Appendix A

Dataset

Breast Cancer Histopathology Dataset (BreakHis)

The experiment utilized the publicly accessible BreakHis dataset for research purposes. For years, researchers have been utilizing this dataset to classify cases of breast cancer, and it is publicly accessible.

The P&D Laboratory in Brazil collected the database during a clinical investigation spanning from January 2014 to December 2014. The samples are acquired through surgical open biopsy and then stained with hematoxylin and eosin (H&E). The dataset comprises a total of 7,909 samples, categorized into two groups: benign and malignant. There are 2,480 benign samples and 5,429 malignant samples. The samples are captured at different magnifications, such as 40X, 100X, 200X, and 400X. The below table illustrates the distribution of the dataset across benign and malignant classes at various magnification levels [63]. The description of the dataset is as follows:

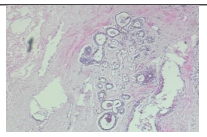
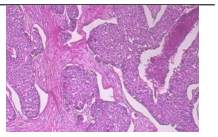
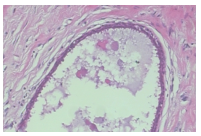
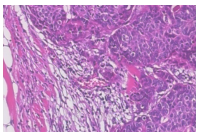
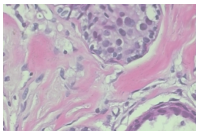
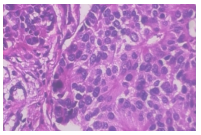
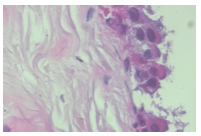
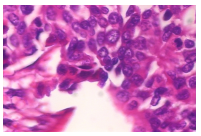
BreakHis dataset distribution

Magnification	**Benign**	**Malignant**	**Total**
40X	625	1370	1995
100X	644	1437	2081
200X	623	1390	2013
400X	588	1232	1820
Total Images	2480	5429	7909

These image collections are converted into grayscale images.

Below are a few examples of the provided images.

