, image\_labels,

test\_size=0.3, random\_state=42)

# Normalize pixel values to [0, 1]

X\_train = X\_train.astype('float32') / 255.0

X\_val = X\_val.astype('float32') / 255.0

# Convert labels to one-hot encoded format

y\_train = to\_categorical(y\_train, num\_classes)

y\_val = to\_categorical(y\_val, num\_classes)

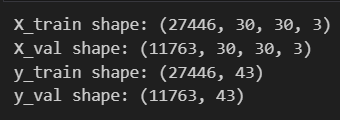
# Display shapes of the training and validation sets

print("X\_train shape:", X\_train.shape)

print("X\_val shape:", X\_val.shape)

print("y\_train shape:", y\_train.shape)

print("y\_val shape:", y\_val.shape)



import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.utils import to\_categorical

# Assuming X\_train, X\_val, y\_train, y\_val are defined from previous preprocessing steps

# Display sample images from each class in the training set

num\_classes = y\_train.shape[1] # Get number of classes from one-hot encoded labels

class\_ids = np.arange(num\_classes) # Array of class IDs

# Determine the layout of subplots based on the number of classes

num\_rows = (num\_classes - 1) // 5 + 1 # Calculate number of rows needed

num\_cols = min(num\_classes, 5) # Maximum of 5 columns

plt.figure(figsize=(num\_cols \* 3, num\_rows \* 3)) # Adjust figure size based on layout

# Loop through each class ID

for i, class\_id in enumerate(class\_ids):

# Find indices of samples corresponding to the current class

sample\_indices = np.where(y\_train[:, class\_id] == 1)[0]

# Select the first sample index (if available) for visualization

if len(sample\_indices) > 0:

sample\_image\_index = sample\_indices[0] # Select the first sample index

sample\_image = X\_train[sample\_image\_index]

# Determine subplot position dynamically based on layout

subplot\_index = i + 1 # Subplot index starts from 1

plt.subplot(num\_rows, num\_cols, subplot\_index)

plt.imshow(sample\_image)

plt.title(f"Class {class\_id}")

plt.axis('off')

plt.tight\_layout()

plt.show()





from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

from tensorflow.keras.optimizers import Adam

import keras

# Build the CNN model

model = keras.Sequential([

Conv2D(filters=16, kernel\_size=(3, 3), activation='relu', input\_shape=(30, 30, 3)),

Conv2D(filters=32, kernel\_size=(3, 3), activation='relu'),

MaxPooling2D(pool\_size=(2, 2)),

BatchNormalization(),

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

Conv2D(filters=128, kernel\_size=(3, 3), activation='relu'),

MaxPooling2D(pool\_size=(2, 2)),

BatchNormalization(),

Flatten(),

Dense(512, activation='relu'),

BatchNormalization(),

Dropout(0.5),

Dense(43, activation='softmax')

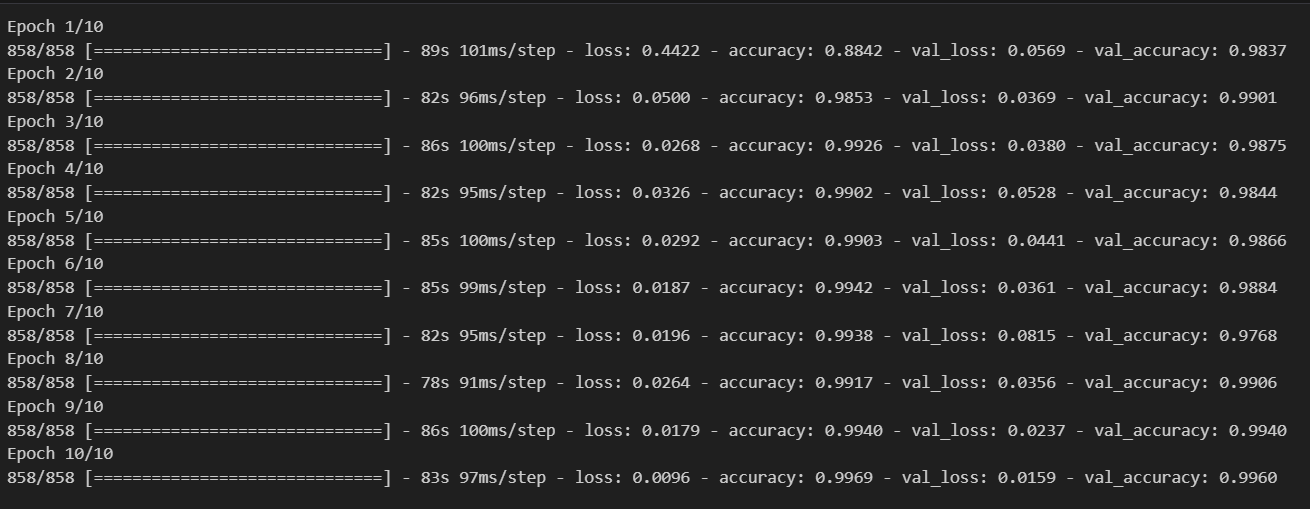
])

# Compile the model

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_val, y\_val), verbose=1)



test = pd.read\_csv(data\_dir + '/Test.csv')

labels = test["ClassId"].values

imgs = test["Path"].values

data =[]

for img in imgs:

try:

image = cv2.imread(data\_dir + '/' +img)

image\_fromarray = Image.fromarray(image, 'RGB')

resize\_image = image\_fromarray.resize((IMG\_HEIGHT, IMG\_WIDTH))

data.append(np.array(resize\_image))

except:

print("Error in " + img)

import numpy as np

from sklearn.metrics import accuracy\_score

# Assuming X\_test and labels are defined

X\_test = np.array(data)

