```
# I have used MNIST dataset from kaggle in csv format.

import pandas as pd
import numpy as np
import math
df = pd.read_csv('/content/drive/MyDrive/mnist_train.csv')



print(df)



# Using first column of dataset as label.

label = df.iloc[:,0]
label = pd.DataFrame(label)
# label = label.drop(0).reset_index(drop=True)
print(label)

# I have padded the image with 0 from all direction.

# This function will create 1d array to 2d array and also apply
padding at the boundary of each image
def padding(arr):
    n = int(len(arr)**0.5)
    arr = arr.reshape(n,n)
    arr1 = [[0 for i in range(n+1)] for j in range(n+1)]
    for i in range(n):
        for j in range(n):
            arr1[i][j]=arr[i][j]
    return arr1

# Code for making differential filter
x_diff = [[1,0,-1],[1,0,-1],[1,0,-1]]
y_diff = [[1,1,1],[0,0,0],[-1,-1,-1]]

# Making Gaussian filter and random sigma and size of kernel so that
hypertuning can be easy.

# Code for making Gaussian filter
def gaussian_filter(sigma,n):
    d = 2*(np.pi)*(sigma**2)
    arr = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            x = n//2-i
            y = n//2-j
            arr[i][j] = (np.exp(-((x**2+y**2)/(2*sigma**2))))/d
    return arr

# Code for multiply two differential and gaussian filter
def multiply(arr1,arr2):
```

```
    arr = [[0 for i in range(len(arr1))] for j in range(len(arr1[0]))]
    for i in range(len(arr1)):
        for j in range(len(arr1[0])):
            arr[i][j] = arr1[i][j]*arr2[i][j]
    return arr
```

Taking the cell size of 4*4 and block size of 2*2 and computing the gradient magnitude and direction and making an array of pairs to store both values simultaneously.

```
# Calculating Gradient magnitude and tan theta and storing these
values into arr of pairs
def gradient(arr,f1,f2):
    arr1 =  [[[0,0] for i in range(len(arr)-1)] for j in
range(len(arr[0])-1)]
    for i in range(1,len(arr)-1):
        for j in range(1,len(arr[0])-1):
            a = 0
            b = 0
            for k in range(i-1,i+2):
                for l in range(j-1,j+2):
                       a+= (arr[k][l]*f1[abs(i-1-k)][abs(j-1-l)])
                       b+= (arr[k][l]*f2[abs(i-1-k)][abs(j-1-l)])
            c = (a**2+b**2)**0.5
            d = math.atan2(b,a)
            d = math.degrees(d)
            arr1[i][j][0] = c
            arr1[i][j][1] = d
    return arr1

# Calculating histogram for n*n square and storing them in array
def histogram(arr):
    n = len(arr)
    m = len(arr[0])
    arr1 = np.zeros((n//4, m//4, 9))
    for i in range(0,n,4):
        for j in range(0,m,4):
             for k in range(i,i+4):
                for l in range(j,j+4):
                         a = arr[i][j][0]
                         b = arr[i][j][1]
                         c = abs((b//45)+1)*45-a
                         if b > 0:
                            # print(b)
                            ind = int(4+b//45)
                            if b == 180:
                              arr1[i//4][j//4][ind]+= a
                            else:
                              arr1[i//4][j//4][ind]+= (c/45)*a
                              arr1[i//4][j//4][ind+1]+=  (45-c/45)*a
                         else:
```

```
                                ind = int(4-b//45)
                                if b==180:
                                  arr1[i//4][j//4][ind]+= (c/45)*a
                                else:
                                  arr1[i//4][j//4][ind]+= (c/45)*a
                                  arr1[i//4][j//4][ind-1]+=  (45-c/45)*a
    return arr1
```

Here I will perform normalization of the array and taking e =1 to avoid division by zero that will lead to 36*36 features of image as I used 9 bins

```
# Now We will perform normalization of the array
def normalization(arr):
    n,m = len(arr),len(arr[0])
    arr1 = np.zeros((n-1,m-1,36))
    r = [0,0,1,1]
    c = [0,1,0,1]
    for i in range(n-1):
        for j in range(m-1):
             arr2 = []
             norm = 1
             for k in range(4):
                ind1 = i+r[k]
                ind2 = j+c[k]
                for val in arr[ind1][ind2]:
                    arr2.append(val)
                    norm+=val**2
             norm = norm**0.5
             for k in range(36):
                arr1[i][j][k]=arr2[i]/norm
    return arr1

# Code for making final feature vector
def make_vector(arr):
    arr1 = []
    for i in range(len(arr)):
        for j in range(len(arr[0])):
          for k in range(36):
            arr1.append(arr[i][j][k])
    return arr1

# Getting Our training dataset
train = df.drop(df.columns[0],axis=1)
print(train.head())
print(train.values.shape)

# Combining all these functions generating dataset which will contain features
data = [[]]
# Gaussian filter
```

```
gf = gaussian_filter(1,3)
xf = multiply(gf,x_diff)
yf = multiply(gf,y_diff)

new_data = train.values
for row in range(train.values.shape[0]):
     temp = padding(new_data[row])
     temp = gradient(temp,xf,yf)
     temp = histogram(temp)
     temp = normalization(temp)
     temp = make_vector(temp)
     data.append(temp)

feature_data = pd.DataFrame(data)

feature_data = feature_data.drop(0).reset_index(drop=True)




test = pd.read_csv('/content/drive/MyDrive/mnist_test.csv')

test_label = test.iloc[:,0]
test =  test.drop(test.columns[0],axis=1)

new_data = test.values
test_data = [[]]
for row in range(test.values.shape[0]):
     temp = padding(new_data[row])
     temp = gradient(temp,xf,yf)
     temp = histogram(temp)
     temp = normalization(temp)
     temp = make_vector(temp)
     test_data.append(temp)

test_data = pd.DataFrame(test_data)

test_data = test_data.drop(0).reset_index(drop=True)

# Using Random forest classifier for classification problem of dataset
and predicting the values and checking the accuracy

import pandas as pd
from sklearn.ensemble import RandomForestClassifier  # Import Random
Forest
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(feature_data)
X_test_scaled = scaler.transform(test_data)
```

```
rf_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)  # You can tune n_estimators, etc.
rf_classifier.fit(X_train_scaled, label)  # Fit the model on the
training data


y_pred = rf_classifier.predict(X_test_scaled)


accuracy = accuracy_score(test_label, y_pred)


print(f'Accuracy: {accuracy * 100:.2f}%')
```