BreakHis Data Samples

Magnification	**Benign**	**Malignant**
40X		
100X		
200X		
400X		

Appendix B

Program simulating concept described in earlier chapters

Feature Extraction Program

Feature Extraction Program using method GLCM, LBP and Gabor Filter

```
# -*- coding: utf-8 -*-
"""All feature extraction.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1TiyyB67sQY7vkC_R88r0yBkRK9F-CRzU
"""

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
from torchvision import datasets
import PIL
from skimage.feature import local_binary_pattern,greycomatrix, greycoprops
from skimage.filters import gabor
import numpy as np
import pickle
```

```
18 import matplotlib.pyplot as plt
19
20 #Upload ZIP File
21 from google.colab import drive
22 drive.mount('/content/gdrive')
23
24 import os
25
26 trainDset = os.listdir('/content/gdrive/MyDrive/40x/40X/malignant'
       ) # dir is your directory path
27 number_files = len(trainDset)
28
29 print (number_files)
30 print(trainDset)
31
32 from skimage.io import imread
33 from matplotlib.pyplot import subplot, imshow
34 import numpy as np
35 import cv2
36 from scipy import misc
37 from skimage.feature import greycomatrix, greycoprops
38
39 dataset = []
40 featLength = 9
41 trainFeats = np.zeros((len(trainDset),featLength))
42 for i in range (len(trainDset)):
43   img=cv2.imread('/content/gdrive/MyDrive/40x/40X/malignant/'+
       trainDset[i],0)
44   glcm = greycomatrix(img,[1],[0],256,symmetric = True,normed =
       True)
45   #LBP
46   feat_lbp = local_binary_pattern(img,5,2,'uniform')
47   lbp_hist,_ = np.histogram(feat_lbp, 8)
48   lbp_hist = np.array(lbp_hist,dtype=float)
49   lbp_prob = np.divide(lbp_hist, np.sum(lbp_hist))
50   lbp_energy = np.nansum(lbp_prob**2)
51   lbp_entropy =-np.nansum(np.multiply(lbp_prob,np.log2(lbp_prob)))
52 #   print("energy= ",lbp_energy)
53 #   print('entropy= ',lbp_entropy)
54   #GLCM
55   contrast = greycoprops(glcm,prop = 'contrast')
56   dissimilarity = greycoprops(glcm, prop = 'dissimilarity')
57   homogeneity = greycoprops(glcm, prop = 'homogeneity')
58   energy = greycoprops(glcm, prop = 'energy')
```

```
#  ASM = greycoprops(glcm, prop = 'ASM')
  correlation = greycoprops(glcm, prop = 'correlation')
#  print('contrast = ',contrast)
#  print('dissimilarity = ', dissimilarity)
#  print('homogeneity = ', homogeneity)
#  print('energy= ',energy)
#  print('correlation = ',correlation)
#  print('ASM = ',ASM)
  print()
  #Gabor
  gaborFilt_real,gaborFilt_image = gabor(img,frequency=0.6)
  gaborFilt = (gaborFilt_real**2 + gaborFilt_image**2)//2
  gabor_hist,_ = np.histogram(gaborFilt,8)
  gabor_hist = np.array(gabor_hist,dtype = float)
  gabor_prob = np.divide(gabor_hist,np.sum(gabor_hist))
  gabor_energy = np.nansum(gabor_prob**2)
  gabor_entropy = -np.nansum(np.multiply(gabor_prob, np.log2(
   gabor_prob)))
#  print('gabor_energy = ', gabor_energy)
#  print('gabor_entropy =',gabor_entropy)
  feat_gabor = np.array([gabor_energy,gabor_entropy])
  feat_glcm = np.array([contrast,dissimilarity,homogeneity,energy,
   correlation])
  feat_lbp = np.array([lbp_energy,lbp_entropy])
  concat_feat = np.concatenate((feat_lbp,feat_glcm,feat_gabor),
   axis=None)
  trainFeats[i,:] = concat_feat #stacking features vectors for
   each image
  dataset.append(trainDset[i])
trainLabel = np.array(dataset)

#Feature Normalization for training dataset