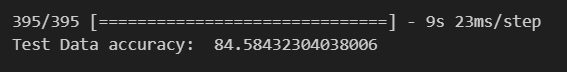
X\_test = X\_test / 255 # Normalize the test data if needed

pred\_probabilities = model.predict(X\_test)

pred\_classes = np.argmax(pred\_probabilities, axis=-1)

# Accuracy with the test data

print('Test Data accuracy: ', accuracy\_score(labels, pred\_classes) \* 100)



# Access the weights of the first convolutional layer

first\_conv\_layer\_weights = model.layers[0].get\_weights()[0]

# Visualize filters of the first convolutional layer

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

for i in range(first\_conv\_layer\_weights.shape[-1]):

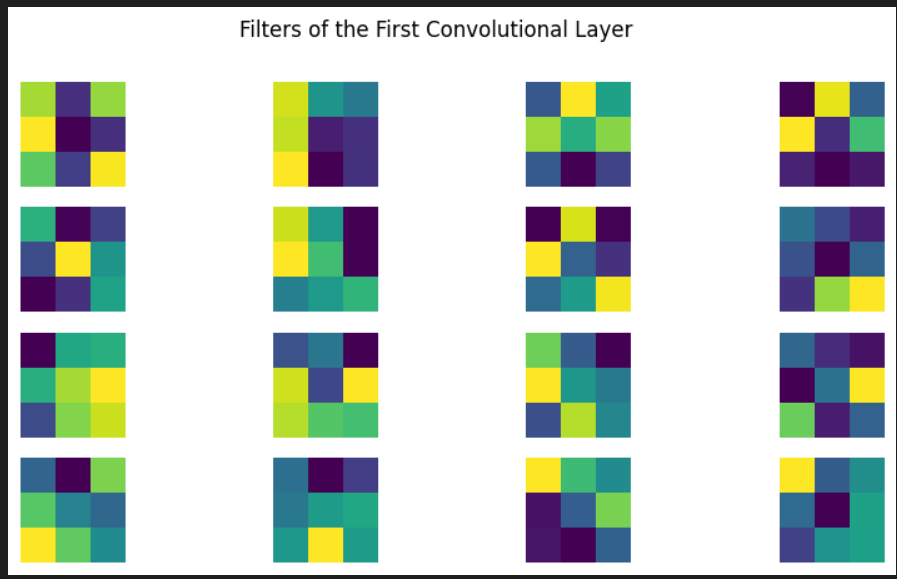
plt.subplot(4, 4, i + 1)

plt.imshow(first\_conv\_layer\_weights[:, :, 0, i], cmap='viridis')

plt.axis('off')

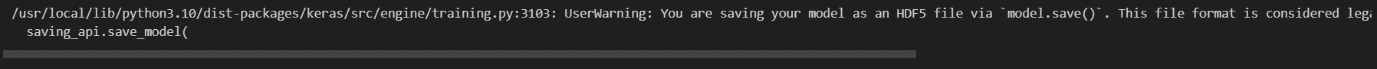
plt.suptitle('Filters of the First Convolutional Layer')

plt.show()



# Save the trained model

model.save('traffic\_sign\_classifier.h5')



import cv2

import numpy as np

from tensorflow.keras.models import load\_model

from IPython.display import display, HTML

import ipywidgets as widgets

from io import BytesIO

from PIL import Image

def preprocess\_image(img):

try:

# Convert uploaded image to numpy array (RGB format)

img = np.array(img)

# Resize the image to match the input size expected by the model

img\_resized = cv2.resize(img, (30, 30))

# Normalize pixel values to [0, 1]

img\_normalized = img\_resized.astype('float32') / 255.0

return img\_normalized

except Exception as e:

print("Error during image preprocessing:", str(e))

return None

def recognize\_traffic\_sign(img, model):

# Preprocess the image

preprocessed\_image = preprocess\_image(img)

# Check if preprocessing was successful

if preprocessed\_image is None:

return None

try:

# Expand dimensions to match model input shape (assuming model expects batches)

preprocessed\_image = np.expand\_dims(preprocessed\_image, axis=0)

# Perform inference using the model

pred\_probabilities = model.predict(preprocessed\_image)

# Get predicted class index

pred\_class\_index = np.argmax(pred\_probabilities, axis=-1)

return pred\_class\_index

except Exception as e:

print("Error during prediction:", str(e))

return None

# Load your trained model

model\_path = '/content/traffic\_sign\_classifier.h5'

model = load\_model(model\_path)

# Create file upload widget

upload\_widget = widgets.FileUpload(accept='image/\*', multiple=False)

def on\_file\_upload(change):

uploaded\_filename = next(iter(upload\_widget.value))

content = upload\_widget.value[uploaded\_filename]['content']

# Open the uploaded image using PIL

img = Image.open(BytesIO(content))

# Display the uploaded image

display(img.resize((200, 200))) # Resize for display

# Recognize traffic sign from the uploaded image

predicted\_class = recognize\_traffic\_sign(img, model)

if predicted\_class is not None:

print(f"Predicted traffic sign class index: {predicted\_class}")

else:

print("Traffic sign recognition failed.")

upload\_widget.observe(on\_file\_upload, names='\_counter')

display(upload\_widget)

