# Import necessary libraries

import cv2

import numpy as np

import os

import matplotlib.pyplot as plt

from tqdm.notebook import tqdm

from google.colab import drive

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

from sklearn.preprocessing import LabelEncoder

# Mount Google Drive

drive.mount('/content/drive')

def load\_images\_from\_folder(folder, target\_size=(128, 128)):

images = []

labels = []

category\_samples = {} # Dictionary to store one sample image per category

# Check the folder and process subfolders

print(f"Checking folder: {folder}")

for category in os.listdir(folder):

category\_path = os.path.join(folder, category)

if os.path.isdir(category\_path):

print(f"Processing category: {category}")

for filename in tqdm(os.listdir(category\_path), desc=f"Loading {category}"):

img\_path = os.path.join(category\_path, filename)

img = cv2.imread(img\_path)

if img is not None:

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

img = cv2.resize(img, target\_size)

images.append(img)

labels.append(category)

# Store a sample image for this category

if category not in category\_samples:

category\_samples[category] = img

# Check if images were loaded

if not images:

print("No images were loaded. Check your dataset structure.")

return None, None

# Visualize sample images from each category

plt.figure(figsize=(15, 10))

for i, (category, sample\_img) in enumerate(category\_samples.items()):

plt.subplot(1, len(category\_samples), i + 1)

plt.imshow(sample\_img, cmap='gray')

plt.title(category)

plt.axis('off')

plt.suptitle("Sample Images from Each Category")

plt.show()

return np.array(images), np.array(labels)

# Set dataset path and target size

dataset\_folder = '/content/drive/MyDrive/Animals'

target\_image\_size = (128, 128)

# Load dataset

images, labels = load\_images\_from\_folder(dataset\_folder, target\_size=target\_image\_size)

# Visualize the distribution of categories

unique, counts = np.unique(labels, return\_counts=True)

plt.bar(unique, counts, color='skyblue')

plt.title("Distribution of Categories in the Dataset")

plt.xlabel("Category")

plt.ylabel("Number of Images")

plt.xticks(rotation=45)

plt.show()

# Normalize the images

images = images / 255.0

# Train-test split

train\_images, test\_images, train\_labels, test\_labels = train\_test\_split(

images, labels, test\_size=0.2, stratify=labels, random\_state=42

)

# Visualize a few random training images

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

for i in range(9):

idx = np.random.randint(0, len(train\_images))

plt.subplot(3, 3, i + 1)

plt.imshow(train\_images[idx], cmap='gray')

plt.title(f"Label: {train\_labels[idx]}")

plt.axis('off')

plt.suptitle("Random Training Images")

plt.show()

# Flatten the images for k-NN

X\_train\_flat = train\_images.reshape(train\_images.shape[0], -1)

X\_test\_flat = test\_images.reshape(test\_images.shape[0], -1)

# Encode labels

le = LabelEncoder()

y\_train = le.fit\_transform(train\_labels)

y\_test = le.transform(test\_labels)

# Visualize flattened images

print("Flattened Image Shape:", X\_train\_flat.shape)

# Visualizing distances (example: between first two test and train samples)

plt.figure(figsize=(6, 4))

plt.plot(X\_train\_flat[0], label="First Train Image")

plt.plot(X\_test\_flat[0], label="First Test Image")

plt.legend()

plt.title("Pixel Values of Flattened Images")

plt.xlabel("Pixel Index")

plt.ylabel("Pixel Value")

plt.show()

# Define k-NN class

class KNearestNeighbor:

def train(self, X, y):

self.X\_train = X

self.y\_train = y

def predict(self, X, k=1):

dists = self.compute\_distances(X)

return self.predict\_labels(dists, k)

def compute\_distances(self, X):

return np.sqrt((-2 \* np.dot(X, self.X\_train.T)) +

np.sum(X\*\*2, axis=1, keepdims=True) +

np.sum(self.X\_train\*\*2, axis=1))

def predict\_labels(self, dists, k):

num\_test = dists.shape[0]

y\_pred = np.zeros(num\_test, dtype=self.y\_train.dtype)

for i in range(num\_test):

k\_nearest\_idxs = np.argsort(dists[i])[:k]

closest\_y = self.y\_train[k\_nearest\_idxs]

y\_pred[i] = np.bincount(closest\_y).argmax()

return y\_pred

# Perform 5-fold cross-validation with visualization

def cross\_validate\_knn(X, y, k\_choices):

from sklearn.model\_selection import StratifiedKFold

skf = StratifiedKFold(n\_splits=5)

k\_to\_accuracies = {k: [] for k in k\_choices}

for train\_idx, val\_idx in skf.split(X, y):

X\_train\_fold, X\_val\_fold = X[train\_idx], X[val\_idx]

y\_train\_fold, y\_val\_fold = y[train\_idx], y[val\_idx]

model = KNearestNeighbor()

model.train(X\_train\_fold, y\_train\_fold)

for k in k\_choices:

y\_val\_pred = model.predict(X\_val\_fold, k=k)

accuracy = np.mean(y\_val\_pred == y\_val\_fold)

k\_to\_accuracies[k].append(accuracy)

# Visualize fold-wise accuracies for one example K

plt.figure(figsize=(10, 6))

plt.plot(k\_choices, [np.mean(k\_to\_accuracies[k]) for k in k\_choices], marker='o')

plt.title("Mean Accuracy Across Folds for Each K")

plt.xlabel("Value of K")

plt.ylabel("Mean Accuracy")

plt.show()

return k\_to\_accuracies

# Evaluate and visualize results

k\_choices = range(1, 31)

k\_to\_accuracies = cross\_validate\_knn(X\_train\_flat, y\_train, k\_choices)

# Final plot of accuracies

mean\_accuracies = [np.mean(k\_to\_accuracies[k]) for k in k\_choices]

std\_accuracies = [np.std(k\_to\_accuracies[k]) for k in k\_choices]

plt.errorbar(k\_choices, mean\_accuracies, yerr=std\_accuracies, fmt='o')

plt.plot(k\_choices, mean\_accuracies, label="Mean Accuracy")

plt.xlabel("Value of K")

plt.ylabel("Accuracy")

plt.title("k-NN Accuracy vs. K")

plt.legend()

plt.show()