trMaxs = np.amax(trainFeats, axis=0)
trMins = np.amin(trainFeats, axis=0)
trMaxs_rep = np.tile(trMaxs,(number_files,1))
trMins_rep = np.tile(trMins,(number_files,1))
trainFeatsNorm  = np.divide(trainFeats-trMins_rep,trMaxs_rep)

#Saving Feature Matrices to disk for training dataset
'''
with open("trainFeats.pck1","wb") as f:
  pickle.dump(trainFeatsNorm,f)
with open("trainLabel.pck1","wb") as f:
  pickle.dump(trainLabel, f)
```

```
99
100 from numpy import savetxt
101 savetxt('test1x.xls',trainFeatsNorm)
102 savetxt('test1l.xls',trainLabel,fmt='%s   ')
103 '''
104 from numpy import savetxt
105 savetxt('40xmalignant.csv',trainFeatsNorm,fmt='%s')
106 #savetxt('test1l.xls',trainLabel,fmt='%s   ')
```

Feature Extraction Program using CNN

```
1  # -*- coding: utf-8 -*-
2  """CNN.ipynb
3
4  Automatically generated by Colab.
5
6  Original file is located at
7      https://colab.research.google.com/drive/1
       dQA9ZtK1chafciQ4b1PiRVHmFGTcjjMy
8  """
9
10 # CNN
11
12
13 import os
14 import numpy as np
15 import tensorflow as tf
16 from tensorflow import keras
17 from PIL import Image
18
19 # Define the CNN model using Keras
20 class CNN(tf.keras.Model):
21     def __init__(self):
22         super(CNN, self).__init__()
23         self.conv1 = keras.layers.Conv2D(32, (3, 3), activation='
       relu', input_shape=(28, 28, 1))
24         self.conv2 = keras.layers.Conv2D(64, (3, 3), activation='
       relu')
25         self.pool = keras.layers.MaxPooling2D(pool_size=(2, 2))
26         self.flatten = keras.layers.Flatten()
27         self.fc1 = keras.layers.Dense(128, activation='relu')
28         self.fc2 = keras.layers.Dense(9)  # 9 output features
```

```
29
30      def call(self, x):
31          x = self.conv1(x)
32          x = self.conv2(x)
33          x = self.pool(x)
34          x = self.flatten(x)
35          x = self.fc1(x)
36          x = self.fc2(x)
37          return x
38
39  # Folder of images
40  folder_path = "/content/gdrive/MyDrive/40X/40X_B"
41
42  # List all images in the folder
43  image_files = os.listdir(folder_path)
44
45  # Initialize the CNN model
46  model = CNN()
47
48  # Initialize an empty array to store the output for each image
49  output_array = []
50
51  # Loop through each image in the folder
52  for i, file in enumerate(image_files):
53      # Load the image and preprocess it
54      image = Image.open(os.path.join(folder_path, file)).convert('L
    ')
55      image = image.resize((28, 28))
56      image = np.asarray(image)
57      image = image / 255.0
58      image = np.expand_dims(image, axis=-1)  # Add a channel
    dimension
59      image = np.expand_dims(image, axis=0)   # Add a batch
    dimension
60
61      # Pass the data through the model
62      output = model.predict(image)
63      # Store the output in the output array
64      output_array.append(output)
65
66  # Convert the output list to a numpy array
67  output_array = np.array(output_array)
68
69  # Normalize the output_array to values between 0 and 1
```

```
output_array_normalized = (output_array - output_array.min()) / (
    output_array.max() - output_array.min())

# Print the normalized output array
print(output_array_normalized)

# Save the normalized output array as a CSV file

np.savetxt("40X_M_CNN.csv", output_array_normalized.reshape(-1, 9)
    , delimiter=",")
```