I initially used a Random Forest Classifier for classification, which gave an accuracy of 46.39%. However, after optimizing my code with the assistance of ChatGPT and switching to a Neural Network for classification, my accuracy significantly improved to 98.41%. Although I learned a lot from the code, I unfortunately didn’t have enough time to implement it from scratch, as the earlier code consumed a considerable amount of time. Below is the code that I used for the same:

```
import numpy as np
import tensorflow as tf
import cv2
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

def gaussian_filter(sigma, n):
    d = 2 * np.pi * (sigma ** 2)
    arr = np.zeros((n, n))
    center = n // 2
    for i in range(n):
        for j in range(n):
            x = i - center
            y = j - center
            arr[i, j] = np.exp(-(x ** 2 + y ** 2) / (2 * sigma ** 2))
/ d
    return arr

def multiply(arr1, arr2):
    return arr1 * arr2
```

```
def gradient(arr, f1, f2):
    arr = np.float32(arr)
    grad_x = cv2.filter2D(arr, -1, f1)
    grad_y = cv2.filter2D(arr, -1, f2)
    gradient_magnitude = np.sqrt(grad_x ** 2 + grad_y ** 2)
    gradient_angle = np.arctan2(grad_y, grad_x) * (180 / np.pi)
    return np.stack((gradient_magnitude, gradient_angle), axis=-1)

def histogram(arr, cell_size=4, bins=9):
    n, m, _ = arr.shape
    hist = np.zeros((n // cell_size, m // cell_size, bins))
    bin_edges = np.linspace(-180, 180, bins + 1)

    for i in range(0, n, cell_size):
        for j in range(0, m, cell_size):
            block = arr[i:i + cell_size, j:j + cell_size, 1]
            magnitude = arr[i:i + cell_size, j:j + cell_size, 0]
            block_angles = block.flatten()
            block_magnitudes = magnitude.flatten()
            indices = np.digitize(block_angles, bin_edges) - 1
            indices = np.clip(indices, 0, bins - 1)
            for idx, mag in zip(indices, block_magnitudes):
                hist[i // cell_size, j // cell_size, idx] += mag
    return hist

def normalization(hist, block_size=(2, 2), bins=9):
    n, m, _ = hist.shape
    norm_hist = np.zeros_like(hist)

    # Iterating over the image using block_size
    for i in range(0, n - block_size[0] + 1):
        for j in range(0, m - block_size[1] + 1):
            block = hist[i:i + block_size[0], j:j + block_size[1], :]
            block_flattened = block.flatten()
            norm = np.linalg.norm(block_flattened)  # Compute the L2
norm of the block
            if norm > 0:
                norm_hist[i:i + block_size[0], j:j + block_size[1], :]
= block / norm
    return norm_hist

def make_vector(arr):
    return arr.flatten()

def extract_hog_features(image, sigma=1, n=3, cell_size=4,
block_size=(2, 2), bins=9):
    gf = gaussian_filter(sigma, n)
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
    if len(image.shape) == 3:
```

```
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    grad = gradient(image, sobel_x, sobel_y)
    hist = histogram(grad, cell_size, bins)
    norm_hist = normalization(hist, block_size, bins)
    feature_vector = make_vector(norm_hist)
    return feature_vector

def train_and_evaluate(train_data, test_data, label_col=0,
test_size=0.2, epochs=10, batch_size=32):
    X_train =
train_data.drop(columns=[train_data.columns[label_col]]).values
    y_train = train_data.iloc[:, label_col].values
    X_test =
test_data.drop(columns=[test_data.columns[label_col]]).values
    y_test = test_data.iloc[:, label_col].values

    X_train_images = X_train.reshape(-1, 28, 28)
    X_test_images = X_test.reshape(-1, 28, 28)

    hog_features_train = [extract_hog_features(image) for image in
X_train_images]
    hog_features_test = [extract_hog_features(image) for image in
X_test_images]

    X_train_hog = np.array(hog_features_train)
    X_test_hog = np.array(hog_features_test)

    scaler = StandardScaler()
    X_train_hog = scaler.fit_transform(X_train_hog)
    X_test_hog = scaler.transform(X_test_hog)

    model = tf.keras.Sequential([

tf.keras.layers.InputLayer(input_shape=(X_train_hog.shape[1],)),
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

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

    history = model.fit(X_train_hog, y_train, epochs=epochs,
batch_size=batch_size, validation_data=(X_test_hog, y_test))

    test_loss, test_accuracy = model.evaluate(X_test_hog, y_test)
```

```
    print(f"Test Accuracy: {test_accuracy * 100:.2f}%")


train_data = pd.read_csv('/content/drive/MyDrive/mnist_train.csv')
test_data = pd.read_csv('/content/drive/MyDrive/mnist_test.csv')

train_and_evaluate(train_data, test_data)
```