Optimized Feature Extraction program for Histopathology Image Analysis

Gradient Decent with Relu and Sigmoid Activation Function

```
# -*- coding: utf-8 -*-
"""GD + relu +sigmoid.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1
    Y0j64KOpajNEPweXvkfyzAvvBPcxLFTn
"""

import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error, confusion_matrix,
    accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import KFold, train_test_split
from time import time

t1 = time()

# Define the neural network architecture
input_size = 9
hidden_size = 2
output_size = 1

```

```
# Define the ReLU activation function
def relu(x):
    return np.maximum(0, x)

# Define the derivative of the ReLU activation function
def relu_derivative(x):
    return np.where(x > 0, 1, 0)

# Define the sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Define the derivative of the sigmoid activation function
def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))

# Define the loss function
def mse_loss(y_true, y_pred):
    epsilon = 1e-5
    y_pred = np.minimum(np.maximum(y_pred, epsilon), 1 - epsilon)
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.
   log(1 - y_pred))

# Load the train and test datasets from CSV files
df = pd.read_csv('/content/CNN_40X.csv', header=None).values

# get the locations
X = df[:, :-1]
y = df[:, -1:]

# Set the learning rate and number of epochs
learning_rate = 0.01
epochs = 20
eps = 1e-8

# Initialize the KFold object
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize performance metric lists for cross-validation
precisions = []
recalls = []
specificities = []
f1s = []
```

```
accuracies = []

# Initialize a running total for the confusion matrix
total_cm = np.zeros((2, 2))

# Iterate over the k-folds
for train_index, test_index in kf.split(X):
    # Split the data into train and test sets
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Initialize the weights and biases for each fold
    W1 = np.random.randn(input_size, hidden_size)
    b1 = np.random.randn(hidden_size)
    W2 = np.random.randn(hidden_size, output_size)
    b2 = np.random.randn(output_size)

    sw1 = 0
    sw2 = 0
    sb1 = 0
    sb2 = 0

    # Train the neural network using backpropagation and gradient descent
    for epoch in range(epochs):
        # Forward pass
        z1 = np.dot(X_train, W1) + b1
        a1 = relu(z1)  # Use ReLU activation in the first layer
        z2 = np.dot(a1, W2) + b2
        y_pred = sigmoid(z2)

        # Compute the loss and gradients
        loss = mse_loss(y_train, y_pred)
        delta2 = (y_pred - y_train) * sigmoid_derivative(z2)
        delta1 = np.dot(delta2, W2.T) * relu_derivative(z1)  # Use ReLU derivative in the first layer
        dW2 = np.dot(a1.T, delta2)
        db2 = np.sum(delta2, axis=0)
        dW1 = np.dot(X_train.T, delta1)
        db1 = np.sum(delta1, axis=0)  # Sum over axis=0

        # Update the weights and biases using gradient descent
        W2 -= learning_rate * dW2
        b2 -= learning_rate * db2
```

```
108         W1 -= learning_rate * dW1
109         b1 -= learning_rate * db1
110 
111         # Compute the loss and gradients
112         loss = mse_loss(y_train, y_pred)
113         if epoch % 5 == 0:  # Print mean squared error every 100
    epochs
114             print(f"Epoch {epoch}, Cross Entropy loss: {loss:.4f}"
    )
115     # Predict the output for the test data using the trained
    neural network
116     y_pred = sigmoid(np.dot(relu(np.dot(X_test, W1) + b1), W2) +
    b2)
117 
118     # Threshold the predictions using the mean of y_pred as the
    threshold
119     M = y_pred.mean()
120     for i in range(len(y_pred)):
121         if y_pred[i] >= M:
122             y_pred[i] = 1
123         else:
124             y_pred[i] = 0
125 
126     # Calculate the evaluation metrics
127     precision = precision_score(y_test, y_pred)
128     recall = recall_score(y_test, y_pred)
129     specificity = recall_score(1 - y_test, 1 - y_pred)
130     f1 = f1_score(y_test, y_pred)
131     accuracy = accuracy_score(y_test, y_pred)
132 
133     # Add the confusion matrix for this fold to the running total
134     cm = confusion_matrix(y_test, y_pred)
135     total_cm += cm
136 
137     # Append the performance metrics for this fold
138     precisions.append(precision)
139     recalls.append(recall)
140     specificities.append(specificity)
141     f1s.append(f1)
142     accuracies.append(accuracy)
143 
144 # Print the final confusion matrix
145 print("Confusion Matrix:")
146 print(total_cm)
```

```
tn, fp, fn, tp = total_cm.ravel()
# Calculate the evaluation metrics
accuracy = (total_cm[0][0] + total_cm[1][1]) / np.sum(total_cm)
precision = total_cm[1][1] / (total_cm[1][1] + total_cm[0][1])
recall = total_cm[1][1] / (total_cm[1][1] + total_cm[1][0])
specificity = total_cm[0][0] / (total_cm[0][0] + total_cm[0][1])
f1 = 2 * precision * recall / (precision + recall)

# Print the evaluation metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"Specificity: {specificity:.4f}")
print(f"F1 score: {f1:.4f}")

# Print the final weights and biases
print("W1: ")
print(W1)
print("b1: ")
print(b1)
print("W2: ")
print(W2)
print("b2: ")
print(b2)

# Print the average evaluation metrics across k-folds
print(f"Average Precision: {np.mean(precisions):.4f}")
print(f"Average Recall: {np.mean(recalls):.4f}")
print(f"Average Specificity: {np.mean(specificities):.4f}")
print(f"Average F1 score: {np.mean(f1s):.4f}")
print(f"Average Accuracy: {np.mean(accuracies):.4f}")

# Print the total time
t2 = time()
total_time = t2 - t1
print('Total Time', total_time)
```

Gradient Decent Momentum with Relu and Sigmoid Activation Function

```
# -*- coding: utf-8 -*-
"""GDM + relu +sigmoid.ipynb
```

```
3
4 Automatically generated by Colab.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1
    UlBSokKJGaAxowDb2ZPclVxD6R0wT6vp
8 """
9
10 import numpy as np
11 import pandas as pd
12 from sklearn.metrics import mean_squared_error, confusion_matrix,
    accuracy_score, precision_score, recall_score, f1_score
13 from sklearn.model_selection import KFold, train_test_split
14 from time import time
15
16 t1 = time()
17
18 # Define the neural network architecture
19 input_size = 9
20 hidden_size = 2
21 output_size = 1
22
23 # Define the ReLU activation function
24 def relu(x):
25     return np.maximum(0, x)
26
27 # Define the derivative of the ReLU activation function
28 def relu_derivative(x):
29     return np.where(x > 0, 1, 0)
30
31 # Define the sigmoid activation function
32 def sigmoid(x):
33     return 1 / (1 + np.exp(-x))
34
35 # Define the derivative of the sigmoid activation function
36 def sigmoid_derivative(x):
37     return sigmoid(x) * (1 - sigmoid(x))
38
39
40 # Define the loss function
41 def mse_loss(y_true, y_pred):
42     epsilon = 1e-5
43     y_pred = np.minimum(np.maximum(y_pred, epsilon), 1 - epsilon)
```

```
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

# Load the train and test datasets from CSV files
df = pd.read_csv('/content/CNN_40X.csv', header=None).values

# get the locations
X = df[:, :-1]
y = df[:, -1:]

# Set the learning rate and number of epochs
learning_rate = 0.01
epochs = 20
b = 0.9
eps = 1e-8

# Initialize the KFold object
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize performance metric lists for cross-validation
precisions = []
recalls = []
specificities = []
f1s = []
accuracies = []

# Initialize a running total for the confusion matrix
total_cm = np.zeros((2, 2))

# Iterate over the k-folds
for train_index, test_index in kf.split(X):
    # Split the data into train and test sets
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Initialize the weights and biases for each fold
    W1 = np.random.randn(input_size, hidden_size)
    b1 = np.random.randn(hidden_size)
    W2 = np.random.randn(hidden_size, output_size)
    b2 = np.random.randn(output_size)

    sw1 = 0
    sw2 = 0
```

```
    sb1 = 0
    sb2 = 0

    # Train the neural network using backpropagation and gradient descent
    for epoch in range(epochs):
        # Forward pass
        z1 = np.dot(X_train, W1) + b1
        a1 = relu(z1)  # Use ReLU activation in the first layer
        z2 = np.dot(a1, W2) + b2
        y_pred = sigmoid(z2)

        # Compute the loss and gradients
        loss = mse_loss(y_train, y_pred)
        delta2 = (y_pred - y_train) * sigmoid_derivative(z2)
        delta1 = np.dot(delta2, W2.T) * relu_derivative(z1)  # Use ReLU derivative in the first layer
        dW2 = np.dot(a1.T, delta2)
        db2 = np.sum(delta2, axis=0)
        dW1 = np.dot(X_train.T, delta1)
        db1 = np.sum(delta1, axis=0)  # Sum over axis=0

       # update weigths and bias by Gradient Decent with momentum
        sw2 = b*sw2 + (1-b)*dW2
        W2 -= learning_rate * sw2
        sb2 = b*sb2+ (1-b)*db2
        b2 -= learning_rate * sb2
        sw1 = b*sw1 + (1-b)*dW1
        W1 -= learning_rate * sw1
        sb1 = b*sb1 + (1-b)*db1
        b1 -= learning_rate * sb1

        # Compute the loss and gradients
        loss = mse_loss(y_train, y_pred)
        if epoch % 5 == 0:  # Print mean squared error every 100 epochs
            print(f"Epoch {epoch}, Cross Entropy loss: {loss:.4f}")

    # Predict the output for the test data using the trained neural network
    y_pred = sigmoid(np.dot(relu(np.dot(X_test, W1) + b1), W2) + b2)

```

```
    # Threshold the predictions using the mean of y_pred as the threshold
    M = y_pred.mean()
    for i in range(len(y_pred)):
        if y_pred[i] >= M:
            y_pred[i] = 1
        else:
            y_pred[i] = 0

    # Calculate the evaluation metrics
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    specificity = recall_score(1 - y_test, 1 - y_pred)
    f1 = f1_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    # Add the confusion matrix for this fold to the running total
    cm = confusion_matrix(y_test, y_pred)
    total_cm += cm

    # Append the performance metrics for this fold
    precisions.append(precision)
    recalls.append(recall)
    specificities.append(specificity)
    f1s.append(f1)
    accuracies.append(accuracy)

# Print the final confusion matrix
print("Confusion Matrix:")
print(total_cm)

tn, fp, fn, tp = total_cm.ravel()
# Calculate the evaluation metrics
accuracy = (total_cm[0][0] + total_cm[1][1]) / np.sum(total_cm)
precision = total_cm[1][1] / (total_cm[1][1] + total_cm[0][1])
recall = total_cm[1][1] / (total_cm[1][1] + total_cm[1][0])
specificity = total_cm[0][0] / (total_cm[0][0] + total_cm[0][1])
f1 = 2 * precision * recall / (precision + recall)

# Print the evaluation metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"Specificity: {specificity:.4f}")
```

```
print(f"F1 score: {f1:.4f}")

# Print the final weights and biases
print("W1: ")
print(W1)
print("b1: ")
print(b1)
print("W2: ")
print(W2)
print("b2: ")
print(b2)

# Print the average evaluation metrics across k-folds
print(f"Average Precision: {np.mean(precisions):.4f}")
print(f"Average Recall: {np.mean(recalls):.4f}")
print(f"Average Specificity: {np.mean(specificities):.4f}")
print(f"Average F1 score: {np.mean(f1s):.4f}")
print(f"Average Accuracy: {np.mean(accuracies):.4f}")

# Print the total time
t2 = time()
total_time = t2 - t1
print('Total Time', total_time)
```

RmsProp with Relu and Sigmoid Activation Function

```
# -*- coding: utf-8 -*-
"""Rmsprop+relu+sigmoid.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1
    LX_u3HysxNhIHL45l88pwrrz3V_MAjQD
"""

import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error, confusion_matrix,
    accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import KFold, train_test_split
from time import time
```

```
15
16 t1 = time()
17
18 # Define the neural network architecture
19 input_size = 9
20 hidden_size = 2
21 output_size = 1
22
23 # Define the ReLU activation function
24 def relu(x):
25     return np.maximum(0, x)
26
27 # Define the derivative of the ReLU activation function
28 def relu_derivative(x):
29     return np.where(x > 0, 1, 0)
30
31 # Define the sigmoid activation function
32 def sigmoid(x):
33     return 1 / (1 + np.exp(-x))
34 # Define the derivative of the sigmoid activation function
35 def sigmoid_derivative(x):
36     return sigmoid(x) * (1 - sigmoid(x))
37
38
39 # Define the loss function
40 def mse_loss(y_true, y_pred):
41     epsilon = 1e-5
42     y_pred = np.minimum(np.maximum(y_pred, epsilon), 1 - epsilon)
43     return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.
    log(1 - y_pred))
44
45 # Load the train and test datasets from CSV files
46 df = pd.read_csv('/content/CNN_40X.csv', header=None).values
47
48 # get the locations
49 X = df[:, :-1]
50 y = df[:, -1:]
51
52 # Set the learning rate and number of epochs
53 learning_rate = 0.01
54 epochs = 20
55 b = 0.9
56 eps = 1e-8
57
```

```
# Initialize the KFold object
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize performance metric lists for cross-validation
precisions = []
recalls = []
specificities = []
f1s = []
accuracies = []

# Initialize a running total for the confusion matrix
total_cm = np.zeros((2, 2))

# Iterate over the k-folds
for train_index, test_index in kf.split(X):
    # Split the data into train and test sets
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Initialize the weights and biases for each fold
    W1 = np.random.randn(input_size, hidden_size)
    b1 = np.random.randn(hidden_size)
    W2 = np.random.randn(hidden_size, output_size)
    b2 = np.random.randn(output_size)

    sw1 = 0
    sw2 = 0
    sb1 = 0
    sb2 = 0

    # Train the neural network using backpropagation and gradient
    descent
    for epoch in range(epochs):
        # Forward pass
        z1 = np.dot(X_train, W1) + b1
        a1 = relu(z1)  # Use ReLU activation in the first layer
        z2 = np.dot(a1, W2) + b2
        y_pred = sigmoid(z2)

        # Compute the loss and gradients
        loss = mse_loss(y_train, y_pred)
        delta2 = (y_pred - y_train) * sigmoid_derivative(z2)
```

```
100         delta1 = np.dot(delta2, W2.T) * relu_derivative(z1)  # Use
    ReLU derivative in the first layer
101         dW2 = np.dot(a1.T, delta2)
102         db2 = np.sum(delta2, axis=0)
103         dW1 = np.dot(X_train.T, delta1)
104         db1 = np.sum(delta1, axis=0)  # Sum over axis=0
105 
106         # Update the weights and biases using gradient descent
107 
108         sw2 = b*sw2 + (1-b)*dW2**2
109         W2 -= learning_rate * dW2/np.sqrt(sw2+eps)
110         sb2 = b*sb2 + (1-b)*db2**2
111         b2 -= learning_rate * db2/np.sqrt(sb2+eps)
112         sw1 = b*sw1 + (1-b)*dW1**2
113         W1 -= learning_rate * dW1/np.sqrt(sw1+eps)
114         sb1 = b*sb1 + (1-b)*db1**2
115         b1 -= learning_rate * db1/np.sqrt(sb1+eps)
116 
117         # Compute the loss and gradients
118         loss = mse_loss(y_train, y_pred)
119         if epoch % 5 == 0:  # Print mean squared error every 100
    epochs
120             print(f"Epoch {epoch}, Cross Entropy loss: {loss:.4f}"
    )
121 
122     # Predict the output for the test data using the trained
    neural network
123     y_pred = sigmoid(np.dot(relu(np.dot(X_test, W1) + b1), W2) +
    b2)
124 
125     # Threshold the predictions using the mean of y_pred as the
    threshold
126 
127     M = y_pred.mean()
128     for i in range(len(y_pred)):
129         if y_pred[i] >= M:
130             y_pred[i] = 1
131         else:
132             y_pred[i] = 0
133 
134 
135     # Calculate the evaluation metrics
136     precision = precision_score(y_test, y_pred)
137     recall = recall_score(y_test, y_pred)
```

```
    specificity = recall_score(1 - y_test, 1 - y_pred)
    f1 = f1_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    # Add the confusion matrix for this fold to the running total
    cm = confusion_matrix(y_test, y_pred)
    total_cm += cm

    # Append the performance metrics for this fold
    precisions.append(precision)
    recalls.append(recall)
    specificities.append(specificity)
    f1s.append(f1)
    accuracies.append(accuracy)

# Print the final confusion matrix
print("Confusion Matrix:")
print(total_cm)

tn, fp, fn, tp = total_cm.ravel()
# Calculate the evaluation metrics
accuracy = (total_cm[0][0] + total_cm[1][1]) / np.sum(total_cm)
precision = total_cm[1][1] / (total_cm[1][1] + total_cm[0][1])
recall = total_cm[1][1] / (total_cm[1][1] + total_cm[1][0])
specificity = total_cm[0][0] / (total_cm[0][0] + total_cm[0][1])
f1 = 2 * precision * recall / (precision + recall)

# Print the evaluation metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"Specificity: {specificity:.4f}")
print(f"F1 score: {f1:.4f}")


# Print the final weights and biases
print("W1: ")
print(W1)
print("b1: ")
print(b1)
print("W2: ")
print(W2)
print("b2: ")
print(b2)
```

```
182
183 # Print the average evaluation metrics across k-folds
184 print(f"Average Precision: {np.mean(precisions):.4f}")
185 print(f"Average Recall: {np.mean(recalls):.4f}")
186 print(f"Average Specificity: {np.mean(specificities):.4f}")
187 print(f"Average F1 score: {np.mean(f1s):.4f}")
188 print(f"Average Accuracy: {np.mean(accuracies):.4f}")
189
190 # Print the total time
191 t2 = time()
192 total_time = t2 - t1
193 print('Total Time', total_time)
```

Novel weight updation optimisation with Relu and Sigmoid Activation Function

```
1  # -*- coding: utf-8 -*-
2  """Novel algo_Relu+sigmoid.ipynb
3
4  Automatically generated by Colab.
5
6  Original file is located at
7      https://colab.research.google.com/drive/1
       LdyzYP702EstV7CIthdFYQ7o2RE4FhN6
8  """
9
10 import numpy as np
11 import pandas as pd
12 from sklearn.metrics import mean_squared_error, confusion_matrix,
       accuracy_score, precision_score, recall_score, f1_score
13 from sklearn.model_selection import KFold, train_test_split
14 from time import time
15
16 t1 = time()
17
18 # Define the neural network architecture
19 input_size = 9
20 hidden_size = 2
21 output_size = 1
22
23 # Define the ReLU activation function
24 def relu(x):
```

```
    return np.maximum(0, x)

# Define the derivative of the ReLU activation function
def relu_derivative(x):
    return np.where(x > 0, 1, 0)

# Define the sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Define the derivative of the sigmoid activation function
def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))

# Define the loss function
def mse_loss(y_true, y_pred):
    epsilon = 1e-5
    y_pred = np.minimum(np.maximum(y_pred, epsilon), 1 - epsilon)
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.
   log(1 - y_pred))

# Load the train and test datasets from CSV files
df = pd.read_csv('/content/CNN_40X.csv', header=None).values

# get the locations
X = df[:, :-1]
y = df[:, -1:]

# Set the learning rate and number of epochs
learning_rate = 0.01
epochs = 10
b = 0.08
eps = 1e-8

# Initialize the KFold object
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize performance metric lists for cross-validation
precisions = []
recalls = []
specificities = []
f1s = []
accuracies = []
```

```
68
69 # Initialize a running total for the confusion matrix
70 total_cm = np.zeros((2, 2))
71
72 # Iterate over the k-folds
73 for train_index, test_index in kf.split(X):
74     # Split the data into train and test sets
75     X_train, X_test = X[train_index], X[test_index]
76     y_train, y_test = y[train_index], y[test_index]
77
78     # Initialize the weights and biases for each fold
79     W1 = np.random.randn(input_size, hidden_size)
80     b1 = np.random.randn(hidden_size)
81     W2 = np.random.randn(hidden_size, output_size)
82     b2 = np.random.randn(output_size)
83
84     sw1 = 0
85     sw2 = 0
86     sb1 = 0
87     sb2 = 0
88
89     # Train the neural network using backpropagation and gradient
   descent
90     for epoch in range(epochs):
91         # Forward pass
92         z1 = np.dot(X_train, W1) + b1
93         a1 = relu(z1)  # Use ReLU activation in the first layer
94         z2 = np.dot(a1, W2) + b2
95         y_pred = sigmoid(z2)
96
97         # Compute the loss and gradients
98         loss = mse_loss(y_train, y_pred)
99         delta2 = (y_pred - y_train) * sigmoid_derivative(z2)
100        delta1 = np.dot(delta2, W2.T) * relu_derivative(z1)  # Use
    ReLU derivative in the first layer
101        dW2 = np.dot(a1.T, delta2)
102        db2 = np.sum(delta2, axis=0)
103        dW1 = np.dot(X_train.T, delta1)
104        db1 = np.sum(delta1, axis=0)  # Sum over axis=0
105
106        # Update the weights and biases using gradient descent
107        sw2 = b * abs(dW2**2 / (np.log(abs(dW2))))
108        W2 -= learning_rate * dW2 / np.sqrt(sw2 + eps)
109        sb2 = b * np.exp(abs(db2))
```

```
        b2 -= learning_rate * db2 / np.sqrt(sb2 + eps)
        sw1 = b * abs(dW1**2 / (np.log(abs(dW1))))
        W1 -= learning_rate * dW1 / np.sqrt(sw1 + eps)
        sb1 = b * np.exp(abs(db1))
        b1 -= learning_rate * db1 / np.sqrt(sb1 + eps)

        # Compute the loss and gradients
        loss = mse_loss(y_train, y_pred)
        if epoch % 5 == 0:  # Print mean squared error every 100 epochs
            print(f"Epoch {epoch}, Cross Entropy loss: {loss:.4f}")

    # Predict the output for the test data using the trained neural network
    y_pred = sigmoid(np.dot(relu(np.dot(X_test, W1) + b1), W2) + b2)

    # Threshold the predictions using the mean of y_pred as the threshold
    M = y_pred.mean()
    for i in range(len(y_pred)):
        if y_pred[i] >= M:
            y_pred[i] = 1
        else:
            y_pred[i] = 0

    # Calculate the evaluation metrics
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    specificity = recall_score(1 - y_test, 1 - y_pred)
    f1 = f1_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    # Add the confusion matrix for this fold to the running total
    cm = confusion_matrix(y_test, y_pred)
    total_cm += cm

    # Append the performance metrics for this fold
    precisions.append(precision)
    recalls.append(recall)
    specificities.append(specificity)
    f1s.append(f1)
    accuracies.append(accuracy)
```

```
149
150 # Print the final confusion matrix
151 print("Confusion Matrix:")
152 print(total_cm)
153
154 tn, fp, fn, tp = total_cm.ravel()
155 # Calculate the evaluation metrics
156 accuracy = (total_cm[0][0] + total_cm[1][1]) / np.sum(total_cm)
157 precision = total_cm[1][1] / (total_cm[1][1] + total_cm[0][1])
158 recall = total_cm[1][1] / (total_cm[1][1] + total_cm[1][0])
159 specificity = total_cm[0][0] / (total_cm[0][0] + total_cm[0][1])
160 f1 = 2 * precision * recall / (precision + recall)
161
162 # Print the evaluation metrics
163 print(f"Accuracy: {accuracy:.4f}")
164 print(f"Precision: {precision:.4f}")
165 print(f"Recall: {recall:.4f}")
166 print(f"Specificity: {specificity:.4f}")
167 print(f"F1 score: {f1:.4f}")
168
169
170 # Print the final weights and biases
171 print("W1: ")
172 print(W1)
173 print("b1: ")
174 print(b1)
175 print("W2: ")
176 print(W2)
177 print("b2: ")
178 print(b2)
179
180 # Print the average evaluation metrics across k-folds
181 print(f"Average Precision: {np.mean(precisions):.4f}")
182 print(f"Average Recall: {np.mean(recalls):.4f}")
183 print(f"Average Specificity: {np.mean(specificities):.4f}")
184 print(f"Average F1 score: {np.mean(f1s):.4f}")
185 print(f"Average Accuracy: {np.mean(accuracies):.4f}")
186
187 # Print the total time
188 t2 = time()
189 total_time = t2 - t1
190 print('Total Time', total_time)
```

In a similar manner, the aforementioned program can be applied to Sigmoid activation, Tanh activation, Leaky Sigmoid, and Exponential ReLU combined with Sigmoid.